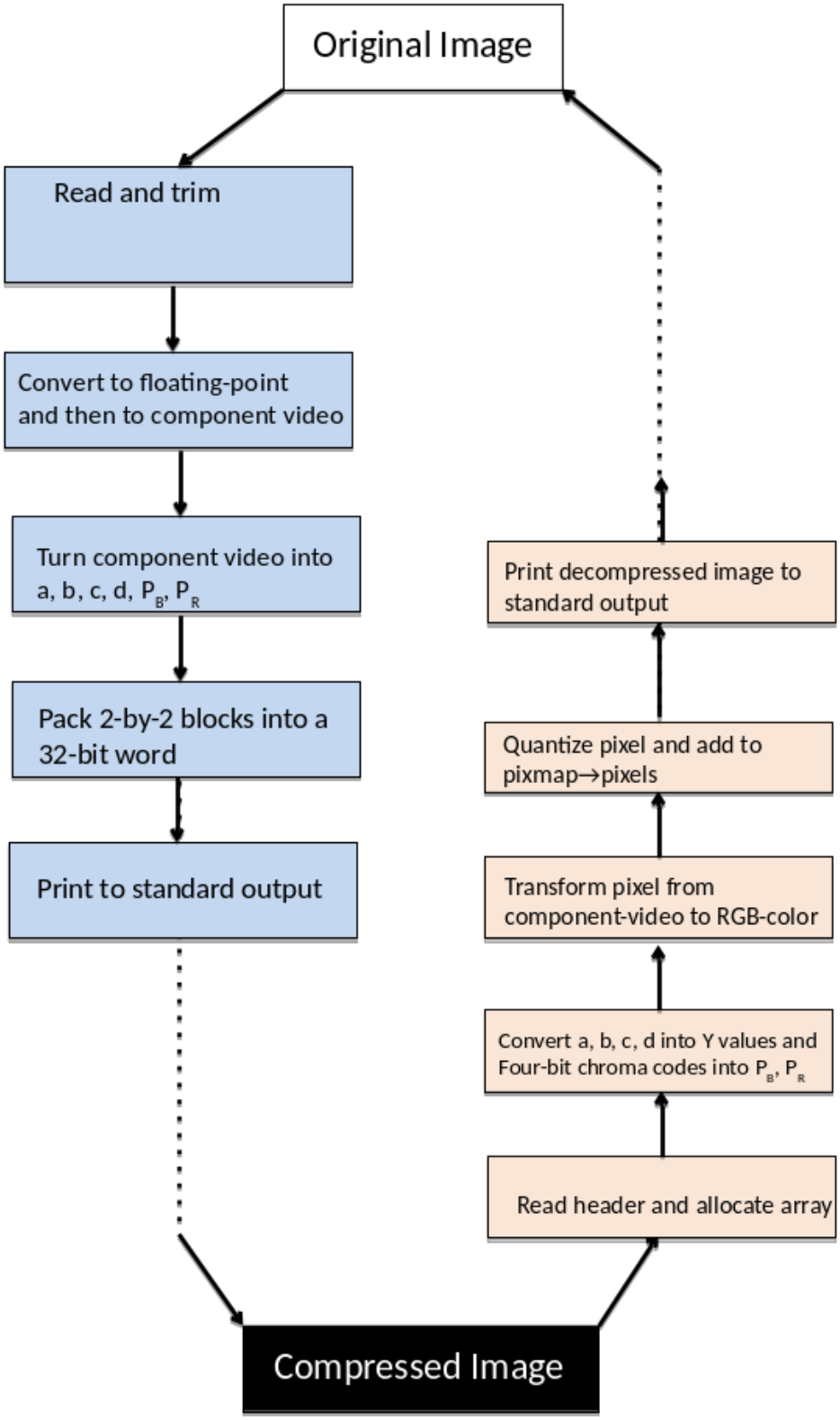
Original Image
Read and trim
Convert to floating-point and then to component video
Turn component video into a, b, c, d, P_B, P_R
Pack 2-by-2 blocks into a 32-bit word
Print to standard output
Compressed Image
Read header and allocate array
Convert a, b, c, d into Y values and Four-bit chroma codes into P_B, P_R
Transform pixel from component-video to RGB-color
Quantize pixel and add to pixmap→pixels
Print decompressed image to standard output

Compression

Step 1: Read and trim

Description

In this step, the PPM image will be read from input and trimmed to even height/width if necessary.

Input

A PPM image (filename or standard input)

Output

A Pnm_ppm struct with the dimensions and raster of the PPM file

Lost information

The height/width that was possibly trimmed away from the image as well as any pixels contained in those lines.
Otherwise, no information lost

Step 2: Convert to floating-point and then to component video

Description

Convert the pixel raster of the read PPM into floating-point representation. Those floating-point representations will then be converted into component video color space.

Input

The raster of a Pnm_ppm struct

Output

The raster of the same Pnm_ppm struct but converted into component video color space

Lost information

No information lost except small amounts if there are rounding errors in the arithmetic.

Step 3: Turn Component video into a, b, c, d, P_B, and P_R

Description

Take the average values of P_B and P_R of the four pixels in a block, then convert those average values into four-bit values. Then use the DCT to transform the four Y values of the pixels into cosine coefficients a, b, c, and d.

Input

A 2-by-2 block of pixels from the pixel raster of a Pnm_ppm struct

Output

Values a, b, c, d, index P_B and index P_R (P_B and P_R should be four-bit values)

Lost information

Information is lost because we take the average values of P_B and P_R and quantize them into four-bit values instead of just using the original whole values.

Step 4: Pack 2-by-2 blocks into a 32-bit word

Description

Convert b, c, and d to five-bit signed integers assuming their values will be between -0.3 and 0.3. Finally, use bitpack interface to pack 2-by-2 blocks from the pixel raster into a 32-bit word

Input

Values a, b, c, d, index P_B and index P_R.

Output

A 32-bit word representing a, b, c, d, index P_B and index P_R

Lost information

We lose information when doing the conversion of b, c, and d because theoretically, these values can range from +0.5 to -0.5. We are limiting the values to +0.3 to -0.3 because that is where most of the values will be. We lose information in the process.

Step 5: Print compressed binary image to standard output

Description

Prints the header of a compressed binary image and then the raster (with 2-by-2 blocks represented by 32-bit words)

Input

Collection of 32-bit words representing the image raster in video component space (a, b, c, d, index P_B, index P_R)

Output

A PPM image file outputted into standard output

Lost information

No extra information is lost in this step.

Decompression

Decompression does not lose information because the information has already been lost in compression.

Step 1: Read header and allocate array

Description

Read the header of the inputted compressed image file and allocate an array of the same dimensions as the image.

Input

A compressed PPM image (filename or standard input)

Output

A struct Pnm_ppm pixmap with the width, height, denominator, pixels array, and methods

Step 2: Convert a, b, c, d into Y_1, Y_2, Y_3, and Y_4 and convert four-bit chroma codes into P_B and P_R

Description

In this step, we will take the a, b, c, and d values from the previous step and use the inverse of the discrete cosine transformation to convert them back into the Y values of the pixels. We will also use the provided function to convert the four-bit chroma codes into P_B and P_R

Input

Variables a, b, c, d, four-bit P_B, and four-bit P_R

Output

Variables Y_1, Y_2, Y_3, Y_4, P_B and P_R

Step 3: Transform pixel from component-video color to RGB color

Description

Use the video component to rgb computations to convert from component-video color to rgb color.

Input

Y_1, Y_2, Y_3, Y_4, P_B and P_R.

Output

RGB color values in the range 0 to 1.

Step 4: Quantize pixel and add to pixmap->pixels

Description

In this step, we will multiply the pixel values by the appropriate denominator to be in the range 0 to xxx.

Input

RGB color values from the previous step

Output

RGB color values in the range 0 to denominator. Insert the values into the pixmap->pixels Uarray2

Step 5: Print decompressed image to standard output

Description

Use the Pnm_ppmwrite function to write the uncompressed image to standard output

Input

Filestream (stdout), Pnm_ppm struct (pixmap)

Output

The decompressed image file in standard output