Programming Assignment 1

CS 474

Alexander Novotny Matthew Lyman Page

October 2, 2020

Contents

1 Image Sampling

2 Image Quantization

3 Histogram Equalization

3.1 Theory

It is desirable to have high contrast in an image, as it allows you (and a computer vision algorithm) to pick out details more easily. In general, images whose histograms have a uniform distribution tend to have high contrast - especially when compared with images with a central mode (represented by a central "hump" in the histogram). To convert a (continuously distributed) random variable X to a uniform distributed random variable Y, we simply apply the transformation

$$Y = F_X(X),$$

where F_X is the CDF (cumulative distribution function) of X. As a transformation of the variable X, we know then that the PDF (probability density function) of Y is

$$f_Y(y) = f_X(F_X^{-1}(y))\left|\frac{\mathrm{d}}{\mathrm{d}y}F_X^{-1}(y)\right|,$$

and by the inverse function theorem of calculus,

$$\begin{aligned}
&= f_X(F_X^{-1}(y))\left|\frac{1}{F_X'(F_X^{-1}(y))}\right| \\
&= f_X(F_X^{-1}(y))\left|\frac{1}{f_X(F_X^{-1}(y))}\right| \\
&= 1,
\end{aligned}$$

so $Y \sim U(0,1)$. Of course, this only applies to continuous random variables, but we hope that a similar behaviour can be observed in discrete random variables. Unfortunately, since all pixels fall into a certain number of 'bins' in the image's histogram (based on the quantization of the image), the transform can't decrease the number of pixels in a bin. Instead, it can only spread bins out in the histogram and consolidate multiple bins into one, increasing the number of pixels in a bin. Therefore, if there are noticeable modes in the original image's histogram, there will still be noticeable modes in the equalized histogram. As well, image quality will drop due to the spreading out and consolidating of bins effectively quantizing the image.

3.2 Implementation

An array of integers is used for the image's histogram, which is calculated by looping over the image's pixels and incrementing the bin whose index is given by the pixel's value. Then, the CDF is calculated by iteratively summing over the calculated histogram, using the recurrence relation for discrete CDFs:

$$\begin{aligned}
F_X(x) &= \sum_{i=-\infty}^{x} P(X = x) \\
&= P(X = x) + \sum_{i=-\infty}^{x-1} P(X = x) \\
&= P(X = x) + F_X(x-1).
\end{aligned} \tag{1}$$

The CDF is never converted to its normalized version. Instead, when applying the transformation, each resulting transformed pixel is multiplied by the normalization constant. This is to prevent accumulation of round-off errors until the final integer pixel value is calculated.

Since the calculation of the original histogram and the transformation is embarrassingly parallel, OpenMP is used to parallelize.

The source code for this implementation can be found in listing 5.

3.3 Results and Discussion

Figure 1 shows the result of applying the algorithm to the image `boat.pgm`. There is a noticeable difference in contrast - especially in the water, which is much clearer, and the shadows on the sail. However, there is some noise introduced in the sky, and loss of detail on the coast.

Figure 1: A comparison of `boat.pgm` with its equalization (right).

Figure 2 compares the original histogram of `boat.pgm` with the histogram of the new equalized image. As discussed in section 3.1, the new histogram is not that of a uniform distribution, but there are some notable improvements over the original histogram. Firstly, the bins concentrated around the various modes have become sparser, so while the modes still exist with the same number of pixels in their bins(as discussed earlier), there are fewer pixels in the region of the bin. As well, a couple of the modes have spread out, making them easier to differentiate between. Finally, the bins in lower regions of the histogram have concentrated so they aren't as low compared to the modes.

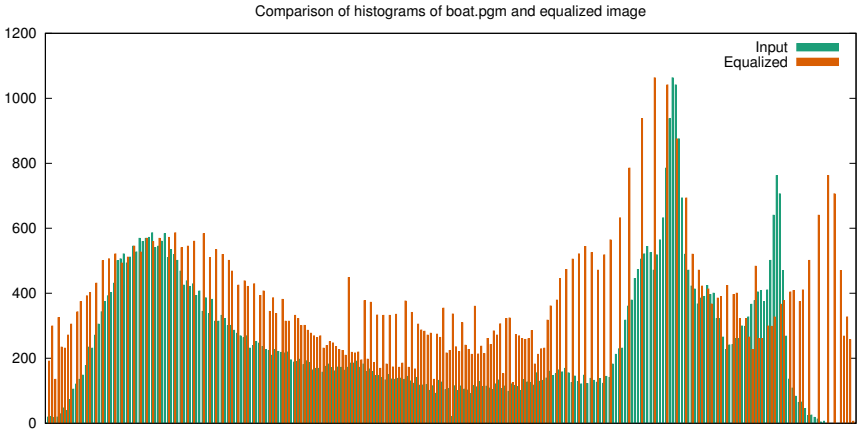

Figure 2: A comparison of histograms of `boat.pgm` and its equalized version

Figure 3 shows the result of applying the algorithm to the image `f_16.pgm`. There's a drastic

increase in contrast in the clouds, but the results around the text on the plane are a mixed bag - the "U.S. AIR FORCE" text in the middle of the plane has good increase in contrast, while the "F-16" text on the tail has a decrease in contrast. As well, there is loss of detail on the mountains and the aberration along the left and lower rims of the image.

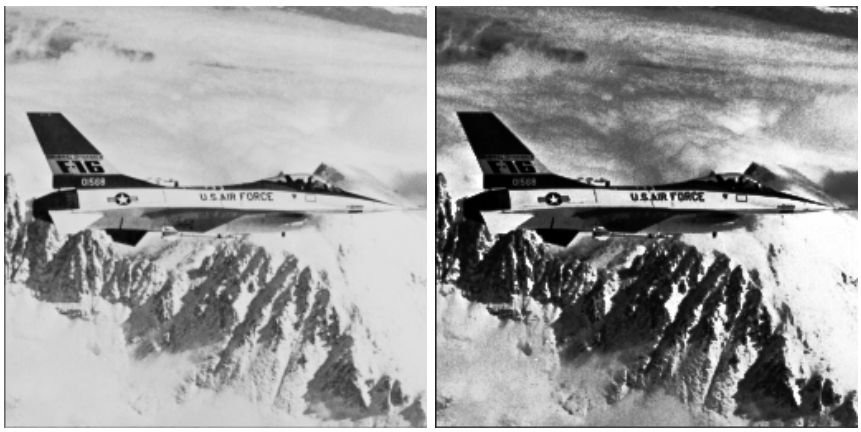

Figure 3: A comparison of `f_16.pgm` with its equalization (right).

Figure 4 compares the original histogram of `f_16.pgm` with the histogram of the new equalized image. The sparseness of bins and consolidation of bins is more apparent than in the previous example, especially around the mode of the image. This probably accounts for the loss of detail in the image, since most of the notable loss of detail happened in brighter regions of the image. These regions all got placed into the same bin, causing them to lose contrast and detail.

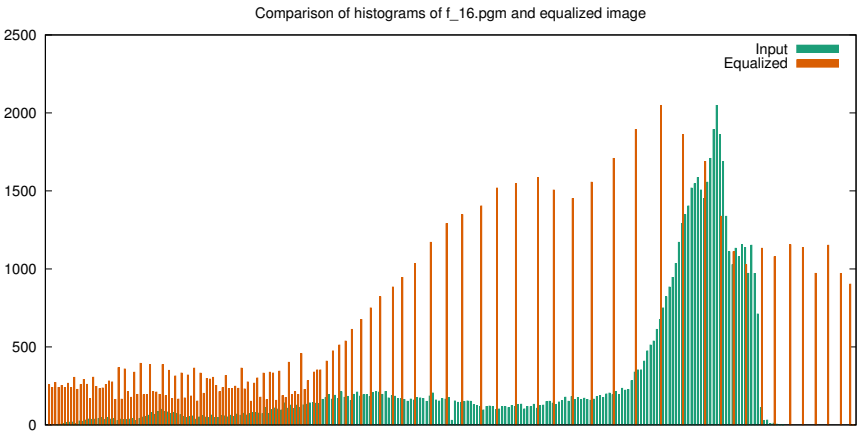

Figure 4: A comparison of histograms of `f_16.pgm` and its equalized version

4 Histogram Specification

4.1 Theory

4.2 Implementation

4.3 Results and Discussion

Figure 5: A comparison of `boat.pgm` with its specification to `sf.pgm` (right).

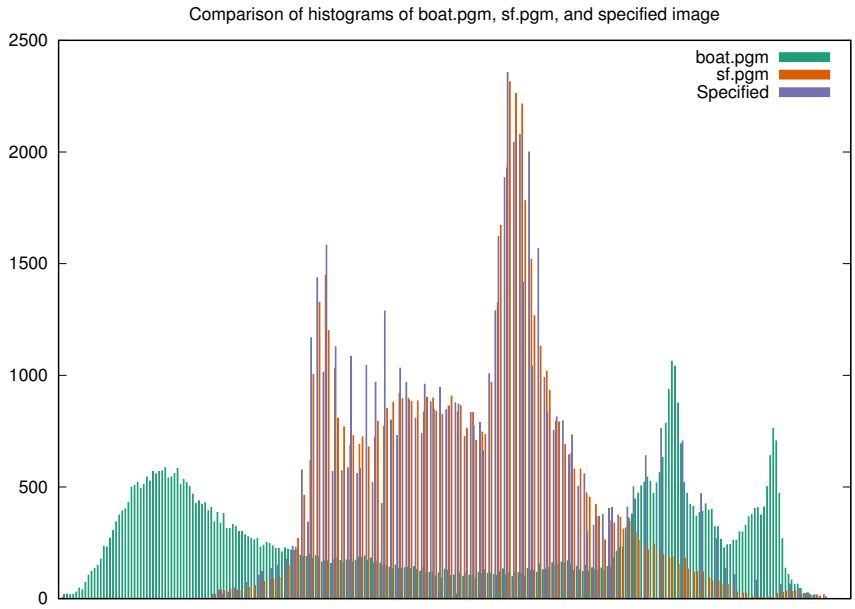

Figure 6: A comparison of histograms of `boat.pgm`, `sf.pgm`, and the specified output above.

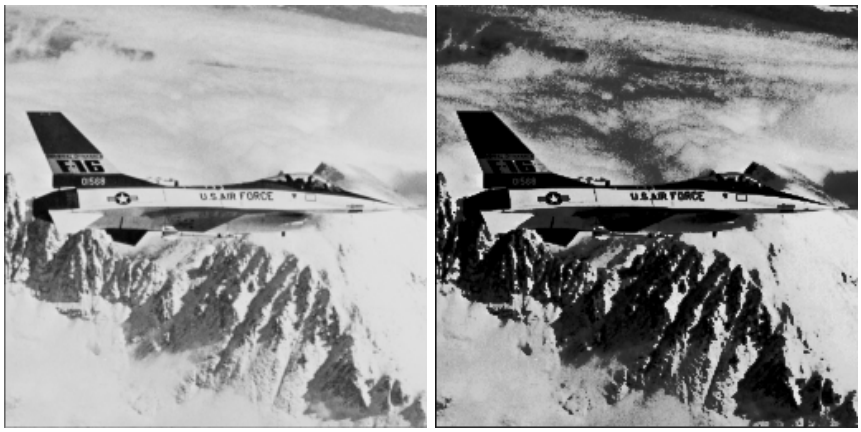

Figure 7: A comparison of `f_16.pgm` with its specification to `peppers.pgm` (right).

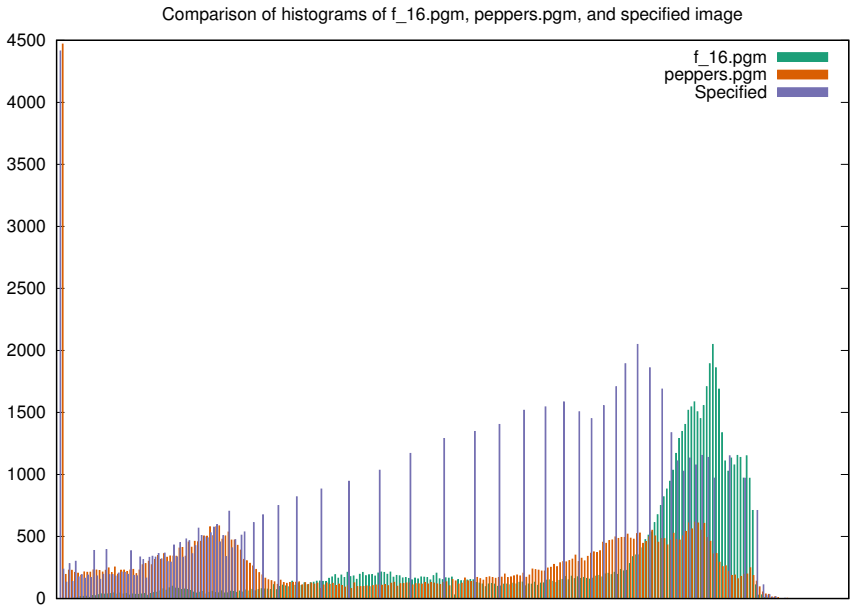

Figure 8: A comparison of histograms of `f_16.pgm`, `peppers.pgm`, and the specified output above.

Code Listings

Listing 1: Header file for the common `Image` class.

```
1  // Common/image.h
2  #pragma once
3
4  #include <iostream>
5
6  class Image {
7  public:
8    // The type that is used for the value of each pixel
9    // As of right now, read and operator<< only work if it is one byte large
10   typedef unsigned char pixelT;
11   // Struct for reading just the header of an image
12   struct Header {
13     enum Type {
14       COLOR,
15       GRAY,
16     } type;
17
18     unsigned M, N, Q;
19
20     // Read header from file
21     // Throws std::runtime_error for any errors encountered,
22     // such as not having a valid PGM/PPM header
23     static Header read(std::istream &in);
24   };
25
26   Image();
27   Image(unsigned, unsigned, unsigned);
28   Image(const Image &);  // Copy constructor
29   Image(Image &&);       // Move constructor
30   ~Image();
31
32   // Read from stream (such as file)
33   // Throws std::runtime_error for any errors encountered,
34   // such as not being a valid PGM image
35   static Image read(std::istream &in);
36
37   // Output to stream (such as file)
38   friend std::ostream &operator<<(std::ostream &out, const Image &im);
39
```

```
  // Pixel access - works like 2D array i.e. image[i][j]
  pixelT *operator[](unsigned i);
  const pixelT *operator[](unsigned i) const;
  Image &operator=(const Image &rhs);  // Assignment
  Image &operator=(Image &&rhs);       // Move

  // Read-only properties
  pixelT *const &pixels  = pixelValue;
  const unsigned &rows   = M;
  const unsigned &cols   = N;
  const unsigned &maxVal = Q;

private:
  Image(unsigned, unsigned, unsigned, pixelT *);
  unsigned M, N, Q;
  pixelT *pixelValue;
};

std::ostream &operator<<(std::ostream &out, const Image::Header &head);
```

Listing 2: Implementation file for the common `Image` class.

```
// Common/image.cpp
#include "image.h"

#include <cassert>
#include <cstdlib>
#include <exception>

Image::Image() : Image(0, 0, 0, nullptr) {}

Image::Image(unsigned M, unsigned N, unsigned Q) : Image(M, N, Q, new
↪  Image::pixelT[M * N]) {}

Image::Image(const Image& oldImage) : Image(oldImage.M, oldImage.N,
↪  oldImage.Q) {
  for (unsigned i = 0; i < M * N; i++) { pixelValue[i] =
  ↪  oldImage.pixelValue[i]; }
}

// Move constructor - take old image's pixel values and make old image
↪  invalid
Image::Image(Image&& oldImage) : Image(oldImage.M, oldImage.N, oldImage.Q,
↪  oldImage.pixelValue) {
  oldImage.M = oldImage.N = oldImage.Q = 0;
  oldImage.pixelValue                  = nullptr;
}

Image::Image(unsigned M, unsigned N, unsigned Q, pixelT* pixels)
```

```
23      : M(M), N(N), Q(Q), pixelValue(pixels) {}
24
25  Image::~Image() {
26    if (pixelValue != nullptr) { delete[] pixelValue; }
27  }
28
29  // Slightly modified version of readImage() function provided by Dr. Bebis
30  Image Image::read(std::istream& in) {
31    int N, M, Q;
32    unsigned char* charImage;
33    char header[100], *ptr;
34
35    static_assert(sizeof(Image::pixelT) == 1,
36                  "Image reading only supported for single-byte pixel
                      ↪  types.");
37
38    // read header
39    in.getline(header, 100, '\n');
40    if ((header[0] != 'P') || (header[1] != '5')) { throw
      ↪  std::runtime_error("Image is not PGM!"); }
41
42    in.getline(header, 100, '\n');
43    while (header[0] == '#') in.getline(header, 100, '\n');
44
45    N = strtol(header, &ptr, 0);
46    M = atoi(ptr);
47
48    in.getline(header, 100, '\n');
49    Q = strtol(header, &ptr, 0);
50
51    if (Q > 255) throw std::runtime_error("Image cannot be read correctly (Q >
      ↪  255)!");
52
53    charImage = new unsigned char[M * N];
54
55    in.read(reinterpret_cast<char*>(charImage), (M * N) * sizeof(unsigned
      ↪  char));
56
57    if (in.fail()) throw std::runtime_error("Image has wrong size!");
58
59    return Image(M, N, Q, charImage);
60  }
61
62  // Slightly modified version of writeImage() function provided by Dr. Bebis
63  std::ostream& operator<<(std::ostream& out, const Image& im) {
64    static_assert(sizeof(Image::pixelT) == 1,
65                  "Image writing only supported for single-byte pixel
                      ↪  types.");
66
67    out << "P5" << std::endl;
```

```
68    out << im.N << " " << im.M << std::endl;
69    out << im.Q << std::endl;
70
71    out.write(reinterpret_cast<char*>(im.pixelValue), (im.M * im.N) *
      ↪  sizeof(unsigned char));
72
73    if (out.fail()) throw std::runtime_error("Something failed with writing
      ↪  image.");
74  }
75
76  Image& Image::operator=(const Image& rhs) {
77    if (pixelValue != nullptr) delete[] pixelValue;
78
79    M = rhs.M;
80    N = rhs.N;
81    Q = rhs.Q;
82
83    pixelValue = new pixelT[M * N];
84
85    for (unsigned i = 0; i < M * N; i++) pixelValue[i] = rhs.pixelValue[i];
86
87    return *this;
88  }
89
90  Image& Image::operator=(Image&& rhs) {
91    if (pixelValue != nullptr) delete[] pixelValue;
92
93    M          = rhs.M;
94    N          = rhs.N;
95    Q          = rhs.Q;
96    pixelValue = rhs.pixelValue;
97
98    rhs.M = rhs.N = rhs.Q = 0;
99    rhs.pixelValue        = nullptr;
100
101   return *this;
102 }
103
104 Image::pixelT* Image::operator[](unsigned i) {
105   return pixelValue + i * N;
106 }
107
108 const Image::pixelT* Image::operator[](unsigned i) const {
109   return pixelValue + i * N;
110 }
111
112 // Slightly modified version of readImageHeader() function provided by Dr.
      ↪  Bebis
113 Image::Header Image::Header::read(std::istream& in) {
114   unsigned char* charImage;
```

```
115   char header[100], *ptr;
116   Header re;
117
118   // read header
119   in.getline(header, 100, '\n');
120   if ((header[0] == 'P') && (header[1] == '5')) {
121     re.type = GRAY;
122   } else if ((header[0] == 'P') && (header[1] == '6')) {
123     re.type = COLOR;
124   } else
125     throw std::runtime_error("Image is not PGM or PPM!");
126
127   in.getline(header, 100, '\n');
128   while (header[0] == '#') in.getline(header, 100, '\n');
129
130   re.N = strtol(header, &ptr, 0);
131   re.M = atoi(ptr);
132
133   in.getline(header, 100, '\n');
134
135   re.Q = strtol(header, &ptr, 0);
136
137   return re;
138 }
139
140 std::ostream& operator<<(std::ostream& out, const Image::Header& head) {
141   switch (head.type) {
142     case Image::Header::Type::COLOR:
143       out << "PPM Color ";
144       break;
145     case Image::Header::Type::GRAY:
146       out << "PGM Grayscale ";
147   }
148   out << "Image size " << head.M << " x " << head.N << " and max value of "
      ↪  << head.Q << ".";
149 }
```

Listing 3: Implementation file for the `Histogram` supporting library.

```
1  // Common/histogram_tools.cpp
2  #include "histogram_tools.h"
3
4  #include <algorithm>
5  #include <iostream>
6
7  void Histogram::print(unsigned* histogram, unsigned bins, unsigned width,
   ↪  unsigned height) {
8    // An adjusted histogram, which has been binned
9    unsigned* binnedHistogram = new unsigned[width];
```

```
  // Maximum number of original bins represented by each new bin
  // Each bin is this size, except maybe the last bin (which may be smaller)
  unsigned binSize = 1 + (bins - 1) / width;
  // The maximum number of observations in all bins
  unsigned maxBin = 0;

  // Calculate new binnedHistogram and maxBin
#pragma omp parallel for reduction(max : maxBin)
  for (unsigned i = 0; i < width; i++) {
    binnedHistogram[i] = 0;
    for (unsigned j = binSize * i; j < binSize * (i + 1) && j < bins; j++) {
      binnedHistogram[i] += histogram[j];
    }
    maxBin = std::max(binnedHistogram[i], maxBin);
  }

  // The maximum number of observations each tick can represent
  // May represent as few as 1, if present on the top of a histogram bar
  unsigned tickSize = 1 + (maxBin - 1) / height;

  for (unsigned i = 1; i <= height; i++) {
    unsigned threshold = (height - i) * tickSize;
    for (unsigned j = 0; j < width; j++) {
      if (binnedHistogram[j] > threshold)
        std::cout << '*';
      else
        std::cout << ' ';
    }
    std::cout << '\n';
  }

  delete[] binnedHistogram;
}
```

Listing 4: Implementation file for the `sample` program.

```
#include <iostream>
#include <fstream>
#include <sstream>

#include "../Common/image.h"

/*
  Subsamples and image based on the sampling factor
  @Param: image - the input image that will be sampled
  @Param: subsample_factor - the factor by which to sample
  @Return: void
*/
void subsample_image(Image& image, int subsample_factor){
```

```
14
15    //Todo: add option to not resize image
16    //int M = image.rows / subsample_factor;
17    //int N = image.cols / subsample_factor;
18    //int Q = image.maxVal;
19    //pixelT* pixels;
20
21    //Image newImage = Image(M, N, Q, image.pixels);
22
23    // iterate through image to get the sample
24    for(int i=0; i<image.cols; i += subsample_factor){
25         for(int j=0; j<image.rows; j += subsample_factor) {
26
27           // save the sampled pixel
28           int pixelSample = image[i][j];
29
30           // Modify neighbor pixels to match the sampled pixel
31           for (int k = 0; k < subsample_factor; k++)
32           {
33             for (int l = 0; l < subsample_factor; l++)
34             {
35               image[i + k][j + l] = pixelSample;
36             }
37           }
38       }
39     }
40 }
41
42 int main(int argc, char** argv) {
43
44    int M, N, Q;
45    bool type;
46    int val;
47    int subsample_factor;
48    std::istringstream ss(argv[3]);
49
50    if(ss >> subsample_factor) {
51      if(256 % subsample_factor != 0 || subsample_factor > 256){
52        std::cout << "Error: Subsample factor should be power of 2 less than
          ↪  256" << std::endl;
53        return 1;
54
55      }
56    }
57
58    std::ifstream inFile(argv[1]);
59
60    Image image = Image::read(inFile);
61
62   std::cout << "Question 1: Sampling." << std::endl;
```

```
63
64    subsample_image(image, subsample_factor);
65
66    // Save output image
67    std::ofstream outFile;
68    outFile.open(argv[2]);
69    outFile << image;
70    outFile.close();
71
72    return 0;
73  }
```

Listing 5: Implementation file for the `equalize` program.

```
1   // Q3-Equalization/main.cpp
2   #include <cstring>
3   #include <fstream>
4   #include <iostream>
5   #include <map>
6   #include <mutex>
7
8   #include "../Common/histogram_tools.h"
9   #include "../Common/image.h"
10
11  // Struct for inputting arguments from command line
12  struct Arguments {
13    char *inputImagePath, *outImagePath;
14    Image inputImage;
15    std::ofstream outFile;
16    unsigned histogramWidth = 64, histogramHeight = 10;
17    bool plot = false;
18    std::ofstream plotFile;
19  };
20
21  void equalize(Arguments& arg);
22  bool verifyArguments(int argc, char** argv, Arguments& arg, int& err);
23  void printHelp();
24
25  int main(int argc, char** argv) {
26    int err;
27    Arguments arg;
28
29    if (!verifyArguments(argc, argv, arg, err)) { return err; }
30
31    equalize(arg);
32
33    return 0;
34  }
35
```

```
36 void equalize(Arguments& arg) {
37   unsigned* histogram    = new unsigned[arg.inputImage.maxVal + 1];
38   unsigned* newHistogram = new unsigned[arg.inputImage.maxVal + 1];
39   unsigned* cdf          = new unsigned[arg.inputImage.maxVal + 1];
40   std::mutex* locks      = new std::mutex[arg.inputImage.maxVal + 1];
41   // Initialise histogram bins to be empty
42 #pragma omp parallel for
43   for (unsigned i = 0; i <= arg.inputImage.maxVal; i++) {
44     histogram[i] = newHistogram[i] = 0;
45   }
46
47   // Create histogram
48 #pragma omp parallel for
49   for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++) {
50     unsigned bin = arg.inputImage.pixels[i];
51     locks[bin].lock();
52     histogram[bin]++;
53     locks[bin].unlock();
54   }
55
56   // Calculate CDF
57   cdf[0] = histogram[0];
58   for (unsigned i = 1; i <= arg.inputImage.maxVal; i++) {
59     cdf[i] = cdf[i - 1] + histogram[i];
60   }
61
62   // Tranform image with the CDF
63 #pragma omp parallel for
64   for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++) {
65     Image::pixelT& pixelVal = arg.inputImage.pixels[i];
66     pixelVal                = cdf[pixelVal] * arg.inputImage.maxVal /
67               (arg.inputImage.rows * arg.inputImage.cols);
68   }
69
70   // Write new transformed image out
71   arg.outFile << arg.inputImage;
72   arg.outFile.close();
73
74   // Calculate histogram of new image
75 #pragma omp parallel for
76   for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++) {
77     unsigned bin = arg.inputImage.pixels[i];
78     locks[bin].lock();
79     newHistogram[bin]++;
80     locks[bin].unlock();
81   }
82
83   // Print histograms
84   std::cout << "\nHistogram of input image \"" << arg.inputImagePath <<
     ↪  "\":\n";
```

```
85    Histogram::print(histogram, arg.inputImage.maxVal + 1, arg.histogramWidth,
86                     arg.histogramHeight);
87
88    std::cout << "\nHistogram of output image \"" << arg.outImagePath <<
      ↪  "\":\n";
89    Histogram::print(newHistogram, arg.inputImage.maxVal + 1,
      ↪  arg.histogramWidth,
90                     arg.histogramHeight);
91
92    // Print histogram data for plot file
93    if (arg.plot) {
94      arg.plotFile << "Image Input Equalized\n";
95      for (unsigned i = 0; i <= arg.inputImage.maxVal; i++) {
96        arg.plotFile << i << "    " << histogram[i] << "    " <<
          ↪  newHistogram[i]
97                     << '\n';
98      }
99      arg.plotFile.close();
100   }
101
102   delete[] histogram;
103   delete[] newHistogram;
104   delete[] cdf;
105   delete[] locks;
106 }
107
108 bool verifyArguments(int argc, char** argv, Arguments& arg, int& err) {
109   // If there are not the minimum number of arguments, print help and leave
110   if (argc < 2 ||
111       (argc < 3 && strcmp(argv[1], "-h") && strcmp(argv[1], "--help"))) {
112     std::cout << "Missing operand.\n";
113     err = 1;
114     printHelp();
115     return false;
116   }
117
118   // If the user asks for the help menu, print help and leave
119   if (!strcmp(argv[1], "-h") || !strcmp(argv[1], "--help")) {
120     printHelp();
121     return false;
122   }
123
124   // Find optional argument switches
125   for (unsigned i = 3; i < argc; i++) {
126     if (!strcmp(argv[i], "-width")) {
127       if (i + 1 >= argc) {
128         std::cout << "Missing width";
129         err = 1;
130         printHelp();
131         return false;
```

```
132       }
133
134       arg.histogramWidth = strtoul(argv[i + 1], nullptr, 10);
135       if (arg.histogramWidth == 0) {
136         std::cout << "Width \"" << argv[i + 1]
137                   << "\" could not be recognised as a positive integer.";
138         err = 2;
139         return false;
140       }
141
142       i++;
143     } else if (!strcmp(argv[i], "-height")) {
144       if (i + 1 >= argc) {
145         std::cout << "Missing height";
146         err = 1;
147         break;
148       }
149
150       arg.histogramHeight = strtoul(argv[i + 1], nullptr, 10);
151       if (arg.histogramHeight == 0) {
152         std::cout << "Height \"" << argv[i + 1]
153                   << "\" could not be recognised as a positive integer.";
154         err = 2;
155         return false;
156       }
157
158       i++;
159     } else if (!strcmp(argv[i], "-p")) {
160       if (i + 1 >= argc) {
161         std::cout << "Missing plot output file";
162         err = 1;
163         break;
164       }
165
166       arg.plot = true;
167       arg.plotFile.open(argv[i + 1]);
168
169       if (!arg.plotFile) {
170         std::cout << "Plot file \"" << argv[i + 1]
171                   << "\" could not be opened";
172         err = 2;
173         return false;
174       }
175
176       i++;
177     }
178   }
179
180   // Required arguments
181   arg.inputImagePath = argv[1];
```

```
  std::ifstream inFile(argv[1]);
  try {
    arg.inputImage = Image::read(inFile);
  } catch (std::exception& e) {
    std::cout << "Image \"" << argv[1] << "\"failed to be read: \"" <<
    ↪  e.what()
              << "\"\n";
    err = 2;
    return false;
  }

  arg.outImagePath = argv[2];
  arg.outFile.open(argv[2]);
  if (!arg.outFile) {
    std::cout << "Could not open \"" << argv[2] << "\"\n";
    err = 2;
    return false;
  }

  return true;
}

void printHelp() {
  std::cout
      << "Usage: equalize <image> <output> [options]   (1)\n"
      << "   or: equalize -h                           (2)\n\n"
      << "(1) Take an image file as input, equalize its histogram,\n"
      << "    and write new image to output file. Displays the original\n"
      << "    histogram and the new equalized histogram.\n"
      << "(2) Print this help menu\n\n"
      << "Options:\n"
      << "  -width <width>    Number of visual histogram bins\n"
      << "  -height <height>  Height of visual histogram (in lines)\n"
      << "  -p <file>         Send histogram plotting data to a file for
      ↪  gnuplot\n";
}
```

Listing 6: Implementation file for the `specify` program.

```
#include <cstring>
#include <fstream>
#include <iostream>
#include <mutex>
#include <regex>
#include <vector>

#include "../Common/histogram_tools.h"
#include "../Common/image.h"

```

```cpp
// Struct for inputting arguments from command line
struct Arguments {
  char *inputImagePath, *outImagePath, *histogramPath;
  Image inputImage;
  std::ifstream histogramFile;
  std::ofstream outFile;
  unsigned histogramWidth = 64, histogramHeight = 10;
  bool plot = false;
  std::ofstream plotFile;
};

int specify(Arguments& arg);
void printHistogram(unsigned* histogram, const Arguments& arg);
bool verifyArguments(int argc, char** argv, Arguments& arg, int& err);
void printHelp();

int main(int argc, char** argv) {
  int err;
  Arguments arg;

  if (!verifyArguments(argc, argv, arg, err)) { return err; }

  return specify(arg);
}

int specify(Arguments& arg) {
  unsigned* histogram    = new unsigned[arg.inputImage.maxVal + 1];
  unsigned* newHistogram = new unsigned[arg.inputImage.maxVal + 1];
  unsigned* cdf          = new unsigned[arg.inputImage.maxVal + 1];
  std::mutex* locks      = new std::mutex[arg.inputImage.maxVal + 1];
  std::vector<unsigned> targetHistogram;
  unsigned targetPixels = 0;  // Number of pixels in the target histogram

  // Initialise histogram bins to be empty
#pragma omp parallel for
  for (unsigned i = 0; i <= arg.inputImage.maxVal; i++) {
    histogram[i] = newHistogram[i] = 0;
  }

  // Create input image histogram
#pragma omp parallel for
  for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++) {
    unsigned bin = arg.inputImage.pixels[i];
    locks[bin].lock();
    histogram[bin]++;
    locks[bin].unlock();
  }

  // Start with enough space to hold our input image. If we need more, we
  ↪  can get
```

```
60    // more, but we're probably working with similarly-valued images.
61    targetHistogram.reserve(arg.inputImage.maxVal + 1);
62
63    // Read in target histogram
64    std::string line;
65    std::regex rHistogram("^([[:digit:]]+)[[:space:]]+([[:digit:]]+).*");
66    std::smatch matches;
67    while (arg.histogramFile) {
68      std::getline(arg.histogramFile, line);
69      if (!std::regex_match(line, matches, rHistogram)) continue;
70
71      if (stoul(matches[1].str()) != targetHistogram.size()) {
72        std::cout << "Error in reading histogram file \"" << arg.histogramPath
73                  << "\":\n"
74                  << "Bucket \"" << stoul(matches[1].str())
75                  << "\" was expected to be \"" << targetHistogram.size()
76                  << "\".";
77      }
78
79      targetHistogram.push_back(stoul(matches[2].str()));
80      targetPixels += targetHistogram.back();
81    }
82    arg.histogramFile.close();
83
84    // Calculate CDFs
85    std::vector<unsigned> targetCDF(targetHistogram.size());
86    cdf[0]       = histogram[0];
87    targetCDF[0] = targetHistogram[0];
88
89    for (unsigned i = 1; i <= arg.inputImage.maxVal; i++) {
90      cdf[i] = cdf[i - 1] + histogram[i];
91    }
92    for (unsigned i = 1; i < targetHistogram.size(); i++) {
93      targetCDF[i] = targetCDF[i - 1] + targetHistogram[i];
94    }
95
96  // Tranform input image with its CDF and inverse CDF of target histogram
97  #pragma region CDF transformation
98    // Separate cases for if the images have different dimensions/maxVal, to
      ↪  make
99    // calculation easier
100   if (arg.inputImage.maxVal == targetHistogram.size() - 1) {
101     if (arg.inputImage.rows * arg.inputImage.cols == targetPixels) {
102 #pragma omp parallel for
103       for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols;
          ↪  i++) {
104         Image::pixelT& pixelVal = arg.inputImage.pixels[i];
105
106         unsigned inversePixel = cdf[pixelVal];
107
```

```
        pixelVal = targetCDF.rend() -
                   std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
                                    inversePixel, std::greater<unsigned>());
      }
    } else {
#pragma omp parallel for
      for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols;
       ↪  i++) {
        Image::pixelT& pixelVal = arg.inputImage.pixels[i];

        unsigned inversePixel = cdf[pixelVal] * targetPixels /
                                (arg.inputImage.rows * arg.inputImage.cols);
        pixelVal = targetCDF.rend() -
                   std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
                                    inversePixel, std::greater<unsigned>());
      }
    }
  } else if (arg.inputImage.rows * arg.inputImage.cols == targetPixels) {
#pragma omp parallel for
    for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++)
     ↪  {
      Image::pixelT& pixelVal = arg.inputImage.pixels[i];

      unsigned inversePixel =
          cdf[pixelVal] * arg.inputImage.maxVal / (targetHistogram.size() -
           ↪  1);
      pixelVal = targetCDF.rend() -
                 std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
                                  inversePixel, std::greater<unsigned>());
    }
  } else {
    // In this case, we need to do math with ull because of the
     ↪  multiplications
    // overflowing The result after division should fit within an unsigned,
     ↪  though
#pragma omp parallel for
    for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++)
     ↪  {
      Image::pixelT& pixelVal = arg.inputImage.pixels[i];

      unsigned inversePixel = ((unsigned long long) cdf[pixelVal]) *
                              arg.inputImage.maxVal * targetPixels /
                              (arg.inputImage.rows * arg.inputImage.cols *
                               (targetHistogram.size() - 1));
      pixelVal = targetCDF.rend() -
                 std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
                                  inversePixel, std::greater<unsigned>());
    }
  }
#pragma endregion CDF transformation
```

```
152
153   // Write new transformed image out
154   arg.outFile << arg.inputImage;
155   arg.outFile.close();
156
157   // Calculate histogram of new image
158 #pragma omp parallel for
159   for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++) {
160     unsigned bin = arg.inputImage.pixels[i];
161     locks[bin].lock();
162     newHistogram[bin]++;
163     locks[bin].unlock();
164   }
165
166   // Print histograms
167   std::cout << "\nHistogram of input image \"" << arg.inputImagePath <<
      ↪  "\":\n";
168   Histogram::print(histogram, arg.inputImage.maxVal + 1, arg.histogramWidth,
169                    arg.histogramHeight);
170
171   std::cout << "\nInput histogram \"" << arg.histogramPath << "\":\n";
172   Histogram::print(&targetHistogram.front(), targetHistogram.size(),
173                    arg.histogramWidth, arg.histogramHeight);
174
175   std::cout << "\nHistogram of output image \"" << arg.outImagePath <<
      ↪  "\":\n";
176   Histogram::print(newHistogram, arg.inputImage.maxVal + 1,
      ↪  arg.histogramWidth,
177                    arg.histogramHeight);
178
179   // Print histogram data for plot file
180   if (arg.plot) {
181     arg.plotFile << "Source Input-Image Input-Histogram Specified\n";
182     unsigned i;
183     for (i = 0; i <= arg.inputImage.maxVal && i < targetHistogram.size();
        ↪  i++) {
184       arg.plotFile << i << "    " << histogram[i] << "    "
185                    << targetHistogram[i] << "    " << newHistogram[i] <<
                        ↪  '\n';
186     }
187     arg.plotFile.close();
188   }
189
190   delete[] histogram;
191   delete[] newHistogram;
192   delete[] cdf;
193   delete[] locks;
194
195   return 0;
196 }
```

```
197
198 bool verifyArguments(int argc, char** argv, Arguments& arg, int& err) {
199   if (argc < 2 ||
200       (argc < 4 && strcmp(argv[1], "-h") && strcmp(argv[1], "--help"))) {
201     std::cout << "Missing operand.\n";
202     err = 1;
203     printHelp();
204     return false;
205   }
206
207   if (!strcmp(argv[1], "-h") || !strcmp(argv[1], "--help")) {
208     printHelp();
209     return false;
210   }
211
212   // Find optional argument switches
213   for (unsigned i = 4; i < argc; i++) {
214     if (!strcmp(argv[i], "-width")) {
215       if (i + 1 >= argc) {
216         std::cout << "Missing width";
217         err = 1;
218         printHelp();
219         return false;
220       }
221
222       arg.histogramWidth = strtoul(argv[i + 1], nullptr, 10);
223       if (arg.histogramWidth == 0) {
224         std::cout << "Width \"" << argv[i + 1]
225                   << "\" could not be recognised as a positive integer.";
226         err = 2;
227         return false;
228       }
229
230       i++;
231     } else if (!strcmp(argv[i], "-height")) {
232       if (i + 1 >= argc) {
233         std::cout << "Missing height";
234         err = 1;
235         break;
236       }
237
238       arg.histogramHeight = strtoul(argv[i + 1], nullptr, 10);
239       if (arg.histogramHeight == 0) {
240         std::cout << "Height \"" << argv[i + 1]
241                   << "\" could not be recognised as a positive integer.";
242         err = 2;
243         return false;
244       }
245
246       i++;
```

```
247     } else if (!strcmp(argv[i], "-p")) {
248       if (i + 1 >= argc) {
249         std::cout << "Missing plot output file";
250         err = 1;
251         break;
252       }
253
254       arg.plot = true;
255       arg.plotFile.open(argv[i + 1]);
256
257       if (!arg.plotFile) {
258         std::cout << "Plot file \"" << argv[i + 1]
259                   << "\" could not be opened";
260         err = 2;
261         return false;
262       }
263
264       i++;
265     }
266   }
267
268   // Required arguments
269   arg.inputImagePath = argv[1];
270   std::ifstream inFile(argv[1]);
271   try {
272     arg.inputImage = Image::read(inFile);
273   } catch (std::exception& e) {
274     std::cout << "Image \"" << argv[1] << "\"failed to be read: \"" <<
      ↪  e.what()
275               << "\"\n";
276     err = 2;
277     return false;
278   }
279
280   arg.histogramPath = argv[2];
281   arg.histogramFile.open(argv[2]);
282   if (!arg.histogramFile) {
283     std::cout << "Could not open \"" << argv[2] << "\"\n";
284     err = 2;
285     return false;
286   }
287
288   arg.outImagePath = argv[3];
289   arg.outFile.open(argv[3]);
290   if (!arg.outFile) {
291     std::cout << "Could not open \"" << argv[3] << "\"\n";
292     err = 2;
293     return false;
294   }
295
```

```
296   return true;
297 }
298
299 void printHelp() {
300   std::cout
301       << "Usage: specify <image> <histogram> <output> [options]   (1)\n"
302       << "   or: specify -h                                       (2)\n\n"
303       << "(1) Take an image file as input, change its histogram to the\n"
304       << "    specified histogram, and write new image to output file.\n"
305       << "    Displays the original histogram and the new equalized
          ↪  histogram.\n"
306       << "    Histogram files can be obtained by running 'equalize' with\n"
307       << "    the -p flag set (or 'specify' with the -p falg set).\n"
308       << "(2) Print this help menu\n\n"
309       << "Options:\n"
310       << "  -width <width>    Number of visual histogram bins\n"
311       << "  -height <height>  Height of visual histogram (in lines)\n"
312       << "  -p <file>         Send histogram plotting data to a file for
          ↪  gnuplot\n";
313 }
```

Listing 7: `gnuplot` plotting file for generating two-histogram comparison plots. Used for generating comparison plots in section 3.3.

```
1  # A gnuplot plotting file to plot the two histograms of data from equalize
   ↪  with the -p switch
2  if (!exists("outfile")) outfile='plot.eps'
3
4  if (!exists("imageName")) {
5    set title "Comparison of histograms of input and output (equalized)
     ↪  images"
6  } else {
7    set title "Comparison of histograms of " . imageName . " and equalized
     ↪  image" noenhanced
8  }
9
10 set terminal postscript eps enhanced color size 6,3
11 set output outfile
12 set style data histogram
13 set style histogram cluster gap 1
14 set style fill solid
15 set boxwidth 0.9
16
17 unset xtics
18
19 plot infile using 2:xtic(1) ti col linecolor rgb "#1b9e77", '' u 3 ti col
   ↪  linecolor rgb "#d95f02"
```

Listing 8: `gnuplot` plotting file for generating three-histogram comparison plots. Used for generating comparison plots in section 4.3.

```
# A gnuplot plotting file to plot the three histograms of data from specify with the -p switch
if (!exists("outfile")) outfile='plot.eps'

if (!exists("imageName") || !exists("histoName")) {
  set title "Comparison of histograms of input image, input histogram, and output image"
} else {
  set title "Comparison of histograms of " . imageName . ", " . histoName . ", and specified image" noenhanced
}

set terminal postscript eps enhanced color
set output outfile
set style data histogram
set style histogram cluster gap 1
set style fill solid
set boxwidth 0.9

unset xtics

plot infile using 2:xtic(1) ti imageName noenhanced linecolor rgb "#1b9e77" , '' u 3 ti histoName noenhanced linecolor rgb "#d95f02", '' u 4 ti col linecolor rgb "#7570b3"
```