**ATILIM UNIVERSITY**

**CMPE464**

**Digital Image Processing**

**HOMEWORK #05**

Müslüm Barış KORKMAZER

19244710045

Ankara,

Spring, 2024

# Implementation Overview

**Initialization**

The program defines constants including the path to the images and the total number of images. It also initializes an empty matrix (cv::Mat) to accumulate the sum of the images.

**Image Processing Loop**

**Image Loading:** Each image in sequence is read from the specified folder. They are read in using OpenCV’s cv::imread() function in color mode.

**Initialization of Sum Matrix:** During the first iteration, the imagesSum matrix is initialized as a zero matrix with the same size and type (32-bit float, 3 channels) as the first image to enable proper accumulation.

**Type Conversion and Accumulation:** Each image is converted into a 32-bit float type and accumulated into the imagesSum matrix to prevent data overflow and maintain precision.

**Average Calculation**

After summing all images, the sum matrix is divided by the total number of images to compute the average.

**Conversion to Different Precisions**

The averaged image is converted into multiple precision formats:

* **8-bit Unsigned Integer:** The image is scaled down to 8 bits for compatibility with common image formats.
* **16-bit Unsigned Integer:** A higher bit-depth version is created for improved dynamic range.
* **32-bit Float:** Maintained in its original summation format for maximum precision.

# Results and Analysis

# **Image Quality**

# Saving images in different bit-depth formats affects the quality and the file size. Typically, the 8-bit images are smaller and suitable for general use, whereas 16-bit and higher are better for tasks requiring high dynamic range and precision.

# **Precision and Range**

# The 32-bit format maintains the highest precision and is crucial for scientific computations where detail, especially in low intensity areas, is critical. Conversely, 8-bit images might exhibit banding in smooth gradients due to limited tonal range.

# **Performance**

# The program’s reliance on disk I/O for reading and writing images and the computational overhead of converting image types means that performance is a function of the image resolution, count, and system I/O speed.

# iç mekan, duvar, vazo, mobilya içeren bir resim Açıklama otomatik olarak oluşturulduiç mekan, duvar, vazo, mobilya içeren bir resim Açıklama otomatik olarak oluşturuldu

Figure 1: average\_image\_8bit.jpg Figure 2: average\_image\_16bit.jpg

# 

Figure 3: average\_image\_32bit.jpg

# Code

## main.cpp

#include <iostream>

#include <string>

#include <vector>

#include <opencv2/opencv.hpp>

const std::string IMAGES\_PATH = "/home/babico/Projects/CMPE464-Github/HW5/images/";

const int         IMAGES\_NUM  = 80;

int main()

{

    cv::Mat imagesSum;

*// Get the sum of all images*

    for (size\_t i = 1; i <= IMAGES\_NUM; i++)

    {

*// Load the image as colored*

        cv::Mat img = cv::imread(IMAGES\_PATH + "PIC" + std::to\_string(i) + ".jpg", cv::IMREAD\_COLOR);

*// Check sum image is initialized. If not, initialize as same size with the first image*

        if (imagesSum.empty())

        {

*// Initialize imagesSum with the same size and type as img, but filled with zeros and 32-bit float*

            imagesSum = cv::Mat::zeros(img.size(), CV\_32FC3);

        }

*// Convert the image to 32-bit float*

        cv::Mat imgFloat;

        img.convertTo(imgFloat, CV\_32FC3);

*// Sum all images pixel-wise*

        imagesSum += imgFloat;

    }

*// Calculate the average of all images*

    imagesSum = imagesSum / (double)IMAGES\_NUM;

*// Convert the averaged image to 8-bit for more sharpness*

    cv::Mat \_8bitImage;

    imagesSum.convertTo(\_8bitImage, CV\_8UC3);

*// Convert the averaged image to 16-bit for more precision*

    cv::Mat \_16bitImage;

    imagesSum.convertTo(\_16bitImage, CV\_16UC3);

*// Convert the averaged image to 64-bit for more precision*

    cv::Mat \_64bitImage;

    imagesSum.convertTo(\_64bitImage, CV\_64FC3);

*// Save the results*

    cv::imwrite(IMAGES\_PATH + "average\_image\_8bit.jpg", \_8bitImage);

    cv::imwrite(IMAGES\_PATH + "average\_image\_16bit.jpg", \_16bitImage);

    cv::imwrite(IMAGES\_PATH + "average\_image\_32bit.jpg", imagesSum);

*// Log the results*

    std::cout << "Average image is saved as average\_image\_8bit.jpg, average\_image\_16bit.jpg and average\_image\_32bit.jpg" << std::endl;

    return 0;

}

# Git Repository

[babico/CMPE464: Atılım University CMPE464 Digital Image Processing (github.com)](https://github.com/babico/CMPE464)