國 立 中 正 大 學

資訊工程研究所

碩士論文

應用機器學習方法於RGB影像之高光譜建置及其程式碼之優化

RGB-to-hyperspectral image reconstruction using machine-learning approach and its programming optimizations

指導教授 陳鵬升 博士

研究生 劉柏豪

中華民國一百一十年六月

致謝

感謝陳鵬升教授在碩士這段期間的指導， 老師平時待人非常親切且常常會關心我們的課業狀況倍感窩心， 在研究指導上教授也給予一些寶貴的見解， 從不同角度去解決研究上的問題，在碩士短短兩年間也累積無數上台報告經驗， 這些歷練著實提升我在學術研究上的信心和動力。

感謝張榮貴教授以及涂嘉恒教授擔任我的口試委員，在口試時給予指導和寶貴的建議 ，讓本論文能夠更加的完善。

感謝實驗室夥伴們，在碩士兩年中一起研究，課餘也能抽空一起出外旅遊， 很慶幸能遇到大家讓這段期間能增添許多珍貴的回憶， 也感謝中正排球小夥伴們在課餘時間打排球運動一下真的非常紓壓。 感謝這段時間大家互相扶持鼓勵，在未來期望大家都能往中心理想的生活邁進。

最後，感謝家人在漫長求學期間對我的支持，你們的支持都是我前進的動力 ，也謝謝你們讓我無後顧之憂地完成我的碩士學業。

摘要

高光譜成像技術獲取的光譜範圍和經典的RGB像素結構相比更廣，該技術顯示的訊息包括肉眼不可見的豐富訊息。然而高光譜重建演算法需要大量的計算和記憶體資源，這個問題阻礙了高光譜技術的廣泛應用。本論文從程式設計角度，旨在以合理的記憶體資源提高高光譜重建演算法的執行時間。首先，將現有的MATLAB高光譜演算法實現轉換爲相應的C++程式碼。在手動處理堆疊記憶體管理和利用外部開源庫OpenBLAS的幫助下，與原始C++版本相比，執行時間實現了4.87倍的加速。其次，提出了四次多項式回歸來預測光譜反射率值。多種程式優化技術(像是，共同子表達式消除、多執行緒和單指令流多資料流) 用於改善其執行時間。對於測試基準與原高光譜重建演算法相比，所提算法平均準確率達97%與翻譯的原始C++版本相比，執行時間可以實現8.7倍的加速和66%的記憶體減少。

關鍵詞：Hyperspectral imaging, OpenBLAS, Polynomial regression, Common subexpression elimination, Multithreading, SIMD

Abstract

Hyperspectral imaging is a technique that collects a broad spectrum of light compared to the classic RGB pixel structure. The information revealed by the technique contains rich information that is invisible to the naked eye. However, the RGB-to-hyperspectral image reconstruction algorithm requires massive computation and memory resources. This issue hinders the hyperspectral technique from wide applications. This thesis, in the view of programming, aims at enhancing the execution time of the hyperspectral reconstruction algorithm with reasonable memory resources. First, the existing MATLAB implementation is translated to the corresponding C++ code. With the help of manually handling heap-memory management and leveraging external open-source library OpenBLAS, the execution time achieves 4.87 times speedup compared to the original C++ version. Second, four-degree polynomial regression is proposed to predict the spectral reflectance values. Several program optimization techniques (i.e., common-subexpression elimination, multithreading, and SIMD) are used to improve its execution time. For the tested benchmarks, the average accuracy of the proposed algorithm reaches 97% compared with the original hyperspectral reconstruction algorithm. The execution time can achieve 8.7 times speedup and 66% memory-usage reduction compared to the original C++ program translated.

Keyword: Hyperspectral imaging, OpenBLAS, Polynomial regression, Common subexpression elimination, Multithreading, SIMD

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Hyperspectral imaging (HSI) [1] is a technique that collects a broad spectrum of light compared to the classic RGB pixel structure. The information revealed by the technique contains rich information that is invisible to the naked eye. In today's Taiwanese healthcare environment, a doctor has hundreds of patients in outpatient clinics every day. This harsh environment, easily causes fatigue diagnosis, leading to misjudgment of the patient's disease. According to the National Development Council, there will be more than $1/4$ of the population over 65 years old [2] after 2032. The aging population increases the chance of going to the hospital; medical care continues to be in short supply. If the current high medical care standards are used without promoting structural changes, the health care system may collapse. McKinsey [3] points out the direction of transformation. It is expensive to manage so many patients. The medical system must transform from incidental treatment to preventive and long-term care management. According to the National Development Council [4], a survey of American primary medical doctors found that nearly 75% of doctors said that information technology applications could reduce error incidents. HSI [1] has the characteristics of high precision, easy operation, and non-invasiveness and determines the stage of the disease according to the changing trend of the spectral characteristics. Let the patient smarter medical treatment to alleviate the shortage of healthcare issues and allow clinicians to stage the disease quickly.

1.1 Motivation

Considering HSI is to be used in the actual medical situation, it is necessary to establish a large number of databases and train a model to determine the specific disease cycle through the neural network, and finally place the model on an embedded system that can be equipped with a camera lens. However, the RGB-to-HSI reconstruction algorithm requires massive computation and memory resources. This issue hinders the hyperspectral technique from wide application. This thesis, in the view of programming, aims at enhancing the execution time of the hyperspectral reconstruction algorithm with reasonable memory resources. The original RGB-to-HSI reconstruction algorithm was implemented by MATLAB [5], which converts the RGB value of each pixel to the corresponding spectral reflectance. However, the program cannot be smoothly ported to embedded systems, and the algorithm requires massive computational and memory resources. Therefore, the existing MATLAB implementation is translated to the corresponding C++ code. With the help of manually handling heap-memory management and leveraging external open-source library OpenBLAS, the execution time achieves 4.87 times speedup compared to the original C++ version. In addition, four-degree polynomial regression is proposed to predict the spectral reflectance values. Several program optimization techniques (i.e., common-subexpression elimination, multithreading, and SIMD) are used to improve its execution time. For the tested benchmarks, the average accuracy of the proposed algorithm reaches 97% compared with the original HSI reconstruction algorithm; the execution time can achieve 8.7 times speedup and 66% memory-usage reduction compared to the original C++ program translated.

1.2 Contribution

In this thesis, we make the following contributions.

1. Translate the RGB-to-HSI reconstruction algorithm from MATLAB to C++ code and optimize its performance by 4.8 times, memory usage reduced to 33%.

2. Use the polynomial regression to predict spectral reflectance instead of the original HSI algorithm. The prediction accuracy achieves 97% on average and 87% in the worst case compared to the original algorithm. Several

program optimization techniques are used to improve the execution time. The optimized version achieves 8.7 times speedup and reduces the memory usage by 66% compared to the original translated C++ program.

3. Provide an optimization script for polynomial regression-related programs.

1.3 Related works

Armin Schneider [6] proposed that Hyperspectral imaging provides additional diagnostic information that can be exploited in various different ways. It can aid in the discrimination between healthy and malignant tissue. Furthermore, automated detection of pathologies is possible. A vast number of potential applications have been shown for this novel imaging technology—for instance, the accuracy rate of tongue tumor detection with computer-aided HSI is 96.5%.

Huang [1] proposed to use hyperspectral imaging technology to determine the early lesions of esophageal cancer. In this study, an endoscopic image system for distinguishing esophageal cancer was developed based on hyperspectral technology. This system distinguishes patients in Stages of esophageal cancer, the test results obtained PPV about 89.6%, NPV of 91.5%, and the time is shorter than that of endoscopists.

Chang [7] proposed HSI technology to judge the periodontal disease cycle of patients. This study shows that the gingival in the wavelength range (620-750nm) will decrease the reflectance due to the expansion of blood vessels, which is consistent with the trend of periodontal tissue deterioration. Through analyzing the spectrum, the band can be used for gingival imaging with unknown inflammation level, and the recognition accuracy is 80-90%.

Yao [8] proposed the diagnosis of diabetic retinopathy through HSI technology. Diabetic retinopathy is called DR. The stage of DR is judged by comparing the spectral characteristics of arteries in the retinal image under the ophthalmoscope with the oxygen saturation curve. The research indicates the accuracy of the judgment. Accuracy is higher than 91% in various disease stages.

Kao [9] retarget the ILC program from the original MATLAB code to C++ code. The SIMD and multi-threading techniques are used to enhance the performance of the translated C++ code. The experimental results show 20.8 times

speedup on total execution time compared to the MATLAB Ubuntu version.

Chen [10] develop compiler optimization to analyze the polynomial expressions at the source level and automatically optimize the codes. The polynomial expression with large components often appears in motion control-related algorithms. The computation significantly affects the efficiency. The proposed algorithm can achieve about 12.2 times speedup at most compared to the original execution.

Considering that HSI is the application of actual medical conditions that require real-time identification, however, the RGB-to-HSI reconstruction algorithm requires a lot of computing and memory resources. This problem hinders the widespread application of hyperspectral technology. From the perspective of program design, our work aims to increase the execution time of the spectral reconstruction algorithm with reasonable memory resources, and finally place the model on an embedded system that can be equipped with a camera lens.

1.4 Organization

The rest of this thesis is organized as follows. Chapter 2 introduces the RGB-to-HSI reconstruction algorithm. It also describes how to translate the original MATLAB implementation to the optimized C++ code. Chapter 3 introduces the polynomial regression to replace the HSI algorithm. Optimization techniques applied for optimizing programs are explained in detail. Finally, the conclusion is described in Chapter 4.

Chapter 2

RGB-to-hyperspectral image reconstruction

2.1 Hyperspectral imaging

The flow chart of using HSI technology to estimate the average spectrum of the source tissue is shown in Figure 2.1. It is divided into the spectrophotometer, camera, and the parts of simulating spectral.

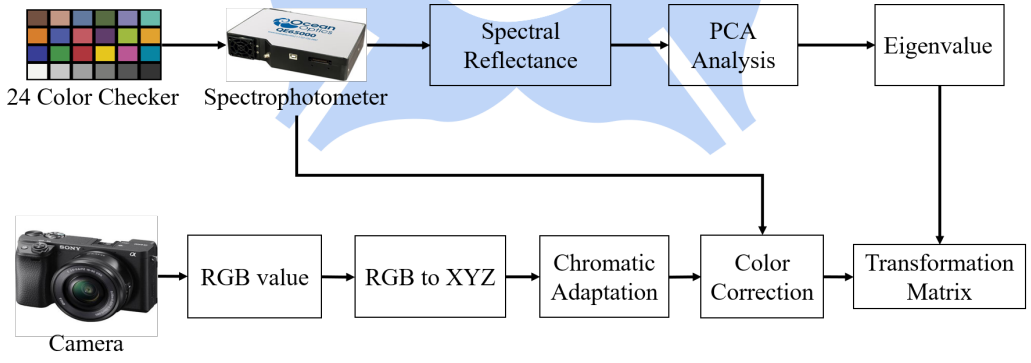

Figure 2.1: Schematic diagram of using a camera to estimate the spectral reflectance of each pixel of the image.

2.1.1 Spectrophotometer of HSI

Use a spectrophotometer (Ocean Optics, QE65000) under uniform artificial illumination, and Obtain the spectral reflectance of the 24 color checker. The arrangement of these spectra is stored as a matrix $[D]_{401\times24}$, where each row is the wavelength intensities separated by 1 nm, and each column represents the color. Through the eigen system and PCA, the six eigenvectors select the matrix with the largest contribution as the basis of the estimation as $[EV]_{6\times401}$. Eigenvalues $[\alpha]_{6\times24}$ corresponding to these six eigenvectors can be determined as Equation 2.1, where "pinv" means that the matrix is pseudo-inverse.

$$[\alpha] = [D]^T pinv[EV] \tag{2.1}$$

2.1.2 Camera of HSI

At the camera part, the output format of the camera under the same light source is RGB format. Each of the red, green, and blue (from 0 to 255) in the color checker is the ratio obtained by calculation and is based on Rsrgb, Gsrgb, Bsrgb (from 0 to 1) scale drawing. The RGB values can be converted into CIE XYZ stimulus values by the following formula : (Equation 2.2 $\sim$ 2.4)

$$[T] = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \tag{2.2}$$

$$f(n) = \begin{cases} \left(\dfrac{n+0.055}{1.055}\right)^{2.4} & , n > 0.04045 \\ \dfrac{n}{12.92} & , otherwise. \end{cases} \tag{2.3}$$

Where

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [T] \begin{bmatrix} f(Rsrgb) \\ f(Gsrgb) \\ f(Bstgb) \end{bmatrix} \tag{2.4}$$

Since the reference white light of the sRGB color space is D65 (standard daylight illuminant), the artificial light in different color check is measured. Therefore, these light sources must be chromatic adaptation through CMCCAT2000. Consider the accuracy of spectrum estimation. The camera must also be color corrected, and the reflectance is measured by spectrophotometry. The spectrum is converted into CIE XYZ tristimulus values using Equation 2.5 $\sim$ 2.8. $S(\lambda)$ is the relative spectral distribution of the artificial light, $R(\lambda)$ is spectral reflectance of respective colorchecker, and $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ and $\bar{z}(\lambda)$ are the color matching functions.

$$X = \int_{780nm}^{380nm} S(\lambda)R(\lambda)\bar{x}(\lambda)\,d\lambda \tag{2.5}$$

$$Y = \int_{780nm}^{380nm} S(\lambda)R(\lambda)\bar{y}(\lambda)\,d\lambda \tag{2.6}$$

$$Z = \int_{780nm}^{380nm} S(\lambda)R(\lambda)\bar{z}(\lambda)\,d\lambda \tag{2.7}$$

Where

$$k = 100/\int_{780nm}^{380nm} S(\lambda)\bar{y}(\lambda)\,d\lambda \tag{2.8}$$

After chromatic adaptation was transformed, the RGB values matching to the new XYZ values were estimated by reverse processes of Equation 2.2 $\sim$ 2.4, and it calls $[F]$. RGB of the spectrophotometer is $[A]$, the relationship between the spectrophotometer and the camera is regressed by the RGB three-color variables using a third-order polynomial, and the color correction matrix $[C]$ is as follows :

$$[C] = [A]^T pinv[F] \tag{2.9}$$

Where camera rgb third-order polynomial

$$[F] = \begin{bmatrix} 1, R, G, B, RG, GB, BR, R^2, G^2, R^3, \\ G^3, B^3, RG^2, RB^2, GR^2, GB^2, BR^2, BG^2 \end{bmatrix}^T \tag{2.10}$$

R, G, and B are the respective value of the Color checker corresponding to the camera. Equation 2.11 obtaining the corrected RGB, $[K]$ is any RGB matrix formed from $[F]$.

$$\begin{bmatrix} Corrected & RGB \end{bmatrix} = [C][K] \tag{2.11}$$

Finally, the transposition matrix of the spectrophotometer and the camera is as follows, matrix $[\beta]$ is second-order expansion corrected RGB.

$$\begin{bmatrix} M \end{bmatrix} = [\alpha] pinv [\beta] \tag{2.12}$$

2.1.3 Simulate spectral of HSI

Combine the spectrophotometer in Section 2.1.1 and the camera in Section 2.1.2, for any pixel's RGB value in the camera. Then multiply the color correction matrix $[C]$ and use the square formula to calculate the corresponding XYZ value. The estimated spectrum of the visible light band $380 - 780nm$ can be obtained through the following formula :

$$\begin{bmatrix} Spectra \end{bmatrix}_{380nm-780nm} = [EV] pinv [M] \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.13}$$

2.2 HSI algorithm analysis

The RGB-to-HSI reconstruction algorithm is Camera of HSI in Section 2.1.2. It includes many matrix operations. With the help of the program analysis tool: Intel VTune profiler [11], we analyze the relationships among different kinds of operations and occupied execution time. Figure 2.2 shows the percentage of the execution time for different kinds of significant operations.

2.3 Translation using OpenCV

C++ programming language does not natively provide computer vision-related operations, so we use the Open Source Computer Vision Library (OpenCV) to help to handle image I/O. The matrix multiplication part uses the Mat type in the

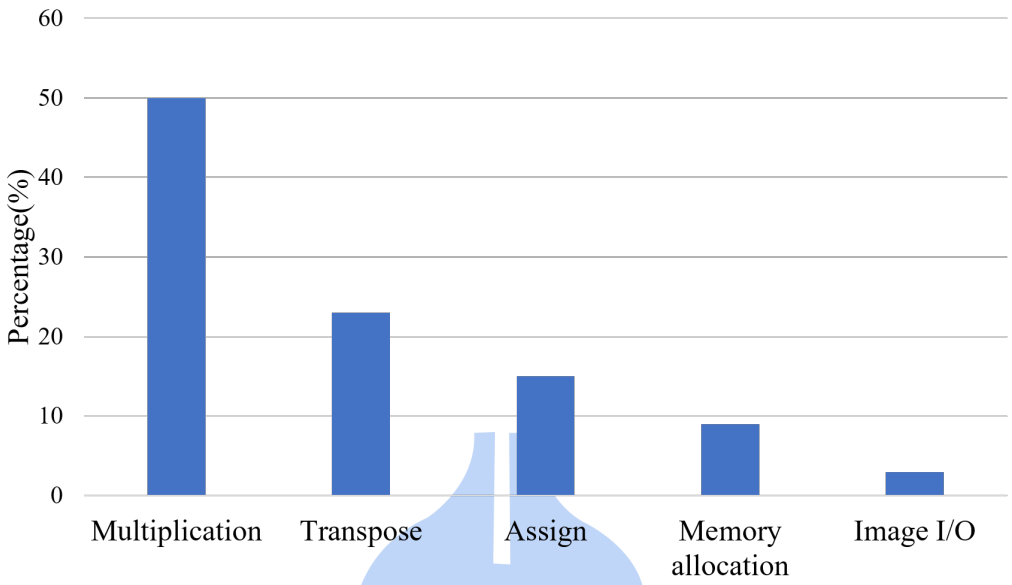

Figure 2.2: Percentage of the execution time for different kinds of significant operations.

OpenCV library (as shown in Listing 2.1). The three-dimensional matrix (RGB) must be converted to a two-dimensional type to be multiplied with other parameters. The calculation of a matrix multiplication takes 4 times matrix traversal (as shown in Listing 2.2).

```
1 std::vector<Point3f> vec;
2 ...
3 Mat pointMat = Mat(vec).reshape(1).t();
4 // make Nx3 1-channel matrix out of Nx1 3-channel.
5 // An O(1) operation
6 // finally, transpose the Nx3 matrix.
7 // This involves copying all the elements
```

Listing 2.1: Example of matrix multiplication in OpenCV.

```
#include <opencv2/opencv.hpp>
using namespace cv;
void 3D_matrix_multiplication(){
 int rows = 3D_Matrix.rows ;
 int cols = 3D_Matrix.cols ;
 2D_Matrix = 3D_Matrix.reshape(1, rows * cols ) ;
 2D_Matrix = transpose(2D_Matrix) ;
 2D_Matrix = 2D_temp * 2D_Matrix ;
 2D_Matrix = transpose(2D_Matrix) ;
 3D_Matrix = 2D_Matrix.reshape(3, rows) ;
 //reshape to RGB 3 channels
}
```

Listing 2.2: Code fragment of 3D matrix multiplication.

2.4 Hand-coded matrix operations

Using Mat data type in OpenCV suffers the problem of multiple matrix traversals. In order to avoid the problem, the matrix operations are hand-coded in the standard C++ programming language. For the operations of reshape matrix and transpose matrix, the traversal of the matrix is reduced by directly controlling the matrix indices. The total number of matrix traversals is reduced from 4 to 2. According to our observation, the locations of the data accessed by each traversal is independent. Therefore, OpenMP [12] is used here to enable thread-level parallelism to improve the execution time. Listing 2.3 shows the code fragment.

```
void reshape_2d_to_3dtranspose(){
 int index_2d = list_2d of index;
 int index_3d = list_3d of index;

 #pragma omp parallel for
 for (int k = 0; k < channel; k++)
  for (int j = 0; j < cols; j++)
   for (int i = 0; i < rows; i++)
    list_3d[index_3d] = list_2d[index_2d];
```

Listing 2.3: Code fragment of using OpenMP.

2.5 OpenBLAS

The hand-coded matrix operations are still time-consuming. Therefore, it is necessary to improve the matrix-related operations. OpenBLAS [13] is a fast matrix calculation library. The single-precision floating-point matrix multiplication function in OpenBLAS is called `cblassgemm()`, where the input is a one-dimensional matrix. In order to use the library, the two-dimensional matrix for accessing RGB is required to be changed to a one-dimensional matrix. Listing 2.4 shows the OpenBLAS code fragment.

```
1 A = (float*) malloc(m * k * sizeof(float));
2 B = (float*) malloc(k * n * sizeof(float));
3 C = (float*) malloc(m * n * sizeof(float));
4 initialize(m, k, n, A, B, C);
5 int alpha = 1;
6 int beta = 0;
7 //C=alpha*A*B+beta*C
8 cblas_sgemm(CblasColMajor,CblasNoTrans,CblasNoTrans,
9             m, n, k, alpha, A, m, B, k, beta, C, m);
```

Listing 2.4: Code fragment of using OpenBLAS.

2.6 Experiment

The 1920×1080 format image is used as the experimental input data, and the execution time is calculated by the algorithm for ten times to take the average. Table 2.1 shows the experimental configuration. We use OpenBLAS library to do the matrix multiplication, OpenCV library to help the operations of image I/O, and the matrix stored in a one-dimensional array to construct the program. Figure 2.3 shows the performance comparison, and Figure 2.4 shows the comparison of memory usage. The execution time can achieve 4.87 times speedup 2.3, and memory usage is reduced by 30% compared to the original MATLAB version.

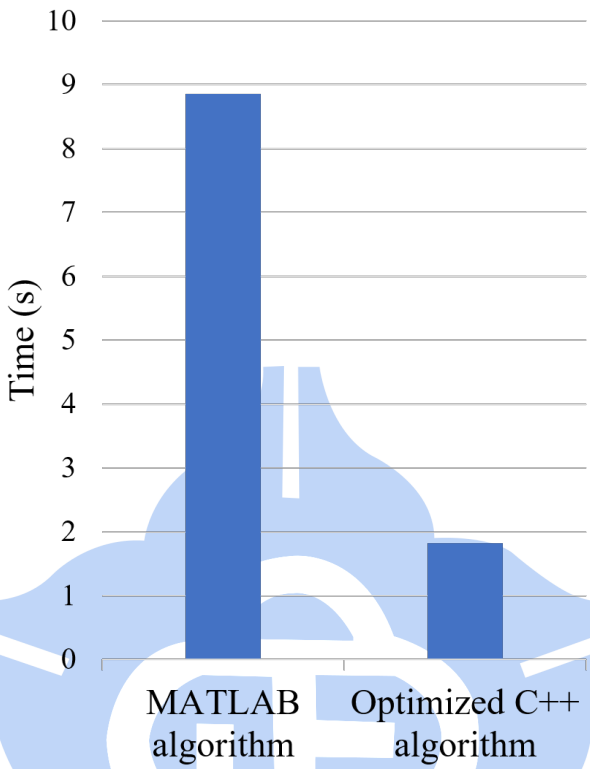

Figure 2.3: Execution time and time speedup is 4.87.

Table 2.1: Experimental configuration.

Item	**Description**
CPU	Intel Core-i5 8500 3.0GHz
RAM	16GB
Operating system	Ubuntu 18.04 LTS
Compiler	G++ 7.5.0
Time evaluation	Linux profiler - Perf
Memory evaluation	Massif with parameter stack=yes [14]
Test data	1920×1080 image

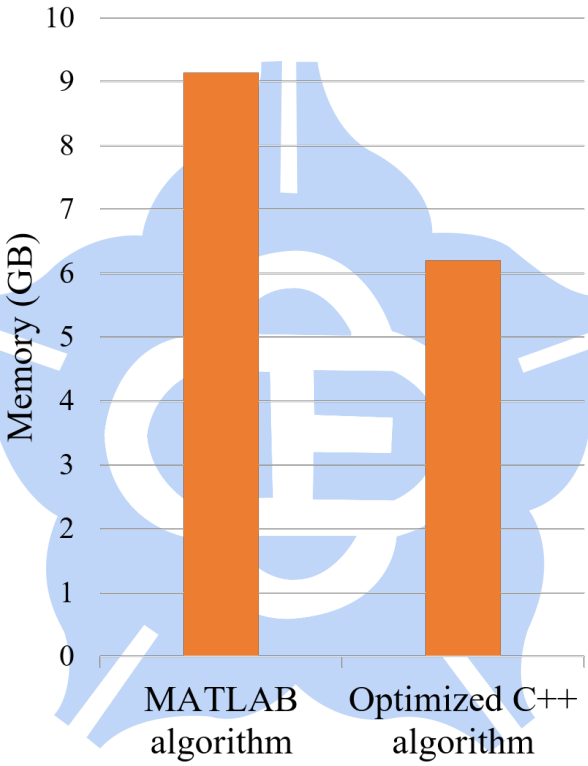

Figure 2.4: Reduce memory usage by 30%.

Chapter 3

Polynomial regression

RGB-to-HSI reconstruction algorithm calculates the RGB corresponding spectral reflectance for each pixel. If a database is used to query the corresponding spectral reflectance using each pixel's RGB, it needs to accommodate $256 \times 256 \times 256 \times 401$ single-precision float points; a total of about 25 GB of physical memory is required for the storage. Because general PC and embedded systems do not usually have enough physical memory, the idea is hard to not be realized In Section 2.1, mentioned that the relationship between the spectrophotometer and the camera in the HSI technology is regressed by the RGB three-color variables using third-order polynomial regression. This inspires us to obtain spectral data using regression analysis. The RGB-to-HSI reconstruction algorithm can provide a large number of data that contain RGB values and the corresponding spectral reflectances. These data are used to train regression weight using Scikit-learn library. The problem of overfitting during training is ignored.

3.1 Regression

Linear regression is to find the straight line (Equation 3.1) that best matches a group of data points. We use scikit-learn library to find the β_0 and β_1 that best match the data.

$$y_i = \beta_0 + \beta_1 x_i \tag{3.1}$$

When the data points do not show a linear distribution, the effect of using a straight line equation to match the data points will not be good. We need to add the straight-line equation to the power term, as shown in like Equation 3.2.

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 \tag{3.2}$$

Because there are three inputs in the HSI algorithm, the curve Equation must be Equation 3.3.

$$y_i = \beta_0 + \beta_1 r_i + \beta_2 g_i + \beta_3 b_i + \beta_4 r_i^2 + \beta_5 g_i^2 + \beta_6 b_i^2 + \beta_7 r_i g_i + \beta_8 r_i b_i + \beta_9 b_i g_i \tag{3.3}$$

The linear regression model training data is all RGB value combinations. Scikit-learn library [15] evaluates the accuracy of the model is R^2, as shown in Equation 3.5. A high degree polynomial equation might have a better predictive value than a low degree polynomial equation. However, a high degree polynomial equation requires more computation resources than a low degree equation. In order to give consideration to the accuracy and execution time, we evaluate the accuracy of predicting the visible light band using 1- to 5-degree polynomial equation. Table 3.1 shows the evaluation results of 1- to 5-degree polynomial equation, accuracy is the worst case in the visible light band, time is the polynomial equation execution. It was decided to use the 4-degree polynomial equation in the prediction algorithm. The 4-degree polynomial regression has 97% accuracy in the visible light band, and the worst light band is 87%.

$$\frac{u}{v} = \frac{((y_{true} - y_{pred})^2).sum()}{((y_{true} - y_{true.mean()})^2).sum()} \tag{3.4}$$

Where

$$R^2 = 1 - \frac{u}{v} \tag{3.5}$$

ICONES-HSI [16] dataset is used as the test data of our model, It contains Hyperspectral remote sensing images of 300×300 pixels generated from several HSI images from the NASA Jet Propulsion Laboratory's Airborne Visible InfraRed

Imaging Spectrometer (AVIRIS). The dataset is organized into nine categories: Agriculture, Forest, Desert, Urban, Snow, Mountain, Ocean, Wetland, and Cloud. Table 3.2 shows the accuracy of 4-degree polynomial regression in the test data set. The average accuracy is 89% in the test data set.

The regression algorithm uses multiplication and addition operations compared to the original HSI algorithm using matrix operations. In addition, the regression algorithm does not need to copy an additional memory space and saves 50% of the memory usage compare with the C++ version. Figure 3.1 shows HSI algorithm and regression algorithm memory usage.

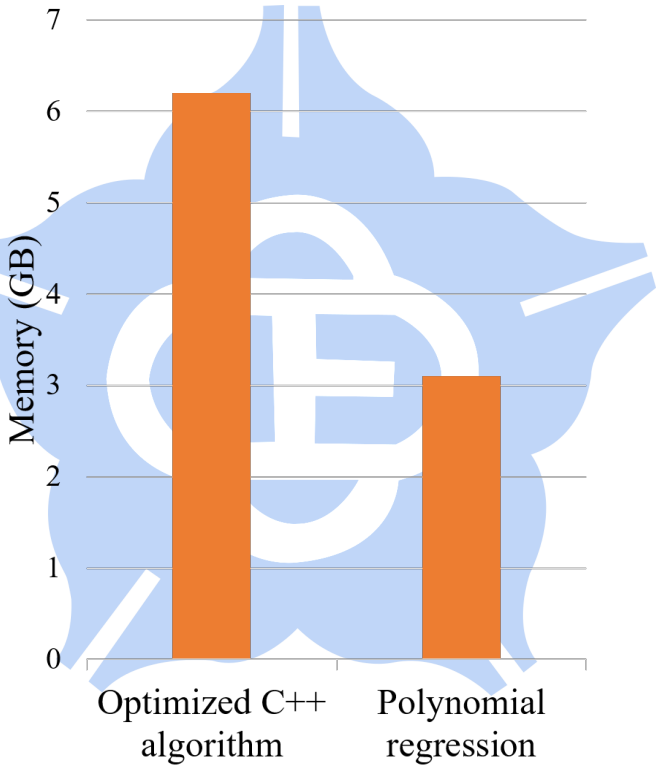

Figure 3.1: Optimized C++ algorithm and polynomial regression of memory.

3.2 Compiler optimization options

Table 3.3 shows the evaluation results of applying GCC compiler optimization options O0, O1, O2, and O3. The regression algorithm contains a large number

of floating-point calculations. We also used the "`-ffast-math`" option that allows the compiler to aggressively handle floating-point calculations. However, the computation is not IEEE-754 compliant. The evaluation results show that using the compiler option for polynomial regression can significantly reduce the execution time. Using compiler options "`O2`" and "`ffast-math`" can get the best execution performance.

3.3 Common subexpression elimination

Common subexpression elimination (CSE) is a compiler optimization technique. It searches for the same sub-expression in an expression and records the evaluated value through a variable to reduce repeated calculations. Listing 3.1 shows the polynomial regression has such characteristics, where a^2 is calculated five times. CSE reduces the calculation of the same expression and uses temporary variables to save a^2 and a^3 to reduce redundant calculations. Figure 3.2 shows the flow chart of CSE. Expressions represent multiplication expressions such as a$\times$a. ReduceSet is a collection of all expressions. Reduce2ndSet stores other expressions, and it is empty at beginning. If the subexpression is another expression's child expression in ReduceSet and appears count is larger than other expressions, it will be removed to Reduce2nd Set. If not, we build a new statement of expression and add its orderly before the original expression. The new statement replaces the original expression. The flow chart termination condition is that both ReduceSet andReduce2nd sets are empty. Algorithm 18 shows the gives pseudo code we applied. ReduceSet is a collection of all expressions at the beginning of the algorithm.

```
1 int x = w0*(a*a) + w1*(a*a*a) + w2*(a*a*b) +
2         w3*(a*a*a*a) + w4*(a*a*a*b) ;
3 // CSE may be trasforming the code to ... //
4 int tmp1 = a*a ;
5 int tmp2 = a * tmp1;
6 int x = w0*tmp1 + w1*tmp2 + w2*(tmp1*b) +
7         w3*(tmp2*a) + w4*(tmp2*b) ;
```

Listing 3.1: Polynomial regression.

Algorithm 1: Algorithm of CSE from ReduceSet.

Input:
$ReduceSet$: $common\_exp$;
Output:
$Reducend$: CSE_exp;

```
if ReduceSet is not empty then
    if other sub_expr be its child exp && sub_expr count > exp then
        ReduceSet.remove(sub_expr);
        Reduce2nd.add(sub_expr);
    end
    ReduceSet.InsertStmt();
    ReduceSet.ReplaceVar();
    ReduceSet.clear();
end
if Reduce2nd is not empty then
    if other sub_expr be its child exp && sub_expr count > exp then
        Reduce2nd.remove(sub_expr);
        ReduceSet.add(sub_expr);
    end
    Reduce2nd.InsertStmt();
    Reduce2nd.ReplaceVar();
    Reduce2nd.clear();
end
```

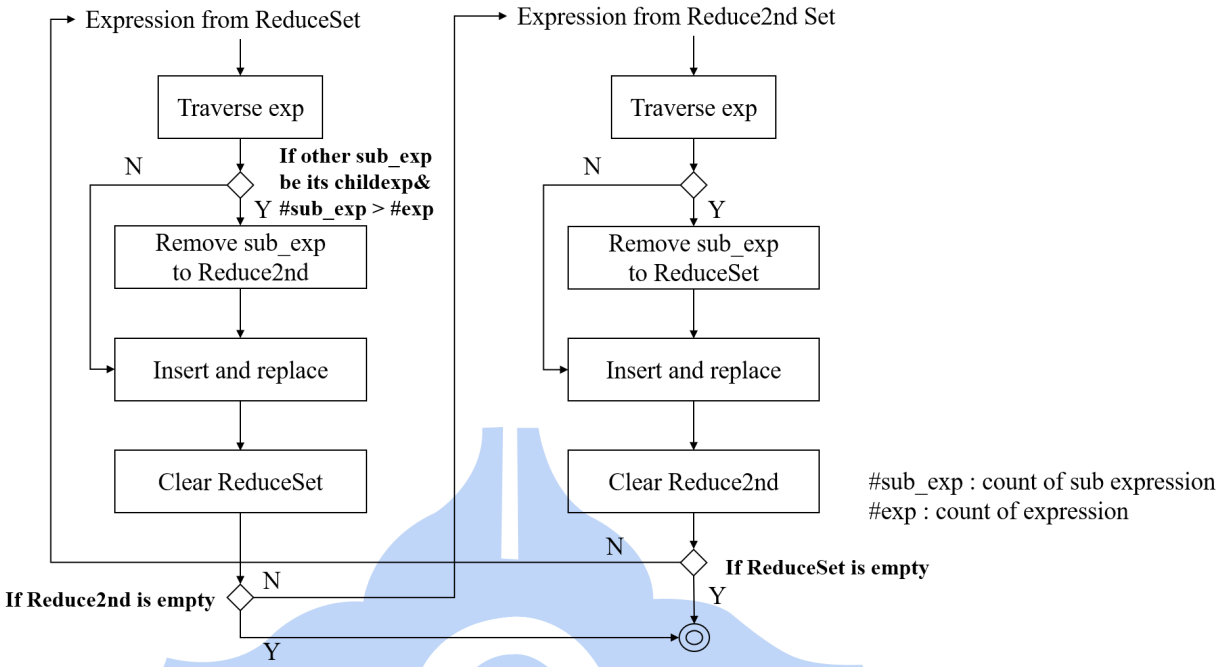

Figure 3.2: Flow chart of CSE algorithm.

3.4 Multithreading

Each pixel in the image is composed of three independent values (i.e., RGB value). Calculation of polynomial regression also inherits the property of computational independence. Therefore, multithreading is leveraged to parallelize the computation of polynomial regression. First, we parallel-for construct in OpenMP for multithreading implementation, as shown in Listing 3.2. However, using OpenMP will suffer additional API-invoked overheads. Therefore, C++11 std::thread is used to rewrite the code, as shown in Listing 3.3. Figure 3.3 shows C++11 std::thread gets more performance improvements than OpenMP.

```
1 #pragma omp parallel for
2 for( int i = 0; i < image_size; i++ )
3     spectral_reflectance = w0 + w1*1 + w2*r + w3*g + w4*b ... ;
```

Listing 3.2: OpenMP.

```
1 std::thread myThreads[numthread] ;
2 int part = image_size/numthread ;
3 for( int i = 0; i < numthread; i++ ){
4     myThreads[i] =
5     std::thread(spectrum, part*i, part*(i+1), ref(image_pixel));
6 }
7 for( unsigned int i = 0; i < numthread; i++ )
8    myThreads[i].join();
```

Listing 3.3: std::thread.

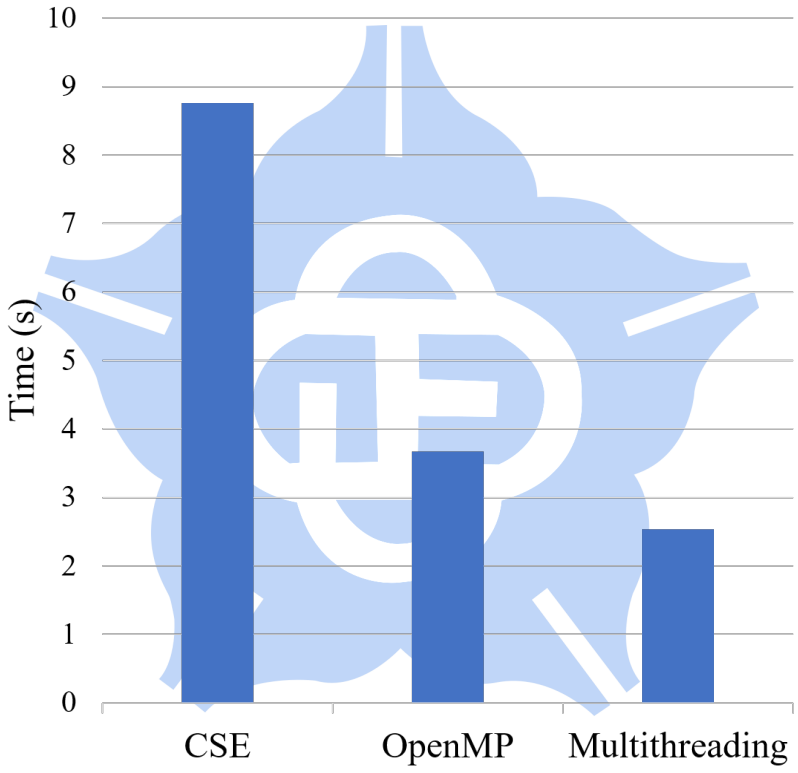

Figure 3.3: Multithreading std::thread better than OpenMP.

3.5 SIMD instruction

The calculation of RGB values in the polynomial regression is the multiplication of the weight and then adding after the multiplication. In addition to mul-

tithreading, it is very suitable to use Single Instruction Multiple Data (SIMD) instructions to exploit instruction-level parallelism, as shown in Listing 3.4.

```
1 int wave_size = 401; //Visible light range
2 float spectral_reflectance;
3 __m128 vRGB_0, vRGB_1, vRGB_2 ;
4 __m128 w_0, w_1, w_2 ;
5 float* pRGB ; //Point to RGB address
6 float* pw ;   //Point to weight address
7 vRGB_0 = _mm_load_ps(pRGB+=4);
8 vRGB_1 = _mm_load_ps(pRGB+=4);
9 for( int i = 0; i < wave_size; j++ ){
10  pw = w[i];
11  w0 = _mm_load_ps(pw+=4);
12  w1 = _mm_load_ps(pw+=4);
13  w0 = w0 * vRGB_0;
14  w1 = w1 * vRGB_1;
15  w0 = w0 + w1;
16  spectral_reflectance = sum( w0 );
17 }
```

Listing 3.4: Code fragment of SIMD.

For Intel x86 processors, SSE4 instruction can accommodate four single-precision floating-point operations, and AVX instruction can accommodate eight single-precision floating-point operations. The rewritten is shown in Figure 3.4. The calculation part uses SSE4 instructions to get 2.11 times speedup compared to the multithreading version. AVX instructions can achieve 2.24 times speedup compared to the multithreading version.

3.6 Memory allocation

After analyzing the polynomial regression algorithm, we found that memory-allocation-related operations occupy about 30% of the execution time. The memory allocation of the two-dimensional array mentioned in Section 2.5 takes a lot of time. Table 3.4 shows the memory-allocation-related function calls execution time. We rewrite the code fragment to reduce the number of the memory-allocation-related function calls, as shown in Listing 3.5. The execution time of the modified program achieves 1.82 times speedup compared to that of the program without memory-allocation optimization.

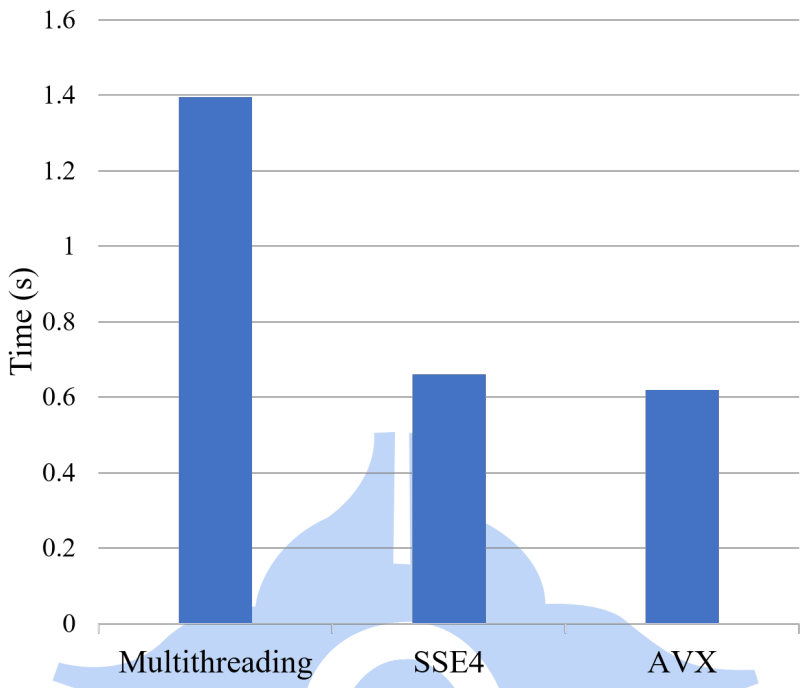

Figure 3.4: Performance comparison using SIMD instructions.

```
int image = 1920*1080;
int ch = 401;
int mem = image*ch;
float** spectral = (float**)malloc(image*sizeof(float*));
for( int i = 0; i < image; i++ )
 spectral[i] = (float*)malloc(ch*sizeof(float));

// Change the memory configuration to the following...
float* spectral = (float*)malloc(mem*sizeof(float));
```

Listing 3.5: Single memory allocation.

3.7 Embedded System

It is convenient if the HSI algorithm can be executed on embedded systems. We port the algorithm to the embedded system, Raspberry Pi 4B, which is a popular and low-price embedded system evaluation board. We rewrite the part of the

SIMD instructions to replace SSE4 intrinsic functions with-NEON-instruction-related intrinsic functions.

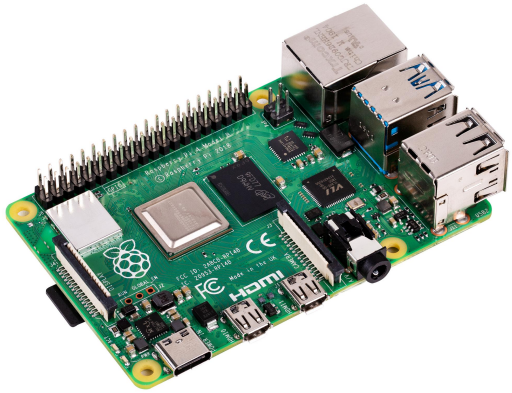

Figure 3.5: Snapshot of Raspberry Pi 4B.

3.8 Experiment

3.8.1 The experiment of polynomial regression

We use the Python Scikit-learn package to train the weight values in polynomial regression. In this experiment, we call it RGB hyperspectral model. The 1920×1080 format image is used as the experimental input data, and the execution time is calculated by the algorithm for ten times to take the average. Table 2.1 shows the experimental environment of the RGB hyperspectral model built on x86. HSI algorithm is the original MATLAB program. HSI algorithm (C++) represents the original MATLAB program implemented by C++ programming language. RGB hyperspectral model using SIMD instructions is equivalent to HSI algorithm (C++) in execution time. After adjusting Memory allocation, the program can be executed in about a second. Model optimization item is shown in Table 3.5. Each item inherits the optimization of the previous item. The speedup of the optimized items is shown in Table 3.6, where the speedup is compared with the polynomial regression (RGB hyperspectral model).

3.8.2 Evaluation of embedded systems

In order to evaluate the performance of embedded systems, we use Raspberry Pi 4B as our experimental environment. Table 3.7 shows the Raspberry Pi 4B experimental environment. The 1280×720 and 1920×1080 format image is used as the experimental input data, and the execution time is calculated by the algorithm ten times to take the average. Table 3.8 shows the HSI algorithm (C++) and polynomial regression execution time on Raspberry Pi 4. The original HSI algorithm (C++) needs more memory resources, so it cannot process on larger images. The proposed polynomial regression algorithm requires fewer memory resources, so it can perform hyperspectral construction on larger images, and the performance is much faster.

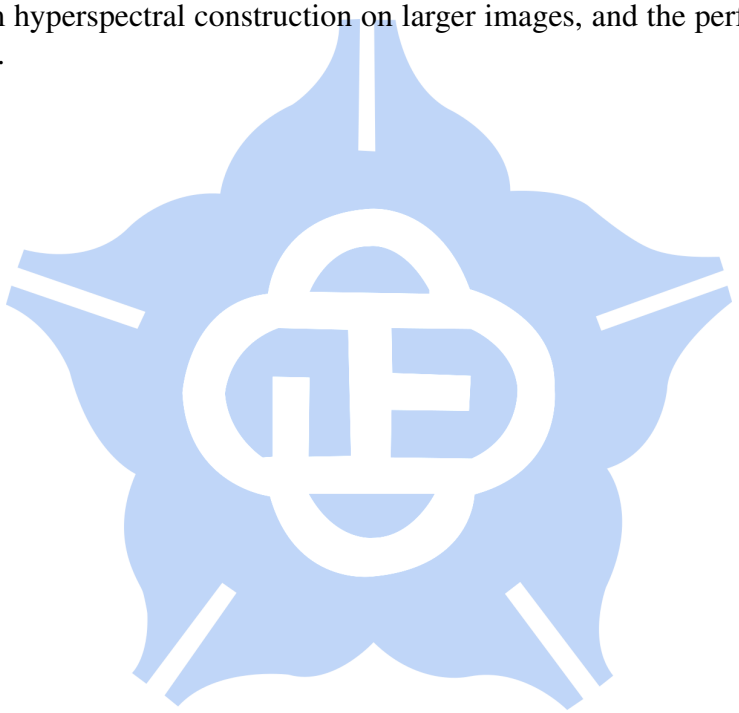

Table 3.1: Reconstruct 1920×1080 image and GCC option is O0.

Power terms	**Worst case accuracy(%)**	**Time(s)**
1th	24	5.634
2th	33	13.240
3th	64	26.852
4th	87	54.525
5th	97	92.706

Table 3.2: Accuracy of the 4-degree polynomial regression model in the CONES-HSI dataset.

Item	**Accuracy(%)**
Agriculture	91
Forest	79
Desert	98
Urban	97
Snow	96
Mountain	78
Ocean	81
Wetland	88
Cloud	97
Average	89

Table 3.3: Compiler option optimization.

Compiler optimize option	**Time(s)**
-O0	55.176
-ffast-math	55.128
-O1	15.818
-O1 -ffast-math	10.510
-O2	15.016
-O2 -ffast-math	10.242
-O3	15.063
-O3 -ffast-math	10.319

Table 3.4: Memory allocation 1920×1080 as an example.

Case	Memory allocation count	Time(s)
Two degree	image size + 1	0.883
One degree	1	0.000015

Table 3.5: Optimization item.

Item	Time(s)
HSI algorithm (MATLAB)	8.867
HSI algorithm (C++)	1.819
RGB hyperspectral model	14.929
Compiler optimize option	10.515
Common subexpression elimination	8.765
Multithreading	2.539
SIMD instruction	1.858
Memory allocation	1.019

Table 3.6: Optimization item of speed up.

Optimization item	Speedup
Polynomial regression	1.000
Compiler optimize option	1.419
Common subexpression elimination	1.120
Multithreading	3.452
SIMD instruction	1.366
Memory allocation	1.823

Table 3.7: Raspberry Pi 4 experimental configuration.

Item	**Description**
CPU	ARM Cortex-A72 (v8) 1.5GHz
RAM	4GB
Operating system	Linux raspberry pi 5.10
Compiler	G++ 8.3.0
Time evaluation	Linux time command - real time
Test data	1280×720, 1920×1080 image

Table 3.8: Execution time on Raspberry Pi 4.

Case	**1280×720**	**1920×1080**
HSI algorithm (C++)	316.036	run time error
Polynomial regression	4.995	8.925

Chapter 4

Conclusion

This thesis, in the view of programming, aims at enhancing the execution time of the hyperspectral reconstruction algorithm with reasonable memory resources. The whole research work consists of two parts.

The first part is to translate the RGB-to-HSI reconstruction algorithm from MATLAB code to the corresponding C++ program. OpenBLAS library is used to optimize the matrix operations, and OpenCV library is used to handle the image I/O-related operations. We also manually manage heap-memory allocations and deallocation for reducing memory footprints. The optimized program can achieve 4.87 times speedup and reduce 33% memory usage compared to the original unoptimized version.

The second part is to develop a new RGB-to-HSI reconstruction algorithm using the 4-degree model with polynomial regression. Several program optimization techniques are applied for improving execution performance. GCC compiler option `"-ffast-math"` is used to enhance the floating-point computation. CSE is used to eliminate redundant computation. Multithreading and SIMD techniques are used to exploit coarse-grained and fine-grained parallelism. The heap memory is manually managed to reduce the number of invoking memory-allocation functions. For the tested benchmarks, the average accuracy of the proposed algorithm reaches 97% compared with the original hyperspectral reconstruction algorithm; the execution time can achieve 8.7 times speedup and 66% memory-usage reduction compared to the original C++ program translated.

References

[1] S.-W. Huang, S.-H. Chen, W. Chen, I.-C. Wu, M. T. Wu, C.-T. Kuo, and H.-C. Wang, "Identification of early cancerous lesion of esophagus with endoscopic images by hyperspectral image technique (Conference Presentation)," in *Multimodal Biomedical Imaging XI* (F. S. Azar and X. Intes, eds.), vol. 9701, pp. 47 – 47, International Society for Optics and Photonics, SPIE, 2016.

[2] Website, "Aging indicators." Online available at `https://pop-proj.ndc.gov.tw/chart.aspx?c=10&uid=66&pid=60`, 2021.

[3] McKinsey, *Transforming healthcare with AI: The impact on the workforce and organizations*, 2020.

[4] N.-C. Kuo, *Research on key issues and countermeasures of smart medical care*, 2017.

[5] Website, "Matlab." Online available at `https://www.mathworks.com/`, 2021.

[6] H. F. Armin Schneider, *Hyperspectral Imaging in Biomedical Engineering in Gastrointestinal Surgery*, 2017.

[7] S.-C. Chang, H.-Y. Syu, Y.-L. Wang, C.-J. Lai, S.-Y. Huang, and H.-C. Wang, "Identifying the incidence level of periodontal disease through hyperspectral imaging," *Optical and Quantum Electronics*, vol. 50, p. 409, Oct 2018.

[8] H.-Y. Yao, K.-W. Tseng, H.-T. Nguyen, C.-T. Kuo, and H.-C. Wang, "Hyperspectral ophthalmoscope images for the diagnosis of diabetic retinopathy stage," *Journal of Clinical Medicine*, vol. 9, no. 6, 2020.

[9] Y.-T. Kao, "Implementation of an iterative learning control using c++," Master's thesis, National Chung Cheng University, 2014.

[10] J.-G. CHEN, "Source-level common subexpression elimination for polynomial expression in motion control of machine tools," Master's thesis, National Chung Cheng University, 2019.

[11] Website, "Intel vtune profiler." Online available at https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/vtune-profiler.html#gs.3o7yx3, 2021.

[12] Website, "Openmp." Online available at https://www.openmp.org/, 2021.

[13] Website, "Openblas." Online available at https://www.openblas.net/, 2021.

[14] Website, "Massif - valgrind." Online available at http://valgrind.org/docs/manual/ms-manual.html, 2019.

[15] Website, "Sklearn linear model linearregression." Online available at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html/, 2021.

[16] Website, "Icones-hsi dataset." Online available at https://xlim-sic.labo.univ-poitiers.fr/datasets/ICONES-HSI/?lang=en, 2021.