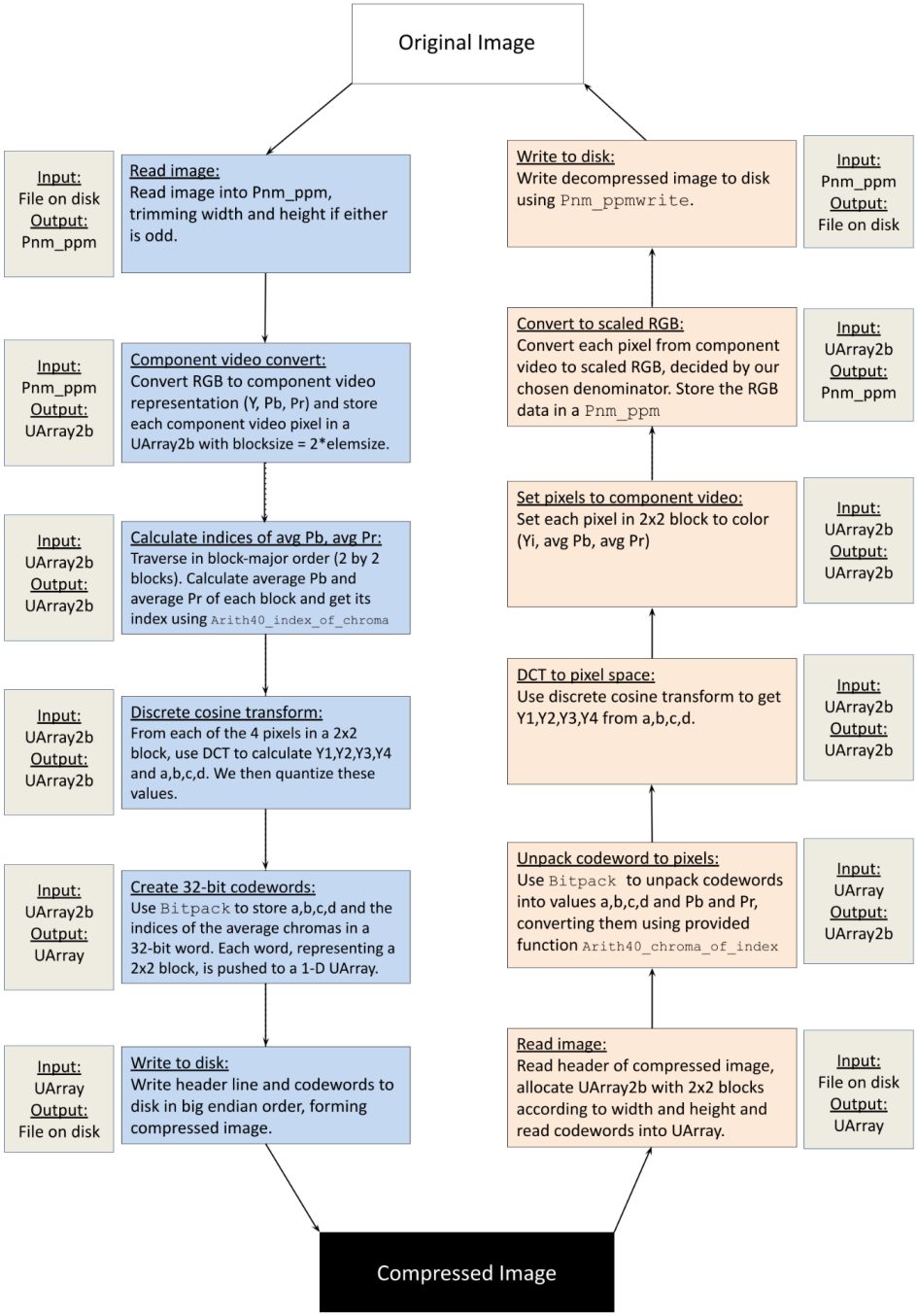
HW 4 Design

Helena Benatar, Ronit Sinha

Compression/Decompression Process



Compression

Read image

We read the image into a `Pnm_ppm`. We use plain methods for Pnm_ppm.methods. *Information is lost in this step.* If the width or height of the image is odd, then when we trim the image we lose those rows/columns.

Component video convert

Using map_row_major, we convert each pixel in the `Pnm_ppm` into component video. The Y, Pr, and Pb are stored in a component-video pixel struct. This struct is added into a UArray2b, with blocksize = 2*sizeof(component-video pixel). In other words, each block in the UArray2b is a 2x2 block of component video pixels.

Calculate indices of avg Pb, avg Pr

Traverse the UArray2b in block-major order. Within each block, calculate average Pb and Pr and get their indices using `Arith40_index_of_chroma`. *Information is lost here in this step.* since we are only storing the average Pb and Pr of a block, so during decompression each pixel in the 2x2 block will be set to these average values.

Discrete cosine transform

We use DCT to get a,b,c,d from the four Y values. To encode b,c,d with ranges of [-0.3,0.3] into the set {-15...15} by multiplying the values by 50 and rounding up. *Information is lost in this step.* Firstly, b,c,d can actually range from [-0.5,0.5], but we're clamping the range because values outside of [-0.3,0.3] are rare. Secondly, when we round up when encoding to {-15…-15}, we lose precision. For example, 0.145 and 0.15 both encode to 8.

Create 32-bit codewords

Use `Bitpack` to store the index of Pr, index of Pb, and quantized coefficients a, b, c, d in a 32-bit codeword. The codewords for each 2x2 block are stored in a Hanson UArray.

Write to disk

We write the header of the compressed image file, followed by the UArray of 32-bit codewords , writing each codeword in big-endian order.

Decompression

Read image

We use `fscanf` to get the header of the compressed image, allocate space for a Uarray2b with 2x2 blocks according to width and height.Then we start reading 32-bit chunks (codewords), storing the codewords into the Uarray2b.

Unpack codeword to pixels

Use `Bitpack` to unpack each codeword into the index of average Pr, index of average Pb, a, b, c, d. We can then get the quantized average Pr and Pb using `Arith40_chroma_of_index`.

DCT to pixel space

Use the inverse of the discrete cosine transform to compute Y1,Y2,Y3,Y4 from a,b,c,d.

Set pixels to component video

Set each pixel in 2x2 block to color (Yi, avg Pb, avg Pr). This is where the information loss from step 3 of compression takes effect, since we are setting each of the 4 pixels the block to the average of their chromas.

Convert to scaled RGB

From the component video UArray2b we just made, traverse it and convert each pixel to scaled RGB based on our chosen denominator, and then set the RGB pixel to the corresponding element in another UArray2. This RGB UArray2 will be used to create a `Pnm_ppm`.

Write to disk

We use `Pnm_ppmwrite` to store the `Pnm_ppm` we just created as a file.

Implementation Plan

With our design, each compression step is an inverse of the decompression step to the right of it in the diagram above. So, in order to test a compression step, we can just run the output of the step into the decompression step to see if it matches the original input, and vice versa (i.et. send decompression step output to compression step).

In order of implementation:

Read image (compression):

- *Information is lost in this step*. If the width or height of the image is odd, then when we trim the image we lose those rows/columns.
- Test by opening PPMs of different sizes, from /comp/40/bin/images folder (converting jpegs to ppm using djpeg) and their ppmtrans transformations, and files that are not PPMs
- Send output to “write to disk” decompression step to make sure image was properly read

Write to disk (decompression):

- Test by taking input from “read image” compression step to make sure file is properly written, use ppmdiff to make sure

Component video convert (compression):

- Test on images that are all one color (i.e. completely white) and images from /comp/40/bin/images folder (converting jpegs to ppm using djpeg) and their ppmtrans transformations
- Send output to convert scaled RGB decompression step to make sure colors were properly converted

Convert to scaled RGB (decompression):

- Test by taking input from “component video convert” compression step to make sure color is properly converted, then write this to disk and use ppmdiff to compare to input image.
- We should also test on images that are all one color (i.e. a big green square) to see how changes in denominator affect quantization artifacts.

Calculate indices of avg Pb, avg Pr (compression):

- Test on images from /comp/40/bin/images folder and their ppmtrans transformations
- Send output to set pixels in component video decompression step to make sure the averages per block were correctly implemented

Set pixels in component video (decompression):

- Test by taking input from “calculate indices of avg Pb, avg Pr” compression step to make sure pixels are set to their block average chroma. Write the output image to disk and compare it using ppmdiff.

Discrete cosine transform (compression):

- Test on images from /comp/40/bin/images folder and their ppmtrans transformations
- Send output to DCT to pixel space decompression step to make sure the transformation was computed correctly.

DCT to pixel space (decompression):

- Test by taking input from “discrete cosine transform” to make sure the transformation to pixels is properly implemented. Then write this to disk and use ppmdiff to compare to input image.

Create 32-bit codewords (compression):

- Test on images from /comp/40/bin/images folder and their ppmtrans transformations
- Send output to unpack codewords to pixels decompression step to make sure the codewords were created correctly

Unpack codeword to pixels (decompression):

- Test by taking input from the “create 32-bit codewords” compression step to make sure codewords are correctly unpacked. Then write this to disk and use ppmdiff to compare to the input image.

Write to disk (compression):

- Test by taking input from “read image” decompression step to make sure file is properly written. Then use ppmdiff to compare the input image.

Read image (decompression):

- Test by opening PPMs of different sizes, from /comp/40/bin/images folder (converting jpegs to ppm using djpeg) and their ppmtrans transformations, and files that are not PPMs
- Send output to “write to disk” compression step to make sure the image was properly read.