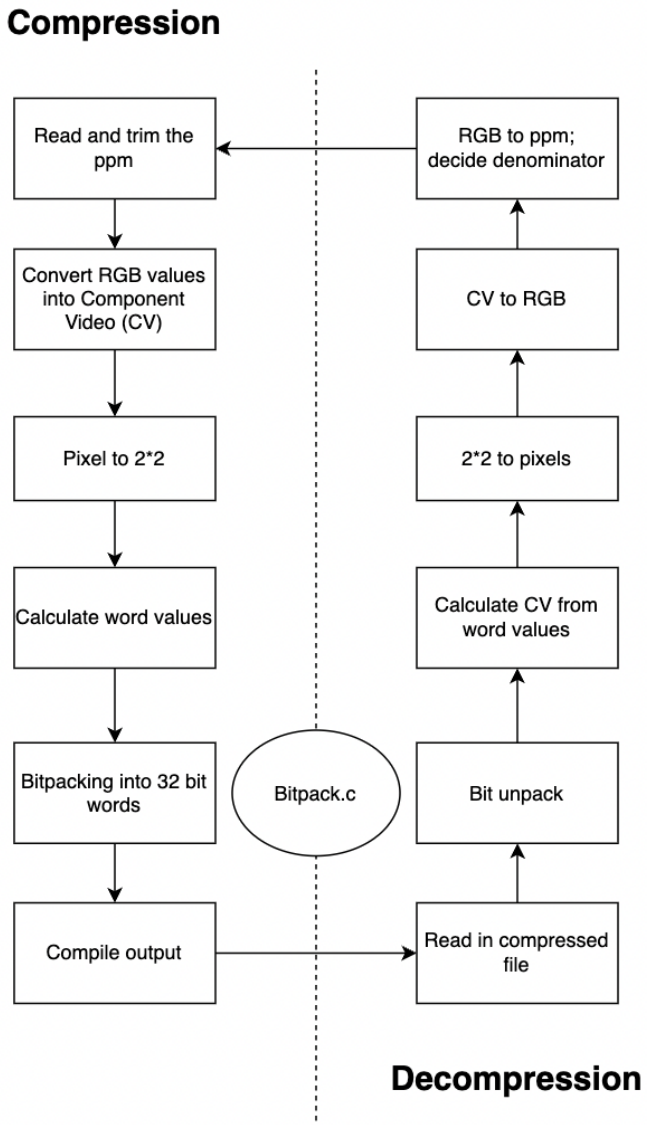
## ***HW4 - Arith Design Doc***

Sam Hu (khu04) Josh Bernstein (jberns02)




# Implementations

**Structs:**

Cv_pixel is a struct which will be used to store the values of component video, that being Y, Pb, and Pr. There will be an instance of each struct per each pixel, allowing us to keep track of each pixel's CV values.

```
Struct Cv_pixel {
        Float Y_value;
        Float Pb_value;
        Float Pr_value;
```

};

wordValues is a struct which will be used to store the values a,b,c,d and the average Pb and Pr index. This allows us to easily access specific values of a 32-bit packed word.

```
Struct wordValues{
        Float a_value;
        Float b_value;
        Float c_value;
        Float d_value;
        Float avg_Pb;
        Float avg_Pr;
};
```

**Note:** unless otherwise specified, "(de)compression step" in each step refers to the very step on its left/right, which does the reversed action.

---------END_OF_NOTE----------

The implementation order will be as per the reading order of the following table, where a pair of compression-decompression step will be implemented and tested together as in nearly all the cases the input and output can be fed into each other as cross-verification:

| Compression | Decompression (reverse ordered) |
|---|---|
| **Read in from ppm:** Using a2methods, read in from the given file and update dimensions if necessary. (30min)<br>**Tests:**<br>- Print results from file read in using the decompression step, should pass diff test.<br>- Make sure both stdin and command line input are accepted<br>- Make sure to exit failure/CRE for too many files or an invalid ppm file<br>- Exit early with a 0*0 header if either width or height is less than 2 and not negative<br>- No Valgrind errors / leaks<br>**Input :** ppm files from stdin or command line arguments<br>**Output:** a trimmed Pnm_ppm object<br>**Loss of information**: Some information might be lost due to trimming, information is otherwise kept. | **RGB to ppm:** Using the pixel map, create a Pnm_ppm object that can be written to output using pnm.h. (30min)<br><br>**Tests:**<br>- All regular prints should not make pnm.h to raise errors<br>- Diff test should pass, i.e., the pixels are inserted to the right place<br>- No Valgrind errors / leaks<br><br>**Input:** the pixel map from previous step<br>**Output:** a Pnm_ppm object ready for output (and printed to stdout)<br>**Loss of information:** no information should be lost at this point. |
| **Convert RGB values to CV:** Calculate CV values using RGB from the pnm object and given formulas (eg, y = 0.299 * r + 0.587 * g + 0.114 * b;), store them as 3 floats in a struct. (30min)<br>**Tests:** | **CV to RGB:** Using the formulas provided (eg., r = 1.0 * y + 0.0 * pb + 1.402 * pr;)(30min)<br>**Tests:**<br>- Create our own pixels of black, white, and random colors and compare its results with the function's results, should be about the same value |

| | |
|---|---|
| - Create our own pixels of black, white, and random colors and compare its results with the function's results, should be about the same value<br>- Results from the decompression step can be converted into its original values (with some loss of precision)<br>- No Valgrind errors / leaks<br>**Input:** for each conversion, a Pnm_rgb struct<br>**Output:** a CV struct with (Y, Pb, Pr) as floating points<br>**Loss of information:** Some information might be lost as we try to represent discrete RGB values as floating points | - Results from the compression step can be converted into its original values (with some loss of precision)<br>- No Valgrind errors / leaks<br><br>**Input:** the pixel map from previous step<br>**Output:** a Pnm_ppm object ready for output (and printed to stdout)<br>**Loss of information:** some information may be lost as float is converted into discrete values. |
| **Pack 2 by 2:** Pack the converted pixels into a sequence of 2*2 blocks. (20min)<br>**Tests:**<br>- Print out packed and original values and check if both are in the same order; (For smaller valuse)<br>- Cross-test with decompression step outputs, should be able to "loop through"<br>- No Valgrind errors / leaks<br>**Input**: Array/2D array of pixels (converted to CV)<br>**Output**: array of structs, each containing 4 CV pixel structs<br>**Loss of information**: no information is loss at this step since it's just a restructure of what we have. | **Unpack 2*2:** Unpack the 2*2 pixel blocks into a single UArray2 of CV pixels. (40min)<br><br>**Tests:**<br>- Print out packed and original values and check if both are in the same order; (For smaller valuse)<br>- Cross-test with compression step outputs, should be able to "loop through"<br>- No Valgrind errors / leaks<br>**Input:** an array of structs of four CV pixel values<br>**Output:** a UArray2 of CV pixels<br>**Loss of information:** no information is loss at this step since it's just a restructure of what we have |
| **Calculate word values:** Convert each struct from the previous step into a struct of individual values of the 32 bit word, using the provided formulas (DCT & index).(15min)<br><br>**Tests:**<br>- Test with small sample inputs of black, white, and colored pixels and check with calculator, should be the same ignoring loss of precision<br>- Compare values from decompression step's output and its input<br>- No valgrind errors / leaks<br>**Input:** struct from the previous step<br>**Output:** a new struct with 1 variable for each value of the 32 bit word<br>**Loss of information**: some negligible loss due to the loss of precision when calculating abcd; Pb Pr have some loss | **Calculate CV:** Using the values in the word (passed in as a parameter), compute the 4 pixels with the provided formulas and functions (15min)<br><br>**Tests:**<br>- Test with small sample inputs of black, white, and colored pixels and check with calculator, should be the same ignoring loss of precision<br>- Compare values from decompression step's output and its input<br>- No valgrind errors / leaks<br>**Input:** a struct of word values unpacked<br>**Output:** an array of structs of four CV pixel values<br>**Loss of information:** some loss of precision may occur due to float calculations, but is unlikely as all values were already compressed. |

| | |
|---|---|
| of information after averaging and most likely have some loss being converted into index. | |
| **<u>Bit Packing:</u>** For each struct, pack the values into an actual 32 bit word using bitpack.c.(20min + ~1.5hr for bitpack.c)<br><br>Specifically, values will be trimmed down and verified by the fit functions, and inserted into the word using the new functions.<br>**Tests: (We expect the usage of bitpack.c to be straightforward and will perform some test cases for bitpack.c itself)**<br>For compress40<br>- Use a manually created word to loop around pack/unpack, should have the same results<br>For bitpack<br>- Expect the values for signed unsigned fit to be the same as externally calculated results<br>- Expects the new function to change the desired bits<br>- And not alter other bits.<br>Also :No Valgrind errors / leaks<br>**Input**: struct from the previous step<br>**Output**: an array of words (unsigned ints?)<br>**Loss of information**: Some information might be lost as values are represented by less bits. | **<u>Bit unpack:</u>** Unpack the word into a struct where each value occupies a single value with functions from bitpack.c. (20min)<br><br>Specifically, values will be retrieved by using get functions, and calculated.<br>**Tests: (We expect the usage of bitpack.c to be straightforward and will perform some test cases for bitpack.c itself)**<br>For compress40<br>- Use a manually created word to loop around pack/unpack, should have the same results<br>For bitpack<br>- Expect get to retrieve the right values<br>- And not alter the word in this process<br>Also :No Valgrind errors / leaks<br>**Input:** 32-bit words<br>**Output:** struct containing individual values for a,b,c,d, and Pb Pr indices<br>**Loss of information:** Information is already loss during packing so no information is lost |
| **<u>Convert output:</u>** Using all the results (listed in input), create header, and print to stdout (10min)<br>**Tests:**<br>- Feed results to the decompression step, should be able to be interpreted and decompress into a relatively same image<br>- No Valgrind errors / leaks<br>**Input**: height and width, the word sequence<br>**Output**: compressed file to stdout<br>**Loss of information**: No information is lost at this point | **<u>Read input:</u>** Read inputs compressed file and record dimensions and store the word sequence. (20min)<br>**Tests:**<br>- Expects the word sequence to be valid (have exact numbers stated in w and h), CRE if not<br>- Prints statement to stderr and exit(0)? If w or h is 0<br>- Exit (1) if w or h is odd<br>- Handles the result properly (being able to feed into the compression step) if valid.<br>- No Valgrind errors / leaks<br>**Input**: struct from the previous step<br>**Output**: an array of words (unsigned ints?)<br>**Loss of information**: No information is lost at this point |