Programming Assignment 1

CS 474

Alexander Novotny Matthew Lyman Page

September 23, 2020

Contents

1 Image Sampling

2 Image Quantization

3 Histogram Equalization

3.1 Theory

3.2 Implementation

3.3 Results and Discussion

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

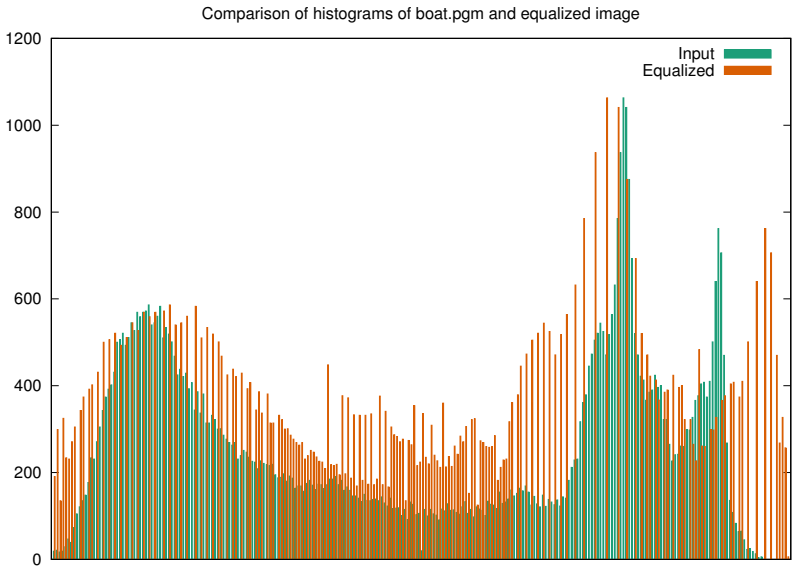

Figure 2: A comparison of histograms of `boat.pgm` and its equalised version

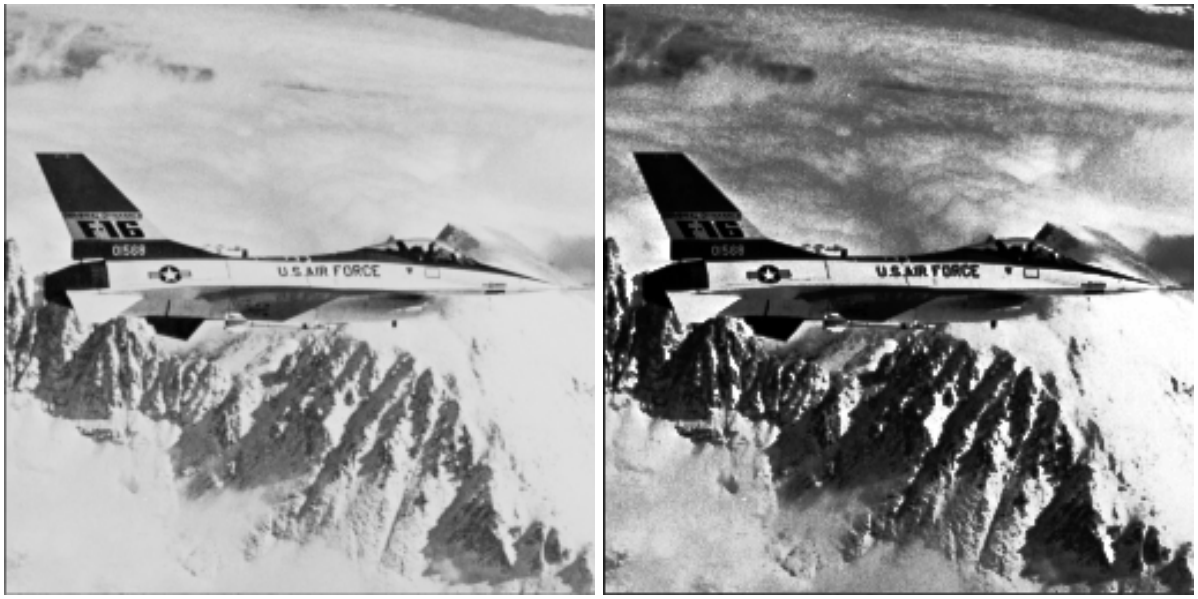

Figure 3: A comparison of `f_16.pgm` with its equalization (right).

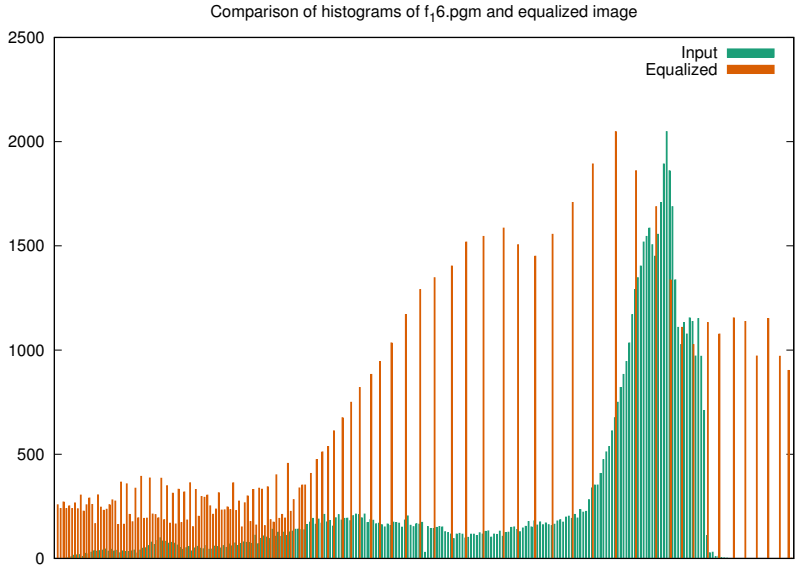

Figure 4: A comparison of histograms of `f_16.pgm` and its equalised version

4 Histogram Specification

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

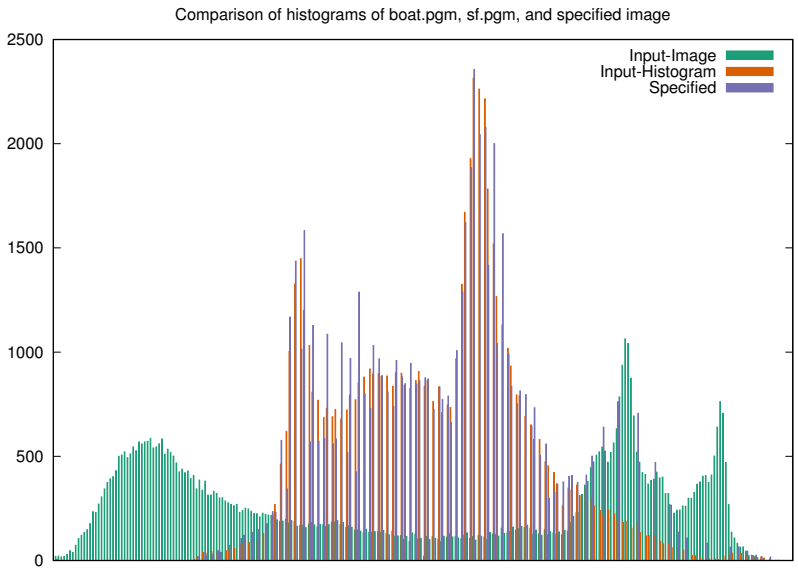

Figure 6: A comparison of histograms of `boat.pgm`, `sf.pgm`, and the specified output above.

Code Listings

Listing 1: Header file for the `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
40   // 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 `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)
    : 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;
```

```
  Header re;

  // read header
  in.getline(header, 100, '\n');
  if ((header[0] == 'P') && (header[1] == '5')) {
    re.type = GRAY;
  } else if ((header[0] == 'P') && (header[1] == '6')) {
    re.type = COLOR;
  } else
    throw std::runtime_error("Image is not PGM or PPM!");

  in.getline(header, 100, '\n');
  while (header[0] == '#') in.getline(header, 100, '\n');

  re.N = strtol(header, &ptr, 0);
  re.M = atoi(ptr);

  in.getline(header, 100, '\n');

  re.Q = strtol(header, &ptr, 0);

  return re;
}

std::ostream& operator<<(std::ostream& out, const Image::Header& head) {
  switch (head.type) {
    case Image::Header::Type::COLOR:
      out << "PPM Color ";
      break;
    case Image::Header::Type::GRAY:
      out << "PGM Grayscale ";
  }
  out << "Image size " << head.M << " x " << head.N << " and max value of "
  ↪  << head.Q << ".";
}
```

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

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

#include <algorithm>
#include <iostream>

void Histogram::print(unsigned* histogram, unsigned bins, unsigned width,
↪  unsigned height) {
  // An adjusted histogram, which has been binned
  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 `equalize` program.

```
// Q3-Equalization/main.cpp
#include <cstring>
#include <fstream>
#include <iostream>
#include <map>
#include <mutex>

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

// Struct for inputting arguments from command line
struct Arguments {
  char *inputImagePath, *outImagePath;
  Image inputImage;
```

```
  std::ofstream outFile;
  unsigned histogramWidth = 64, histogramHeight = 10;
  bool plot = false;
  std::ofstream plotFile;
};

void equalize(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; }

  equalize(arg);

  return 0;
}

void equalize(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];
  // 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 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();
  }

  // Calculate CDF
  cdf[0] = histogram[0];
  for (unsigned i = 1; i <= arg.inputImage.maxVal; i++) {
    cdf[i] = cdf[i - 1] + histogram[i];
  }

  // Tranform image with the CDF
#pragma omp parallel for
  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 ||
```

```
    (argc < 3 && strcmp(argv[1], "-h") && strcmp(argv[1], "--help"))) {
  std::cout << "Missing operand.\n";
  err = 1;
  printHelp();
  return false;
}

// If the user asks for the help menu, print help and leave
if (!strcmp(argv[1], "-h") || !strcmp(argv[1], "--help")) {
  printHelp();
  return false;
}

// Find optional argument switches
for (unsigned i = 3; i < argc; i++) {
  if (!strcmp(argv[i], "-width")) {
    if (i + 1 >= argc) {
      std::cout << "Missing width";
      err = 1;
      printHelp();
      return false;
    }

    arg.histogramWidth = strtoul(argv[i + 1], nullptr, 10);
    if (arg.histogramWidth == 0) {
      std::cout << "Width \"" << argv[i + 1]
                << "\" could not be recognised as a positive integer.";
      err = 2;
      return false;
    }

    i++;
  } else if (!strcmp(argv[i], "-height")) {
    if (i + 1 >= argc) {
      std::cout << "Missing height";
      err = 1;
      break;
    }

    arg.histogramHeight = strtoul(argv[i + 1], nullptr, 10);
    if (arg.histogramHeight == 0) {
      std::cout << "Height \"" << argv[i + 1]
                << "\" could not be recognised as a positive integer.";
      err = 2;
      return false;
    }

    i++;
  } else if (!strcmp(argv[i], "-p")) {
    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];
182   std::ifstream inFile(argv[1]);
183   try {
184     arg.inputImage = Image::read(inFile);
185   } catch (std::exception& e) {
186     std::cout << "Image \"" << argv[1] << "\"failed to be read: \"" <<
      ↪  e.what()
187               << "\"\n";
188     err = 2;
189     return false;
190   }
191
192   arg.outImagePath = argv[2];
193   arg.outFile.open(argv[2]);
194   if (!arg.outFile) {
195     std::cout << "Could not open \"" << argv[2] << "\"\n";
196     err = 2;
197     return false;
198   }
199
200   return true;
201 }
202
203 void printHelp() {
204   std::cout
205       << "Usage: equalize <image> <output> [options]   (1)\n"
206       << "   or: equalize -h                           (2)\n\n"
207       << "(1) Take an image file as input, equalize its histogram,\n"
208       << "    and write new image to output file. Displays the original\n"
209       << "    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 5: 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"

// 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];
```

```
40    std::mutex* locks      = new std::mutex[arg.inputImage.maxVal + 1];
41    std::vector<unsigned> targetHistogram;
42    unsigned targetPixels = 0;  // Number of pixels in the target histogram
43
44    // Initialise histogram bins to be empty
45  #pragma omp parallel for
46    for (unsigned i = 0; i <= arg.inputImage.maxVal; i++) {
47      histogram[i] = newHistogram[i] = 0;
48    }
49
50    // Create input image histogram
51  #pragma omp parallel for
52    for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++) {
53      unsigned bin = arg.inputImage.pixels[i];
54      locks[bin].lock();
55      histogram[bin]++;
56      locks[bin].unlock();
57    }
58
59    // 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() -
108                     std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
109                                      inversePixel, std::greater<unsigned>());
110       }
111     } else {
112 #pragma omp parallel for
113       for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols;
          ↪  i++) {
114         Image::pixelT& pixelVal = arg.inputImage.pixels[i];
115
116         unsigned inversePixel = cdf[pixelVal] * targetPixels /
117                                 (arg.inputImage.rows * arg.inputImage.cols);
118         pixelVal = targetCDF.rend() -
119                    std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
120                                     inversePixel, std::greater<unsigned>());
121       }
122     }
123   } else if (arg.inputImage.rows * arg.inputImage.cols == targetPixels) {
124 #pragma omp parallel for
125     for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++)
      ↪  {
126       Image::pixelT& pixelVal = arg.inputImage.pixels[i];
127
128       unsigned inversePixel =
129           cdf[pixelVal] * arg.inputImage.maxVal / (targetHistogram.size() -
            ↪  1);
130       pixelVal = targetCDF.rend() -
131                  std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
132                                   inversePixel, std::greater<unsigned>());
133     }
```

```
134   } else {
135     // In this case, we need to do math with ull because of the
      ↪  multiplications
136     // overflowing The result after division should fit within an unsigned,
      ↪  though
137 #pragma omp parallel for
138     for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++)
      ↪  {
139       Image::pixelT& pixelVal = arg.inputImage.pixels[i];
140
141       unsigned inversePixel = ((unsigned long long) cdf[pixelVal]) *
142                               arg.inputImage.maxVal * targetPixels /
143                               (arg.inputImage.rows * arg.inputImage.cols *
144                                (targetHistogram.size() - 1));
145       pixelVal = targetCDF.rend() -
146                  std::lower_bound(targetCDF.rbegin(), targetCDF.rend(),
147                                   inversePixel, std::greater<unsigned>());
148     }
149   }
150 #pragma endregion CDF transformation
151
152   // Write new transformed image out
153   arg.outFile << arg.inputImage;
154   arg.outFile.close();
155
156   // Calculate histogram of new image
157 #pragma omp parallel for
158   for (unsigned i = 0; i < arg.inputImage.rows * arg.inputImage.cols; i++) {
159     unsigned bin = arg.inputImage.pixels[i];
160     locks[bin].lock();
161     newHistogram[bin]++;
162     locks[bin].unlock();
163   }
164
165   // Print histograms
166   std::cout << "\nHistogram of input image \"" << arg.inputImagePath <<
    ↪  "\":\n";
167   Histogram::print(histogram, arg.inputImage.maxVal + 1, arg.histogramWidth,
168                    arg.histogramHeight);
169
170   std::cout << "\nInput histogram \"" << arg.histogramPath << "\":\n";
171   Histogram::print(&targetHistogram.front(), targetHistogram.size(),
172                    arg.histogramWidth, arg.histogramHeight);
173
174   std::cout << "\nHistogram of output image \"" << arg.outImagePath <<
    ↪  "\":\n";
175   Histogram::print(newHistogram, arg.inputImage.maxVal + 1,
    ↪  arg.histogramWidth,
176                    arg.histogramHeight);
177
```

```
178   // Print histogram data for plot file
179   if (arg.plot) {
180     arg.plotFile << "Source Input-Image Input-Histogram Specified\n";
181     unsigned i;
182     for (i = 0; i <= arg.inputImage.maxVal && i < targetHistogram.size();
      ↪  i++) {
183       arg.plotFile << i << "    " << histogram[i] << "    "
184                    << targetHistogram[i] << "    " << newHistogram[i] <<
                      ↪  '\n';
185     }
186     arg.plotFile.close();
187   }
188
189   delete[] histogram;
190   delete[] newHistogram;
191   delete[] cdf;
192   delete[] locks;
193
194   return 0;
195 }
196
197 bool verifyArguments(int argc, char** argv, Arguments& arg, int& err) {
198   if (argc < 2 ||
199       (argc < 4 && strcmp(argv[1], "-h") && strcmp(argv[1], "--help"))) {
200     std::cout << "Missing operand.\n";
201     err = 1;
202     printHelp();
203     return false;
204   }
205
206   if (!strcmp(argv[1], "-h") || !strcmp(argv[1], "--help")) {
207     printHelp();
208     return false;
209   }
210
211   // Find optional argument switches
212   for (unsigned i = 4; i < argc; i++) {
213     if (!strcmp(argv[i], "-width")) {
214       if (i + 1 >= argc) {
215         std::cout << "Missing width";
216         err = 1;
217         printHelp();
218         return false;
219       }
220
221       arg.histogramWidth = strtoul(argv[i + 1], nullptr, 10);
222       if (arg.histogramWidth == 0) {
223         std::cout << "Width \"" << argv[i + 1]
224                   << "\" could not be recognised as a positive integer.";
225         err = 2;
```

```
226         return false;
227       }
228
229       i++;
230     } else if (!strcmp(argv[i], "-height")) {
231       if (i + 1 >= argc) {
232         std::cout << "Missing height";
233         err = 1;
234         break;
235       }
236
237       arg.histogramHeight = strtoul(argv[i + 1], nullptr, 10);
238       if (arg.histogramHeight == 0) {
239         std::cout << "Height \"" << argv[i + 1]
240                   << "\" could not be recognised as a positive integer.";
241         err = 2;
242         return false;
243       }
244
245       i++;
246     } else if (!strcmp(argv[i], "-p")) {
247       if (i + 1 >= argc) {
248         std::cout << "Missing plot output file";
249         err = 1;
250         break;
251       }
252
253       arg.plot = true;
254       arg.plotFile.open(argv[i + 1]);
255
256       if (!arg.plotFile) {
257         std::cout << "Plot file \"" << argv[i + 1]
258                   << "\" could not be opened";
259         err = 2;
260         return false;
261       }
262
263       i++;
264     }
265   }
266
267   // Required arguments
268   arg.inputImagePath = argv[1];
269   std::ifstream inFile(argv[1]);
270   try {
271     arg.inputImage = Image::read(inFile);
272   } catch (std::exception& e) {
273     std::cout << "Image \"" << argv[1] << "\"failed to be read: \"" <<
      ↪  e.what()
274               << "\"\n";
```

```
275     err = 2;
276     return false;
277   }
278
279   arg.histogramPath = argv[2];
280   arg.histogramFile.open(argv[2]);
281   if (!arg.histogramFile) {
282     std::cout << "Could not open \"" << argv[2] << "\"\n";
283     err = 2;
284     return false;
285   }
286
287   arg.outImagePath = argv[3];
288   arg.outFile.open(argv[3]);
289   if (!arg.outFile) {
290     std::cout << "Could not open \"" << argv[3] << "\"\n";
291     err = 2;
292     return false;
293   }
294
295   return true;
296 }
297
298 void printHelp() {
299   std::cout
300       << "Usage: specify <image> <histogram> <output> [options]   (1)\n"
301       << "   or: specify -h                                       (2)\n\n"
302       << "(1) Take an image file as input, change its histogram to the\n"
303       << "    specified histogram, and write new image to output file.\n"
304       << "    Displays the original histogram and the new equalized
      ↪  histogram.\n"
305       << "    Histogram files can be obtained by running 'equalize' with\n"
306       << "    the -p flag set (or 'specify' with the -p falg set).\n"
307       << "(2) Print this help menu\n\n"
308       << "Options:\n"
309       << "  -width <width>    Number of visual histogram bins\n"
310       << "  -height <height>  Height of visual histogram (in lines)\n"
311       << "  -p <file>         Send histogram plotting data to a file for
      ↪  gnuplot\n";
312 }
```

Listing 6: `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")) {
```

```
  set title "Comparison of histograms of input and output (equalized)
  ↪ images"
} else {
  set title "Comparison of histograms of " . imageName . " and equalized
  ↪ image"
}

# set term png size 1280,960
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 col linecolor rgb "#1b9e77", '' u 3 ti col
↪ linecolor rgb "#d95f02"
```

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

```
# 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")) {
  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"
}

# set term png size 1280,960
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 col linecolor rgb "#1b9e77", '' u 3 ti col
↪ linecolor rgb "#d95f02", '' u 4 ti col linecolor rgb "#7570b3"
```