Comp40 Homework 4: Arith Design Document

Names: Isabelle Lai (ilai01) and Andrea Foo (afoo01)

Table of Contents:

IMPLEMENTATION PLAN

1. Implement module to trim width and height of image
2. Implement module to:
    a. Convert RGB value to floating-point representation
    b. Convert RGB floats into RGB integers and put RGB values of each pixel into pixel map
3. Implement module to:
    a. Transform RGB floats into component video color space
    b. Transform pixel from component video color to RGB floats
4. Implement module to take and store average Pb and Pr values
5. Implement module to:
    a. Transform Pb, Pr value floats to four-bit values
    b. Transform four-bit values into Pb, Pr floats
6. Implement module to:
    a. Transform four Y values into a, b, c, d cosine coefficients (floats)
    b. Compute Y1, Y2, Y3, Y4 for each pixel in the block from a, b, c, d
7. Implement module to:
    a. Convert b, c, d from floating-point representation into five-bit signed values
    b. Convert b, c, d from five-bit signed values into floating-point representation
8. Implement module to:
    a. Convert a from floating-point representation into into nine unsigned bits
    b. Convert a from nine unsigned bits into floating-point representation
9. Implement bitpack module, which will be used to:
    a. Pack a, b, c, d, Pb, and Pr into a 32-bit word
    b. Unpack 32-bit word into a, b, c, d, Pb, and Pr
10. Implement module to write compressed binary image to standard output
11. Implement code to allocate new 2D array of pixels
12. Implement module to read 32-bit code words in sequence
13. Implement module to read header (in 40image.c)

APPENDIX A: COMPRESSION STEPS WITH DETAILED EXPLANATION AND TESTING PLAN AND DECOMPRESSION STEPS DETAILED EXPLANATION WITH TESTING PLAN

Compression

1. Name: Trim width and height of image (in trim.c)
   Description: Trim width and height of image to even numbers (if necessary)
   Input: The original Pnm_ppm image
   Output: The trimmed Pnm_ppm image
   Loss: The trimmed pixels on the edge of the image
   Testing Plan:
   - Input image with both even and odd widths and heights
   - Check that outputted images have even widths and heights

----

2. Name: Transform RGB values to floating-point representation
   Description: Turn the RGB scaled integers to floating point representation
   Input: Each pixel of the Pnm_pm image as a Pnm_rgb struct
   Output: Each pixel of the Pnm_ppm image as a struct containing RGB floats that we will implement.
   Loss: Minimal loss because we are converting to floats
   Testing Plan:
   - Implement the corresponding decompression step that converts RGB floats back to scaled integers
   - Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
   - Test input images:
     - All red image
     - All blue image
     - All green image
     - All black image
     - All white image
     - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

----

3. Name: Transform RGB floats into component video color space
   Description: Transform the image of RGB float structs into an image where each pixel is composed of Y, Pb, and Pr value float structs.
   Input: Image with RGB float pixels
   Output: Image with Y, Pb, Pr float pixels.
   Loss: No loss because it is a linear transformation
   Testing Plan:

- Implement the corresponding decompression step that transforms component video color space pixels to RGB floats
- Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
- Test input images:
    - All red image
    - All blue image
    - All green image
    - All black image
    - All white image
    - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

---

4. Name: Take and store average values of Pb and Pr
   Description: Take the average value of the Pb and Pr for the four pixels in each 2 x 2 block
   Input: Pb and Pr elements of the image by block (four pixels)
   Output: The average Pb and Pr values for each block
   Loss: Some loss of chroma (color) data due to averaging of the values
   Testing Plan:
    - Implement the corresponding decompression step that stores the averaged Pb and Pr values as the Pb and Pr values for each individual component video pixel
    - Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
    - Test input images:
        - All red image
        - All blue image
        - All green image
        - All black image
        - All white image
        - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

---

5. Name: Transform Pb, Pr value floats to four-bit values
   Description: Using provided Arith40_index_of_chroma(float x) function, convert a chroma value between -0.5 and +0.5 to a 4-bit quantized representation
   Input: Pb and Pr float values
   Output: 4-bit quantized representations of these values
   Loss: We lose data because we are compressing the float into an unsigned 4 bit value.
   Testing Plan:

- Implement the corresponding decompression step that transforms four-bit values into floating-point Pb and Pr values
- Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
- Test input images:
    - All red image
    - All blue image
    - All green image
    - All black image
    - All white image
    - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

----

6. Name: Transform four Y values into a, b, c, d cosine coefficients (floats)
   Description: Use discrete cosine transform to convert the four Y values of a 2 x 2 pixel block into a, b, c, d cosine coefficients.
   Input: Y1, Y2, Y3, Y4
   Output: a, b, c, d
   Loss: No loss because it is a linear transformation
   Testing Plan:
    - Implement the corresponding decompression step that transforms a, b, c, d back into Y values
    - Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
    - Test input images:
        - All red image
        - All blue image
        - All green image
        - All black image
        - All white image
        - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

----

7. Name: Convert b, c, d into five-bit signed values
   Description: Assuming that b, c, d are in the range of -0.3 to +0.3, convert them to signed five-bit scaled integers.
   Input: floating point representations of b, c, and d for each pixel in the image
   Output: five bit signed values of b, c, and d for each pixel in the image
   Loss: 32 bits -> 5 bits compression.
   Testing Plan:

- Implement the corresponding decompression step that converse five-bit values back into floating-point representations of b, c, d
- Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
- Test input images:
    - All red image
    - All blue image
    - All green image
    - All black image
    - All white image
    - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

----

8. Name: Convert a into nine unsigned bits
   Description: Multiply by 511 and round to code a into nine unsigned bits
   Input: floating point representation of a for each pixel in the image
   Output: nine bit unsigned value of a for each pixel in the image
   Loss: There is loss because floating points are 32 bits and a is converted into 9 bits. 23 bits are lost per pixel in this conversion.
   Testing Plan:
    - Implement the corresponding decompression step that converts nine bit value back into floating-point representation of a
    - Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
    - Test input images:
        - All red image
        - All blue image
        - All green image
        - All black image
        - All white image
        - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

----

9. Name: Pack a, b, c, d, Pb, and Pr into a 32-bit word.
   Description: Use the Bitpack interface we develop in part B to pack the above values into a 32-bit word.
   Input: a (as nine unsigned bit values), b, c, d (as five bit signed values), Pb, and Pr (as four bit values) of each pixel in the image
   Output: 32 bit words for each 2 x 2 block of pixels in the image
   Loss: No loss because we are using the same number of bits, just packing them into a single word

Testing Plan:

- Implement the corresponding decompression step that unpacks word into a, b, c, d, Pb, and Pr
- Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
- Test input images:
    - All red image
    - All blue image
    - All green image
    - All black image
    - All white image
    - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

----

10. Name: Write compressed binary image to standard output
    Description: Write each 32 bit code word to disk in big endian order. Code words should be written in row major order.
    Input: The compressed binary image with 32 bit code words at each block
    Output: Write to disk in big endian order with most significant byte first
    Loss: No loss because it is just outputting the compressed binary image.
    Testing Plan:
    - Implement the corresponding decompression step that reads in the binary image header and words
    - Compress and decompress and image up to this step, and use ppmdiff to ensure the original image and compressed/decompressed image are reasonably similar
    - Test input images:
        - All red image
        - All blue image
        - All green image
        - All black image
        - All white image
        - Flowers.jpg and halligan.jpg (from /comp/40/bin/images)

APPENDIX B: DECOMPRESSION STEPS DETAILED EXPLANATION

<u>Decompression</u>

Note: No loss during decompression. Each step of decompression will be tested with its corresponding compression step as written above.

1. Name: Read header (in 40image.c)
   Description: Read the header of the compressed file using fscanf
   Input: Header of compressed file
   Output: store information of height and width of compressed file

---

2. Name: Allocate new 2D array of pixels
   Description: Allocate new 2D array of pixels with height, width, denominator (of our choice), and methods attributes to store decompressed pixels
   Input: Width, height, and denominator information
   Output: new 2D array of pixels

---

3. Name: Read 32-bit code words in sequence
   Description: Read in each 32-bit code word in sequence, paying particular attention to the big-endian order of each code word
   Input: Every 32-bit code word of the image
   Output: no output

---

4. Name: Unpack 32-bit word into a, b, c, d, Pb, and Pr
   Description: Use the Bitpack interface we develop in part B to unpack each 32-bit word into local variables of a, b, c, d, Pb, and Pr
   Input: 32 bit words for each 2 x 2 block of pixels in the image
   Output: a (as nine unsigned bit values), b, c, d (as five bit signed values), Pb, and Pr (as four bit values) of each pixel in the image

---

5. Name: Convert four bit chroma codes to average Pb and Pr variables
   Description: Use the given Arith40_chroma_of_index function to convert the four bit chroma codes to average Pb and Pr values.
   Input: 4-bit quantized representations of these values
   Output: Pb and Pr float values

---

6. Name: Convert a into floating-point representation
   Description: Convert nine unsigned bit representation of a into a float
   Input: a of each 2 x 2 block of pixels as a nine unsigned bit value
   Output: a of each 2 x 2 block of pixels as a float

7. Name: Convert b, c, d into floating-point representation
   Description: Convert five-bit signed representations of b, c, d into floats
   Input: b, c, d of each 2 x 2 block of pixels as five-bit signed values
   Output: b, c, d of each 2 x 2 block of pixels as floats

---

8. Name: Compute Y1, Y2, Y3, Y4 for each pixel in the block
   Description: Compute Y1, Y2, Y3, Y4 from a, b, c, d for each pixel in the block using the inverse of the discrete cosine transform function
   Input: a, b, c, d for a 2 x 2 block of pixels
   Output: Y1, Y2, Y3, Y4

---

9. Name: Transform pixel from component video color to RGB floats
   Description: Transform each pixel with the Yi, Pb, and Pr values into RGB color floats.
   Input: Y, Pb, and Pr for each pixel
   Output: RGB color floats for each pixel

---

10. Name: Convert RGB floats into RGB integers
    Description: Convert each pixel in RGB color floating point representations into RGB integers in range 0 to denominator.
    Input: Image containing pixels with RGB floating point representations
    Output: Image containing pixels with RGB scaled integer representation

11. Name: Put RGB values of each pixel into pixel map
    Description: Place each RGB scaled integer value of each pixel into pixmap-> pixels
    Input: Image containing pixels with RGB scaled integer representation
    Output: Pixel map consisting of the RGB integers

---

12. Name: Write the uncompressed image to stdout

Description: Write the uncompressed image, stored as a Pnm_ppm, to stdout using Pnm_ppmwrite function
Input: Uncompressed Pnm_ppm image
Output: Image printed to stdout