CSCI 3280 Introduction to Multimedia Systems

Spring 2023, Assignment 1 - Photomosaic

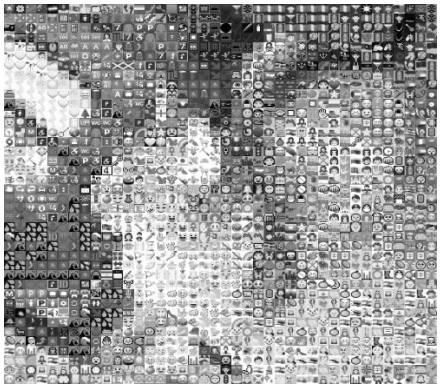
Due Date: Feb. 13, 2023 (11:59pm) Submission via Blackboard

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

PLAGIARISM penalty: Whole Course Failed

Introduction

Photomosaic is a form of digital art in which a picture is divided into a grid of equal-sized cells, with its cells substituted using photos of similar colors (similar gray tones in the basic part of this assignment). It is a popular way of presenting a large number of photos on posters or videos, so that seeing it from far away tells viewer the overall theme of the photos, while looking closer reveal the content of each single photo in the album. In this assignment, you are required to develop a photomosaic generator, which creates grayscale photomosaic of an image with a given set of small images.


into


General Requirements

1. Program should be coded in ANSI C/C++ and uses libraries listed in the skeleton code only. You are encouraged to use Vector class for handling image and color data.
2. The compiled program should run in **Windows** command prompt as a console program and accepts source bitmap (.bmp format) with the following syntax and save the output image to a bitmap file.

   C:\> photomosaic <input.bmp> <photo_dir>
   <output_w,output_h,cell_w,cell_h> <output.bmp>

   photomosaic is your program executable.

   <input.bmp> is the full path to the source bitmap file.

   <photo_dir> is the full path to the photo tile bitmaps directory.

   <output_w,output_h,cell_w,cell_h> is the output width, height and cell width, height. You can assume the output width to be an integer multiple of the cell width, so as the heights.

   <output.bmp> is the full path name to the output bitmap file.

3. Simple libraries for reading/writing .bmp format and listing files in a directory will be given as bmp.cpp and list_files.cpp (as well as the corresponding header files).
4. You are required to submit source codes only. We will use Visual Studio 2019 C++ compiler and have your program compiled via visual studio command prompt and the following command line.

   C:\> cl photomosaic.cpp bmp.cpp list_files.cpp -std:c++17

   photomosaic.cpp is the provided skeleton code for the basic part of the assignment. For the enhancement part, copy the source file and name it **photomosaic_enhancement.cpp**; it will be compiled and executed in the exact same way.

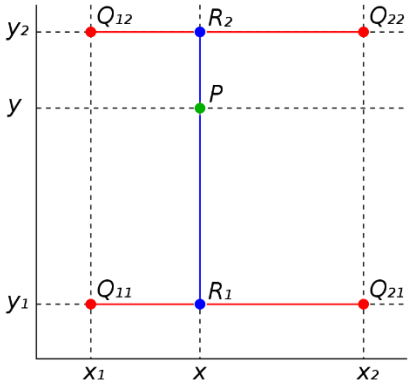
   Please make sure that your source code gets compiled well with it, "failed to compile" receives 10-point deduction for each failed source.
5. Test bitmaps are included with the skeleton code for testing.

Part 1 – Grayscale Photomosaic (`photomosaic.cpp`, 80 points)

For the basic part of the assignment, your program should process the RGB image into a grayscale photomosaic in a way similar to the following steps:

1. Read and arrange source image and photo tiles: - Read in the .bmp as an RGB bitmap, each pixel has R, G and B channels and each channel uses 'unsigned char' (i.e., one single byte) as their storage unit.

2. Bilinear Interpolate for resizing images: - Use bilinear interpolation to resample the input image to desired output size and resample the photo tiles to the given cell size.



When resampling an image, consider pixels as square regions (for this assignment) and define a mapping function to map each unknown pixel in the target image to a pixel location P in the source image. You can locate four pixels (Q) surrounding the point P and linearly interpolate the values of the pixels to get the pixel value of P. For pixel P that has only 2 neighbors (e.g., on borders), you can interpolate between the two border pixels.

3. Query for photo tiles with nearest brightness values: - For each `cell_w` by `cell_h` region in the resampled source image (assumed to be resized to the desired output shape in the following of the instructions), match it with the photo tile that has the nearest brightness value, such that the perceptual difference compared to the source image is minimized. To acquire a single brightness value for a piece of RGB image, you may use the following formula, then average among all pixels in the image.

   Y' = 0.299 * R + 0.587 * G + 0.114 * B

   For computational efficiency, you are encouraged to cache the brightness values for each photo tile.

4. Compose the output image with photo tiles: - With the photo tiles selected for each cell region in the resampled source image, copy the pixel array from the photo tile to the corresponding locations of the output image.

5. Save grayscale output image as bitmap file: - The convert the output image to grayscale because our similarity metric only makes sense for grayscale photomosaic. Save the result image to a bitmap file according to the path given as command line argument. (No need to convert to 8-bit .bmp file, just use the given library)

Part 2 – Enhancement Features (`photomosaic_enhancement.cpp`, 20 points)

You are encouraged to implement some of the following enhancements or some other features that you find interesting on top of the standard requirements. Please put the program with standard requirements plus enhanced features into a separate standalone source file and name it `photomosaic_enhancement.cpp`. You are free to use other libraries and output colored image depending on your enhancement features.

Some suggested features:

- Implement other interpolation methods (e.g., bicubic interpolation) or do extrapolation on the borders.
- Use other metrics for measuring distance between source image regions and photo tiles that are more perceptually similar than using mean brightness as metric. Apart from using Euclidean distance between RGB colors, you may also explore other color spaces or structural-based similarity metric for better visual quality. (The output may not always be visually pleasing due to the limited variety of photo tiles. We will accept your method as long as it is correctly implemented and well explained in your report)
- Use method to accelerate nearest neighbor search, assuming the collection of photo tiles are fixed and queried multiple times with different source image. (You may use functions in the standard library for ease of implementation, but the core procedure of the method should not be a function call to a predefined library function) You can also generate and read/write metadata for the photo tiles to handle excessive amount of photo tiles that cannot fit into the RAM.
- Develop a method to avoid adjacent cells to use the same photo tile when there are insufficient photo tiles (trade off the color accuracy for visual richness).
- Other interesting and creative extension of the basic requirements

Please write a report that consists of the following items:

- For each additional feature:
  - The explanation of the features
  - The source code segment related to this feature
  - Sample runs (including the execution code and output) (if applicable)
  - The techniques used (if applicable)
  - References (E.g., the feature you implemented is based on an algorithm found on Internet or a reference book)

We will do the grading and test the correctness of the source code based on the features you mentioned in the report. You are required to provide sufficient information in details to facilitate the grading. No marks will be given if you implemented a feature in the source code without mentioning it in the report. If the

features claimed in the report can only be barely used (e.g., with a lot of bugs) or even do not exist in the source code, marks will be deducted.

Submission (Deadline: Feb. 13, 2023 11:59pm)

We expect the following files zipped into a file named by your student ID (e.g., 1155xxxxxx.zip) and have it uploaded to the course's Blackboard system.

1. `report.pdf` (Tell us anything that we should pay attention to, especially about the enhancement part)
2. `photomosaic.cpp` (Standard requirement source code)
3. `photomosaic_enhancement.cpp` (Basic + enhancement part source code)

Other files that are provided (e.g. `bmp.cpp`) are not required to be submitted and will be overwritten during compilation.

*******IMPORTANT*******

All code should be implemented by yourself. Any kind of plagiarism (including copying online source code or/and code of classmates) in all assignment parts (both the general part and the enhancement part) will not be tolerated and will be subjected to disciplinary penalties.