Programming Assignment 1

CS 474

Alexander Novotny Matthew Lyman Page

September 22, 2020

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).

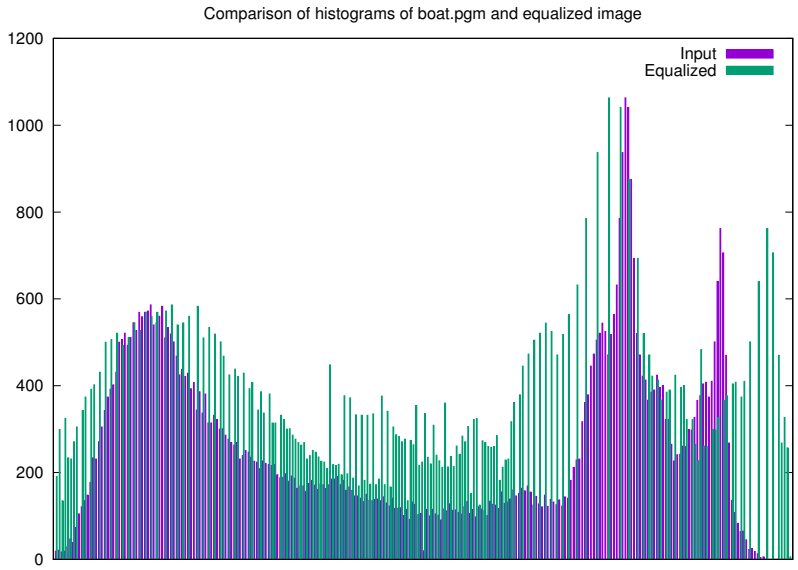

Figure 2: A comparison of histograms of `boat.pgm` and its equalised version

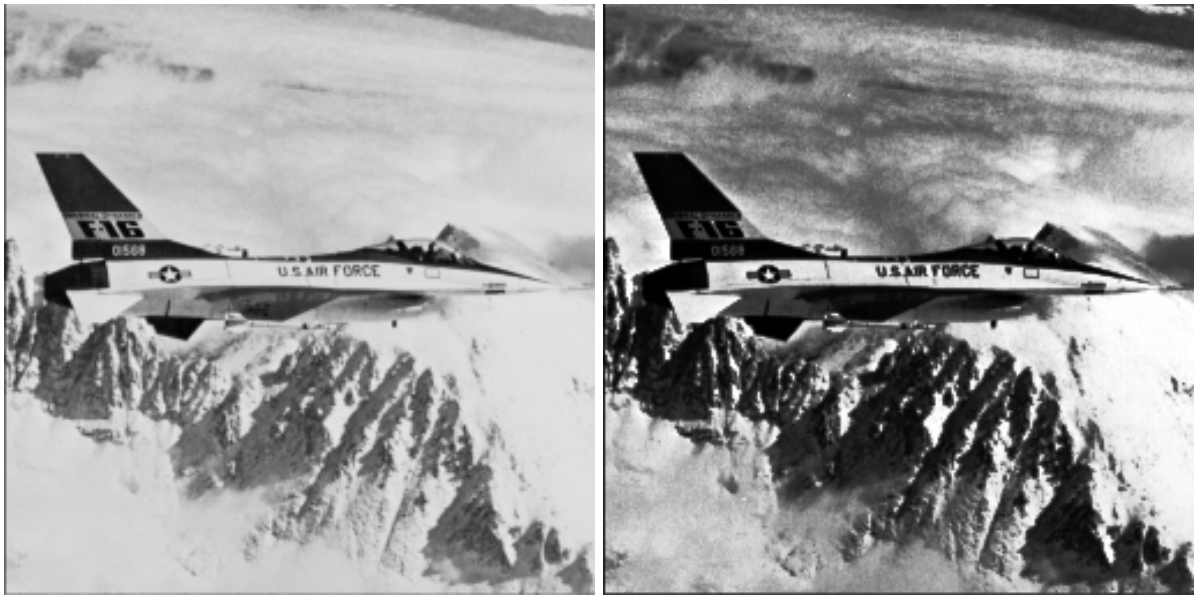

Figure 3: A comparison of `f_16.pgm` with its equalization (right).

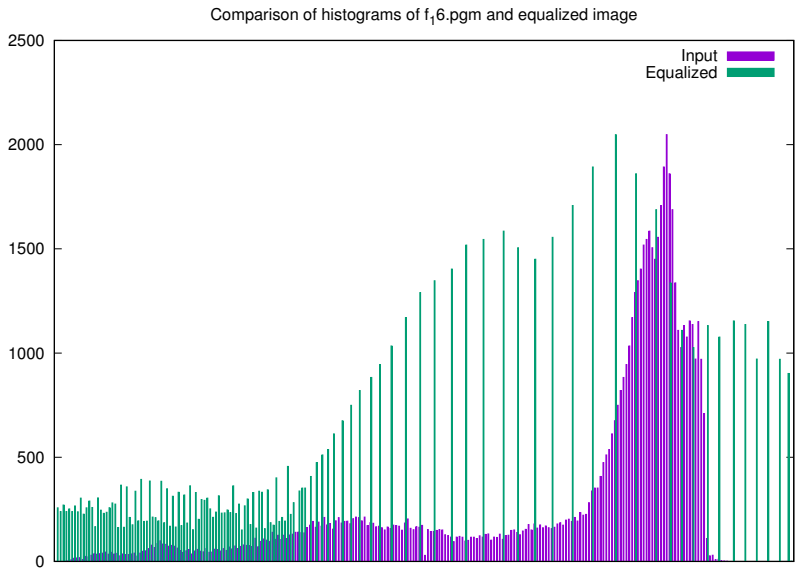

Figure 4: A comparison of histograms of `f_16.pgm` and its equalised version

4 Histogram Specification

Code Listings

Common

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

```
// Common/image.h
#pragma once

#include <iostream>

class Image {
public:
    // The type that is used for the value of each pixel
    // As of right now, read and operator<< only work if it is one byte
    ↪  large
    typedef unsigned char pixelT;
    // Struct for reading just the header of an image
    struct ImageHeader {
        enum Type {
            COLOR,
            GRAY,
        } type;

        unsigned M, N, Q;

        // Read header from file
        // Throws std::runtime_error for any errors encountered,
        // such as not having a valid PGM/PPM header
        static ImageHeader read(std::istream &in);
    };

    Image();
    Image(unsigned, unsigned, unsigned);
    Image(const Image &);  // Copy constructor
    Image(Image &&);       // Move constructor
    ~Image();

    // Read from stream (such as file)
    // Throws std::runtime_error for any errors encountered,
    // such as not being a valid PGM image
    static Image read(std::istream &in);

    // Output to stream (such as file)
    friend std::ostream &operator<<(std::ostream &out, const Image &im);

    // 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
```

```
45
46      // Read-only properties
47      pixelT *const &pixels  = pixelValue;
48      const unsigned &rows   = M;
49      const unsigned &cols   = N;
50      const unsigned &maxVal = Q;
51
52  private:
53      Image(unsigned, unsigned, unsigned, pixelT *);
54      unsigned M, N, Q;
55      pixelT *pixelValue;
56  };
57
58  std::ostream &operator<<(std::ostream &out, const Image::ImageHeader &head);
59
60  #endif
```

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

```
1   // Common/image.cpp
2   #include "image.h"
3
4   #include <cassert>
5   #include <cstdlib>
6   #include <exception>
7
8   Image::Image() : Image(0, 0, 0, nullptr) {}
9
10  Image::Image(unsigned M, unsigned N, unsigned Q) : Image(M, N, Q, new
    ↪  Image::pixelT[M * N]) {}
11
12  Image::Image(const Image& oldImage) : Image(oldImage.M, oldImage.N,
    ↪  oldImage.Q) {
13      for (unsigned i = 0; i < M * N; i++) { pixelValue[i] =
        ↪  oldImage.pixelValue[i]; }
14  }
15
16  // Move constructor - take old image's pixel values and make old image
    ↪  invalid
17  Image::Image(Image&& oldImage) : Image(oldImage.M, oldImage.N, oldImage.Q,
    ↪  oldImage.pixelValue) {
18      oldImage.M = oldImage.N = oldImage.Q = 0;
19      oldImage.pixelValue                  = nullptr;
20  }
21
22  Image::Image(unsigned M, unsigned N, unsigned Q, pixelT* pixels)
23      : M(M), N(N), Q(Q), pixelValue(pixels) {}
24
25  Image::~Image() {
```

```
    if (pixelValue != nullptr) { delete[] pixelValue; }
}

// Slightly modified version of readImage() function provided by Dr. Bebis
Image Image::read(std::istream& in) {
    int N, M, Q;
    unsigned char* charImage;
    char header[100], *ptr;

    static_assert(sizeof(Image::pixelT) == 1,
                  "Image reading only supported for single-byte pixel
                  ↪  types.");

    // read header
    in.getline(header, 100, '\n');
    if ((header[0] != 'P') || (header[1] != '5')) { throw
    ↪  std::runtime_error("Image is not PGM!"); }

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

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

    in.getline(header, 100, '\n');
    Q = strtol(header, &ptr, 0);

    if (Q > 255) throw std::runtime_error("Image cannot be read correctly (Q
    ↪  > 255)!");

    charImage = new unsigned char[M * N];

    in.read(reinterpret_cast<char*>(charImage), (M * N) * sizeof(unsigned
    ↪  char));

    if (in.fail()) throw std::runtime_error("Image has wrong size!");

    return Image(M, N, Q, charImage);
}

// Slightly modified version of writeImage() function provided by Dr. Bebis
std::ostream& operator<<(std::ostream& out, const Image& im) {
    static_assert(sizeof(Image::pixelT) == 1,
                  "Image writing only supported for single-byte pixel
                  ↪  types.");

    out << "P5" << std::endl;
    out << im.N << " " << im.M << std::endl;
    out << im.Q << std::endl;

```

```
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::ImageHeader Image::ImageHeader::read(std::istream& in) {
114     unsigned char* charImage;
115     char header[100], *ptr;
116     ImageHeader 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::ImageHeader& head)
    ↪ {
141     switch (head.type) {
142         case Image::ImageHeader::Type::COLOR:
143             out << "PPM Color ";
144             break;
145         case Image::ImageHeader::Type::GRAY:
146             out << "PGM Grayscale ";
147     }
148     out << "Image size " << head.M << " x " << head.N << " and max value of
        ↪  " << head.Q << ".";
149 }
```