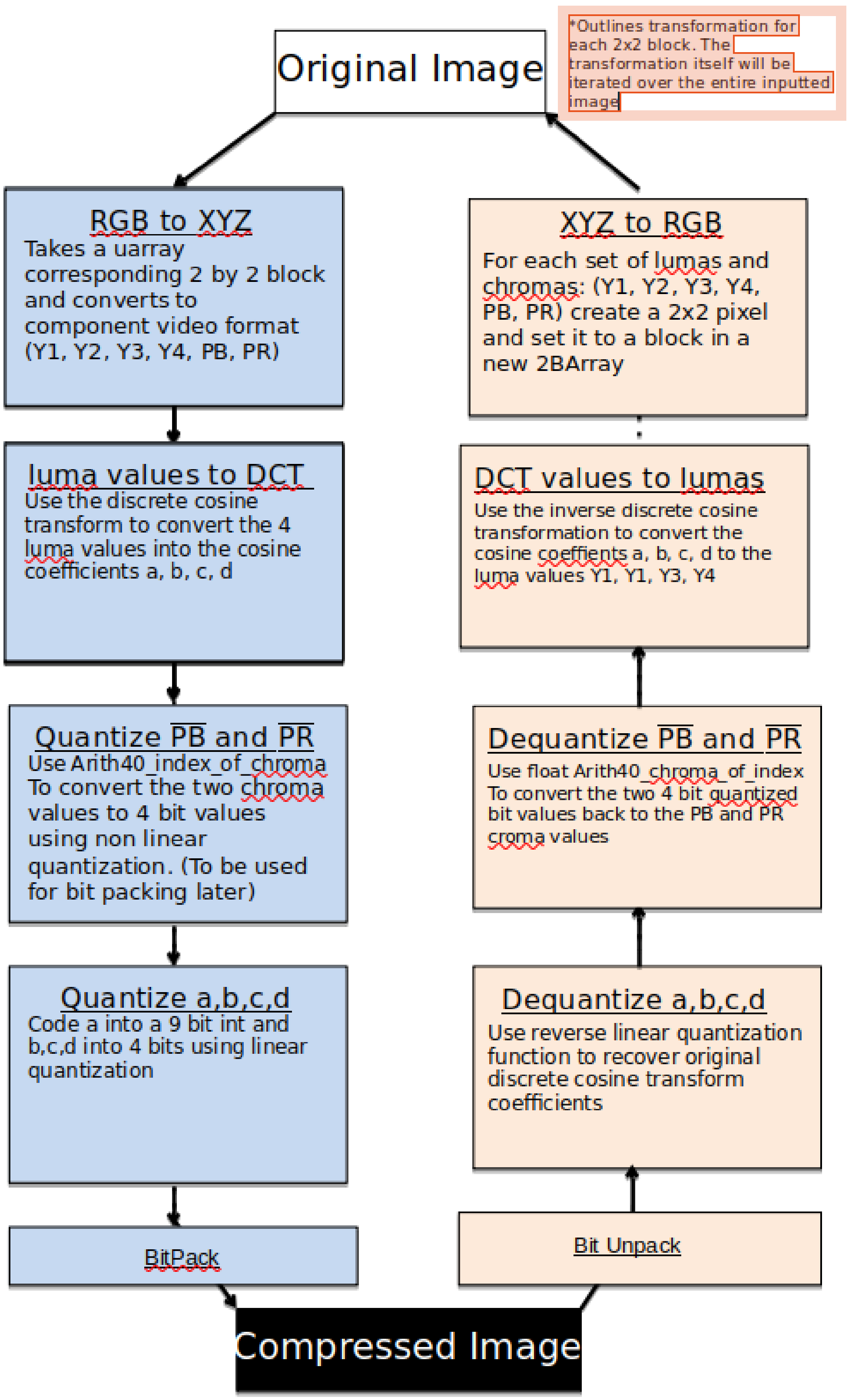
Jake Kerrigan (jkerri01) and Jacob Frieman (jfriem01) - arith design



Original Image
*Outlines transformation for each 2x2 block. The transformation itself will be iterated over the entire inputted image
RGB to XYZ
Takes a uarray corresponding 2 by 2 block and converts to component video format (Y1, Y2, Y3, Y4, PB, PR)
luma values to DCT
Use the discrete cosine transform to convert the 4 luma values into the cosine coefficients a, b, c, d
Quantize PB and PR
Use Arith40_index_of_chroma To convert the two chroma values to 4 bit values using non linear quantization. (To be used for bit packing later)
Quantize a,b,c,d
Code a into a 9 bit int and b,c,d into 4 bits using linear quantization
BitPack
XYZ to RGB
For each set of lumas and chromas: (Y1, Y2, Y3, Y4, PB, PR) create a 2x2 pixel and set it to a block in a new 2BArray
DCT values to lumas
Use the inverse discrete cosine transformation to convert the cosine coeffients a, b, c, d to the luma values Y1, Y1, Y3, Y4
Dequantize PB and PR
Use float Arith40_chroma_of_index To convert the two 4 bit quantized bit values back to the PB and PR croma values
Dequantize a,b,c,d
Use reverse linear quantization function to recover original discrete cosine transform coefficients
Bit Unpack
Compressed Image

Compression Design

Structs:

1. Pnm_rgb - (unsigned int r, g, b)
2. CVFFields - (unsinged ints Y1, Y2, Y3, Y4, PB, PR)
3. DCTFields - (float a, float b, float c, float d)
4. WordFields - (signed int a, unsigned b, c, d, unsigned int b, IndexPB, IndexPR)

Data Structures:

UArray2b (blocksize 2) of Pnm_rgb inputImage

- Feeds each block into the RGB to XYZ
- Not mentioned in any of the functions as they are designed so that a single block of pnm_rgbs can be passed in and converted to a single integer

UArray2 of ints compressedWords

UArray2b (blocksize 2) of Pnm_rgb outputImage

Compression Steps:

<u>RGB to XYZ</u>

Input: Pnm_rgb representing a block from the inputImage

Output: CVFFields containing the conversion from the inputted RGB values

CVFFields componentVideoFormat(uarray block)

- Uses the provided calculations in spec to complete the conversion

<u>Luma Values to DCT:</u>

Input: CVFFields although we only need Y1, Y2, Y3, and Y4

Output: DCTFields struct

DCTFields discreteCosineTransfer(CVFFields input)

- Uses the provided calculation in spec to complete the transformation

<u>Quantize PB and PR:</u>

Input: PB and PR luma values (unsigned ints) retrieved through CVFFields

Output: Void, but passed through WordFields (IndexPB, IndexPR)

Void quantizeChromas(CVFFields input, WordFields output)

- Uses the arith40_index_to_croma

- When quantizing we lose a degree of specificity with the PB/PR values because they are being converted to a lower size. Because this function is given to us we can't say how much exactly however

<u>Quantize a, b, c, d:</u>
Input: Luma values a, b, c, d (unsigned ints)
Output: Void, but passed through WordFields
Void quantizeLums(DCTFields input, WordFields output)

- Uses a linear quantization to convert the cosine coefficients that we will implement ourselves by scaling up a, b, c, d
- Again when quantizing we lose a degree of specificity for converting to a lower size, for example if a, b, c, d or are over -.5/.5 they will be dequantized at .3/-.3

<u>Bitpack:</u>
Input: WordField struct
Output: 32 bit Integer
int bitpack(WordFields input)

- Uses our implementation of the bitpack.h interface
- First checks if the bit will fit in a signed or unsigned integer
- Then extracts the values from the WordFields struct and packs them into the designated integer

Decompression Design

Structs:

Same as compression structs

Data Structures:

UArray2b (blocksize 2) of Pnm_rgb outputImage

Decompression Steps

<u>BitUnpack:</u>
Input: 32 bit Integer
Output: WordField struct
Notes: Allocates a WorldField Struct
WordField bitpack(int input)

- Uses Bitpack_get to retrieve the elements and assign them to WordField

<u>Dequantize a, b, c, d:</u>

Input: quantized a, b, c, d passed through WordFields
Output: Luma values a, b, c, d (unsigned ints)
Void quantizeLums(DCTFields input, WordFields output)
-Uses inverse of our linear quantization to convert the cosine coefficients back to floats

Dequantize PB and PR
Input: Quantized PB and PR (retrieved from wordField)
Output: Dequenatied PB and PR values
DCT values to luma
Input: DCTField
Output: void but returned through CVFFields by passing by reference and setting fields Y1, Y2, Y3, and Y4
CVFFields retransformToCVF(DCTFields input)

XYZ to RGB:
Input: CVFFields containing the component video format of the block
Output: Uarray representing a block of RGB values
uarray RGBBlockCreateor(CVFFields input)

Implementation Plan

Testing Disclamer: Test with handwritten inputs and outputs / small images. Each step is tested with its inverse to ensure they both work as expected.

1. RGB to XYZ
2. XYZ to RGB
    - Testing:
    - 1a) Test with one block image, feed into XYZ to RGB
      1b) feed the CVFField output into RGB to XYZ and diff with the original output
3. Luma values to DCT
4. DCT to Luma values
    - Testing: After confirming the previous step works:
      2a) Perform 1a then feed the resulting CVFField into Luma Values to DCT
      2b) Use DCT to Luma values and XYZ to RGB then perform 1b
5. Quantize PB and PR
6. Dequantize PB and PR
    - Testing: After confirming the previous step works:
      3a) Perform 1a then feed the resulting CVFField into Quantize PB and PR
      3b) Use Dequantize PB and PR then perform 1b
7. Quantize a, b, c, d
8. Dequantize a, b, c, d
    - Testing: After confirming the previous steps work:

- 4a) Perform 2a then feed the resulting DCTField into Quantize a, b, c, d
- 4b) Feed the resuling WordField into dequantize a,b,c,d then perform 2b

9. Bitpack
10. BitUnpack
    - Testing: After confirming the previous steps work:
      5a) Perform 4a and 3a then fed the resulting WordField into Bitpack

      5b) Use BitUnpack then perform 4b and 3b.