CS 595 Advanced Scientific Computing

Course Project

Using Freeman Chain Code to process a tumor image

By

Zhi Su & Zhiwei Zhang

12/05/2018

Abstract

Last few decades, human made huge progress on developing high quality camera lens. As a result, people could take high quality photos anywhere, anytime with their cameras, cell phones or even tablet computers. Normally, high quality image means more memory to storage, more time to transmit (upload or download). Chain code could be a perfect way to lossless compress the monochrome image for efficient transmission and storage. In this paper, we will present how to use one of the most popular chain codes, Freeman Chain Code, to process a given monochrome tumor image.

Introduction

1. Introduction of Freeman Chain Code

A chain code is a lossless compression algorithm for monochrome image. Generally, how chain code works is to separate a whole image to several connected components and process each component separately.

Freeman Chain Code is one of the most popular chain codes methods. Figure 1 shows two ways to realize Freeman Chain Code: 4 directions and 8 directions.

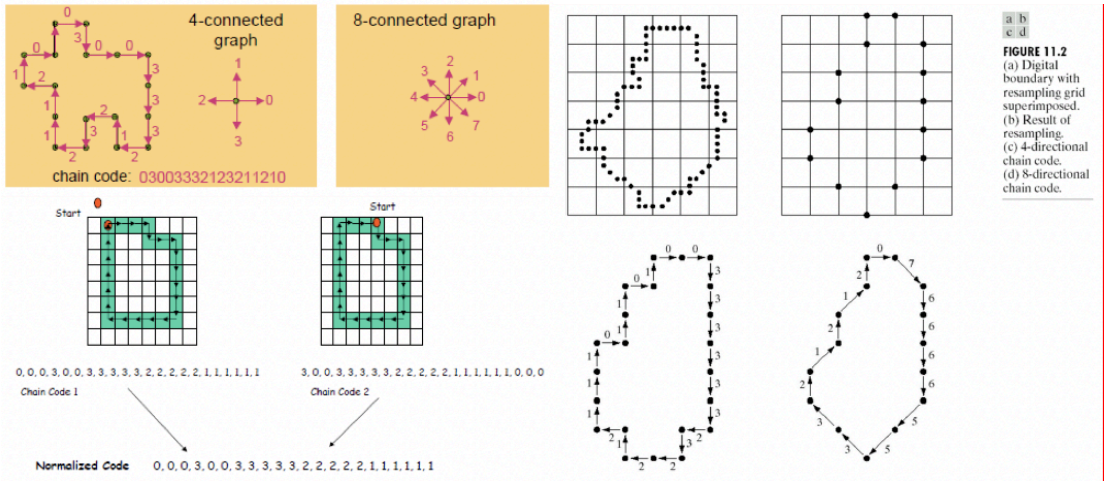


Figure 1. 2 kinds of Freeman Chain Code and how to realize them.

2. Procedure for our project

Out object is a raw 570 *570 image of a circular stroke embedded in specular noise shown in Figure 2 (a). The objective of the project is to obtain the Freeman chain code, the first difference of the outer boundary of the largest object, and the integer of minimum magnitude of the code.

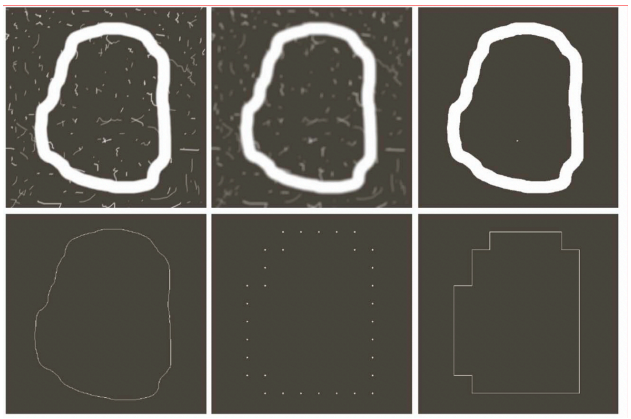


Figure 2. Procedure to get the Freeman Chain Code

The procedure of the project is shown in Figure 2.

(a). Original noisy image.

(b). Image smoothed with a 9*9 averaging (or Gaussian) mask filter matrix.

Smoothing data is the section we decide to use PETSc lib and may apply parallel computation in the future.

The mathematical model for this part is to use a 9*9 filter (average, Gaussian or kernel) matrix to multiply a 9*9 matrix extract from the image for every pixel as the center point and refill the diagonal summation to the pixel.

$$V = \left| \frac{\sum_{i=1}^{} \left(\sum_{j=1}^{} f_{ij} d_{ij} \right)}{F} \right|$$

Where:

- f_{ij} = the coefficient of a convolution kernel at position i,j (in the kernel)
- d_{ij} = the data value of the pixel that corresponds to f_{ij}
- q = the dimension of the kernel, assuming a square kernel (if $q = 3$, the kernel is 3×3)
- F = either the sum of the coefficients of the kernel, or 1 if the sum of coefficients is 0
- V = the output pixel value

In cases where V is less than 0, V is clipped to 0.

(c). Smoothed image, thresholded using Otsu's method.

(d). Longest outer boundary.

(e). Subsampled boundary (the points are shown enlarged for clarity).

(f). Connected points from subsampled boundary.

Then we can get the Freeman Chain Code and display the output.

Previous work

1. Moves to realize the project

(a). Project selection.

Since we both have qualifying exam so we did not make a lot effort on finding a project at the beginning of the semester. When we know this project, we thought handling computing matrices suits the goal to apply parallel computation.

(b). Fully understand the problem.

We applied PETSc lib, but our project is neither linear nor non-linear. So, we did not use ksp or snes packages.

(c). Task distribution.

In introduction, we set a list for all jobs, we separate the job clearly. Zhi worked on read data, smooth image, translate to binary number, exact the boundary and eliminate the inner boundary. Zhiwei worked on building the filter (average filter and Gaussian filter), subsample the outer boundary and connect the subsampled points. We both responsible for explain the Chain Code knowledge for our own job to each other.

After C code job was done, we use Atom to work together on applying PETCs lib.

(d). Learn knowledge about Freeman chain coding and relevant algorithms.

At the beginning, we tried to use Matlab code as the reference for learning Freeman Chain Code. However, it did not help. Then we learn the knowledge in course PPT and notes.

(e). Write C code.

Since it is a team work, we need to select a language that we both are familiar with. We both have some basic understanding for C code. So, we decide to use C language.

At first, we would like to transfer the Matlab code (Provided by another student Qinbo Zhang and ECE 565 instructor Dr. Joohee Kim). However, every function in Matlab code contains more unknown functions or packages. So, we decide to write our own code in C without understanding and transferring the Matlab code.

(f). Test C code using a 25*25 small sample.

Our objective image is 570*570. It is relative large for us to test the C code. Which means when we display or output the result, it is hard to tell if the result is good or not. So, we build a simple 13*13 matrix to test the result. Furthermore, we use #define to set the value of the rows and column number so we could change it from 13 to 570.

0	0	0	0	0	0	50	0	0	0	0	0	0
0	0	0	0	0	0	50	0	0	0	0	0	0
0	0	0	0	0	50	50	50	0	0	0	0	0
0	0	0	0	0	50	50	50	0	0	0	0	0
0	0	0	0	50	50	50	50	50	0	0	0	0
0	0	0	50	50	50	50	50	50	50	0	0	0
50	50	50	50	50	50	50	50	50	50	50	50	50
0	0	0	50	50	50	50	50	50	50	0	0	0
0	0	0	0	50	50	50	50	50	0	0	0	0
0	0	0	0	0	50	50	50	0	0	0	0	0
0	0	0	0	0	50	50	50	0	0	0	0	0
0	0	0	0	0	0	50	0	0	0	0	0	0
0	0	0	0	0	0	50	0	0	0	0	0	0

Figure 3. Sample image (presented with RGB value).

(g). Test and improve the code using objective picture.

After finishing sample test, we use the 570*570 image and make a comparison for the result from Matlab code. The result will show in Results and Analysis section.

2. Toolbox or software used in project

(a). Git hub:

(1). The version control functionality helps us to keep good track of all our previous codes and our changes to it. It releases our burden on worrying about messing up with lines of code because we can always return to the previous version.

(2). It helps our whole group, as well as Professor Zhang, to help with the codes without interfering with each other's work when each of us is working under different branches.

(b). Atom

(1). Functionality: Collaborate in real time in Atom.

Teletype introduces the concept of real-time "portals" for sharing workspaces. When a host opens a portal, their active tab becomes a shared workspace. There, invited collaborators can join in and make edits in real time. As the host moves between files, collaborators follow along with the active tab automatically.

(2). How we used it:

We share our files when we have some when we need to make some important changes to our code. At the same time, we talk to each other via phones or face to face. We can work individually on different sections after the tasks are distributed. We can also instantly track what others are doing using the built-in functionality. One can check the code while the other type. It also reduced the amount of time for debugging.

(c). Ulimit and top command in linux:

Ulimit and top command: as a method of monitor CPU and memory availability.

Ulimit command is the one to set a limit to the maximum usage to prevent occupying too muck memory. As our program involves a lot of small dense matrices multiplication the dimension is set up high. We encounter the problem that the program is getting slower and slower. So, we used the top command in Linux to monitor out usage of CPU and memory. Which actually tells us some possible reason for our gradually slower program. Based on the observed data, we then use ulimit to set a limit on the maximum usage of the virtual memory.

As 100% CPU is used all the time when the program is running and the program occupied more that 83% of the whole memory, we guess a higher CPU capacity will help us with the efficiency of the program.

(d). The comparison with branch merge functionality from github:

(1). Better communication about the updates.

(2). Reduce the possibility of occurrence of bugs.

(3). About us: We both are just starting to use PETSc and not very familiar with C. Coding while discussion reduces time and energy for us to understand how a certain function in PETSc was used or how to build a thing that satisfy our needs in C.

(e). PETSc library related:

(1).We use the codes to establish matrices and set up values in each matrix. After assembling all the matrices, we used the "Matmatmult()" function from PETSc library to

carry out all the matrix multiplications involved. Inside the whole smooth function, whenever we use the function from PETSc library, we use "CHKERRQ()" to check the error codes.

(2). PETSc user manual:

Another thing we used most is the PETSc User manual. It helps us to understand functions in PETSc with examples.

Results and Analysis

1. Results obtained from different code or method:

(a). Matlab code

The original code is Matlab code, so we get the Matlab result at the beginning. Table 1 shows the subsample of the image obtained from Matlab.

Table 1. Matlab subsample result

Row index	Column index	Row index	Column index	Row index	Column index	Row index	Column index
52	205	103	460	307	103	460	460
52	256	154	154	307	460	511	154
52	307	154	460	358	103	511	205
52	358	205	103	358	460	511	256
52	409	205	154	409	103	511	307
103	154	205	460	409	460	511	358
103	205	256	103	460	103	511	409
103	409	256	460	460	154	511	460

The Freeman chain code obtained from Matlab is:

FCC: [0 0 0 0 3 0 3 3 3 3 3 3 3 3 2 2 2 2 2 2 1 2 1 1 1 1 1 0 1 1 0 1]

(b). C code result (with or without PETSc).

We write our own C code and get as the reference answer. Then we apply PETSc lib and get the exactly same result as the reference answer.

Table 2. C code subsample result
(Same result for with or without PETSc)

Row index	Column index	Row index	Column index	Row index	Column index	Row index	Column index
50	200	100	450	300	100	450	450
50	250	150	150	300	450	500	150
50	300	150	450	350	100	500	200
50	350	200	100	350	450	500	250
50	400	200	150	400	100	500	300
100	150	200	450	400	450	500	400
100	200	250	100	450	100	500	450
100	400	250	450	450	150	550	300

						550	350
						550	400

The Freeman chain code obtained from C code is:

FCC: [0 0 0 0 3 0 3 3 3 3 3 3 3 3 2 3 2 2 1 2 2 2 1 2 1 1 1 1 1 0 1 1 0 1]

After make comparison between Matlab result and C code result. We believe we get the correct answer. (I will explain why there are some difference in analysis section). So, I use this result to test if we apply PETSc correctly.

2. Analysis and comparison

(a). Compare between Matlab and C codes:

Since every function in Matlab code contains more unknown functions or packages. So, we decide to write our own code in C without understanding and transferring the Matlab code. We also believe that we have got the correct answer at least for this image. However, there are still some little difference.

We could see that C code subsample result is the multiple of 50 and Matlab code subsample result is the multiple of 51 plus 1. This is because C code selects point on boundary every 50 pixels and Matlab code set every 51 pixels. And that is also the reason why there are 2 more points selected in C code. Because row index in Matlab code is 511+51=565 and that is the boundary of the image and there is nothing but row index 550 has point.

Then the 2 more points leads to 2 more number in Freeman Chain Code.

(b). Compare between C code and Matlab code ideas;

For extracting the out boundary Matlab code and C code have different algorithm.

In Matlab code, it first finds a point on outer boundary and then find another one colosed to it. For C code, we settle every row index and find out all points that is the boundary. So, in this way, we got 2 boundaries: inner boundary and outer boundary.

(c). Compare between PETSc code without PCP algorithm and original C:

To check if codes with PETSc setup and without PETSc setup, Figure 4 shows the results of smooth function for both of them. Two figures are shown. In each figure, the one on the left is the one with PETSc setup. The one on the right is the one without PETSc setup.

7	360	0.000000	7	360	0	8	359	0.000000	8	358	0
7	361	0.000000	7	361	0	8	360	0.000000	8	359	0
7	362	0.000000	7	362	0	8	361	0.000000	8	360	0
7	363	0.000000	7	363	0	8	362	0.000000	8	361	0
7	364	0.000000	7	364	0	8	363	0.000000	8	362	0
7	365	0.061728	7	365	0	8	364	0.000000	8	363	0
7	366	2.358025	7	366	2	8	365	0.061728	8	364	0
7	367	7.679012	7	367	7	8	366	2.358025	8	365	0
7	368	15.049383	7	368	15	8	367	7.679012	8	366	2
7	369	22.740741	7	369	22	8	368	14.370370	8	367	7
7	370	30.395062	7	370	30	8	369	21.308642	8	368	14
7	371	37.975309	7	371	37	8	370	28.567901	8	369	21
7	372	45.530864	7	372	45	8	371	36.086420	8	370	28
7	373	52.888889	7	373	52	8	372	43.641975	8	371	36
7	374	59.802469	7	374	59	8	373	51.000000	8	372	43
7	375	64.185185	7	375	64	8	374	57.913580	8	373	51
7	376	65.432099	7	376	65	8	375	62.296296	8	374	57
7	377	64.604938	7	377	64	8	376	63.543210	8	375	62
7	378	63.456790	7	378	63	8	377	63.395062	8	376	63
7	379	61.333333	7	379	61	8	378	63.000000	8	377	63
7	380	56.938272	7	380	56	8	379	61.271605	8	378	63
7	381	50.135802	7	381	50	8	380	56.938272	8	379	61
7	382	42.790123	7	382	42	8	381	50.135802	8	380	56
7	383	35.814815	7	383	35	8	382	42.790123	8	381	50
7	384	29.135802	7	384	29	8	383	35.814815	8	382	42
7	385	22.567901	7	385	22	8	384	29.135802	8	383	35
7	386	16.024691	7	386	16	8	385	22.567901	8	384	29
7	387	9.481481	7	387	9	8	386	16.024691	8	385	22
7	388	3.950617	7	388	3	8	387	9.481481	8	386	16
7	389	0.765432	7	389	0	8	388	3.950617	8	387	9
7	390	0.012346	7	390	0	8	389	0.765432	8	388	3
7	391	0.000000	7	391	0	8	390	0.012346	8	389	0
7	392	0.000000	7	392	0	8	391	0.000000	8	390	0
7	393	0.000000	7	393	0	8	392	0.000000	8	391	0
7	394	0.000000	7	394	0	8	393	0.000000	8	392	0
7	395	0.000000	7	395	0	8	394	0.000000	8	393	0

Figure 4. Comparison of the smoothed data

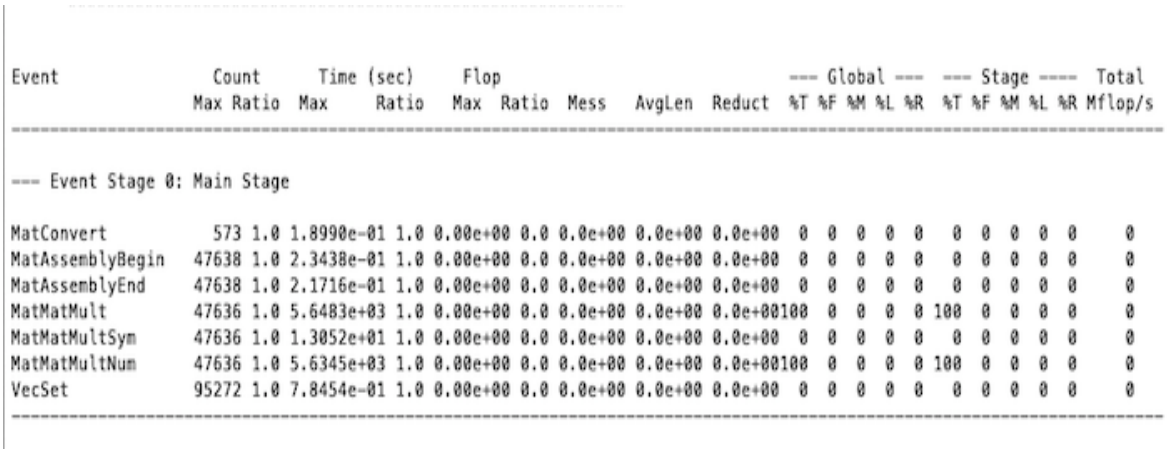
(d). Time result of PETSc code with Preselection of the candidate points (PCP) algorithm:

Before presenting the result, I need to emphasize our designed algorithm to reduce the computation work: (Since by utilizing this algorithm, we could at least get a result without being interrupted by "out of memory").

Preselection of the candidate points (PCP method):

Due to the limit of CPU and memory, we are unable to finish running our program on Fourier platform at the beginning. So, we developed an algorithm to pre-select the points that needs to be smoothed using the matrix multiplication method with "Matmatmult()" function. The details are as the following: as there are many empty spaces around our graph, we decided to set a bar so that if the value of the points falls below that bar, the indexes of that point will be eliminated out of the loop for the smooth function. For example, for an n by n filter, we set the bar to be 0.5*RGB*n2. (RGB is the maximum RGB value=256).

We also explored the time used by each function in the in the PETSc version with PCP algorithm with log_view time section as Figure 5:



```
Event                Count      Time (sec)     Flop                              --- Global ---  --- Stage ----  Total
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   AvgLen  Reduct  %T %F %M %L %R  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------------------------

--- Event Stage 0: Main Stage

MatConvert           573 1.0 1.8990e-01 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatAssemblyBegin   47638 1.0 2.3438e-01 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatAssemblyEnd     47638 1.0 2.1716e-01 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatMatMult         47636 1.0 5.6483e+03 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00100  0  0  0  0 100  0  0  0  0     0
MatMatMultSym      47636 1.0 1.3052e+01 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
MatMatMultNum      47636 1.0 5.6345e+03 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00100  0  0  0  0 100  0  0  0  0     0
VecSet             95272 1.0 7.8454e-01 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0
------------------------------------------------------------------------------------------------------------------------
```

Figure 5. Log view

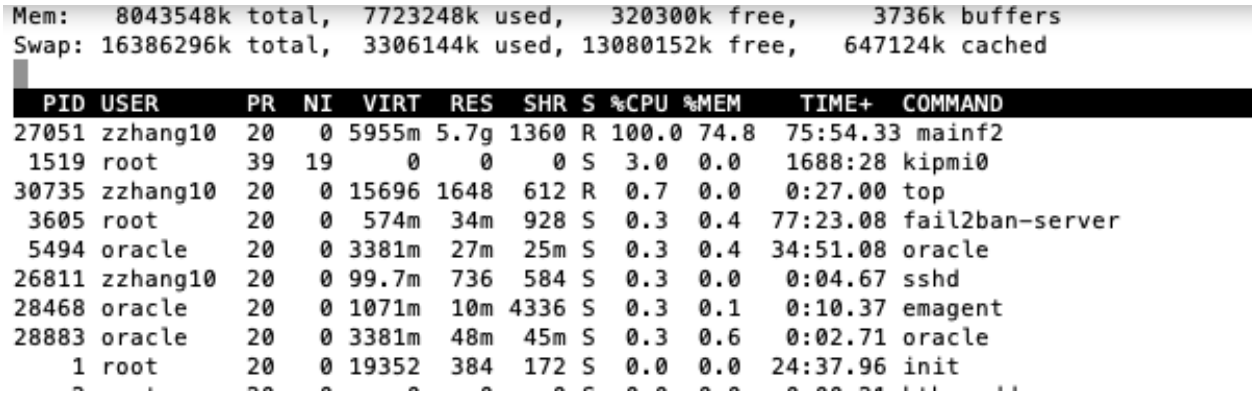
We can see that almost all the time are used by the "Matmatmult()" function.

By comparison, without PCP algorithm, we can get the result matching the one produced by original C code. However, we can't finish the program with the computational resources given by Fourier platform.

(e). Memory and CPU usage detected with Top and Ulimt command

The following are the status of CPU and memory usage while we are running PETSc version codes with PCP algorithm shown by top command.

The following are the status of CPU and memory usage while we are running PETSc version codes with PCP algorithm shown by top command. We can clearly see the shortage of CPU capacity and memory.



```
Mem:   8043548k total,  7723248k used,   320300k free,     3736k buffers
Swap: 16386296k total,  3306144k used, 13080152k free,   647124k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
27051 zzhang10  20   0 5955m 5.7g 1360 R 100.0 74.8  75:54.33 mainf2
 1519 root      39  19     0    0    0 S  3.0  0.0   1688:28 kipmi0
30735 zzhang10  20   0 15696 1648  612 R  0.7  0.0   0:27.00 top
 3605 root      20   0  574m  34m  928 S  0.3  0.4  77:23.08 fail2ban-server
 5494 oracle    20   0 3381m  27m  25m S  0.3  0.4  34:51.08 oracle
26811 zzhang10  20   0 99.7m  736  584 S  0.3  0.0   0:04.67 sshd
28468 oracle    20   0 1071m  10m 4336 S  0.3  0.1   0:10.37 emagent
28883 oracle    20   0 3381m  48m  45m S  0.3  0.6   0:02.71 oracle
    1 root      20   0 19352  384  172 S  0.0  0.0  24:37.96 init
```

Conclusion and Future work

1. Conclusion

During the whole process of the project. We learned a lot about C, Linux, MPI, and PETSc library. We are very glad that with the knowledge we learned from class and internet, with the help of Dr. Zhang, We are able to achieve most of our goals that we set up at the beginning and create a good algorithm to enable our PETSc-built C program to work.

2. Future work

(a). First thing we will do is to use the “MatCreateSubMatrices” function instead of the current way of fill in values. Reason is that it may save the assembling time though the assembling time may not be the major factor of time-consuming.

(b). The second thing we will do is to parallelize the smooth part. With multiple process running simultaneously, we believe that the CPU usage per processes will be reduced to a large extend, thus enhancing the speed of the whole program.

(c). As our matrix multiplications are all multiplications of dense matrices, we would like to explore a way to optimize it. Either by checking if PETSc has smart way of handling it or we write something by ourselves. Other way, we at least get a better understanding of the PETSc library. This will be followed up by a detailed comparison between the performance of the parallel codes compared with the sequential codes.

(d). Our current algorithm on the smoothing are brought up by ourselves. it works very well on convex shapes. However, it might fail on some special type of shapes. We will be testing around different type of picture shapes and try to make changes to our algorithms for each part so it can handle different shapes that we might see in real life.

(e). We can see that almost all the time are used by the “Matmatmult()” function. So we are wondering if there is a better way to handle the dense matrices multiplication.

References

[1]. Frank Kelle, "Computational Foundations of Cognitive Science, Convolutions and Kernel".

[2]. Dr. Joohee Kim, “Computer Vision and Image Processing Representation and Description (MATLAB Programming).” ECE 565 Image Processing, Spring 2018

[3]. Introduction to atoms Teletype function: https://teletype.atom.io/

[4]. Wikipedia, “Chain code:” https://en.wikipedia.org/wiki/Chain_code