PPM Image Transformations

CS-350: Systems Programming

Instructor: Dr. Dorian Arnold
Computer Science Department, Emory University

PPMCVT (ppm convert)

`ppmcvt [bg:i:r:smt:n:o]`

manipulates input Portable Pixel Map (PPM) files and outputs a new image based on its given options.

The options are:

-b
convert input file to a Portable Bitmap (PBM) file. (DEFAULT)

-g:
convert input file to a Portable Gray Map (PGM) file using the specified max grayscale pixel value [1-65535].

-i:
isolate the specified RGB channel. Valid channels are “red”, “green”, or “blue”.

-r:
remove the specified RGB channel. Valid channels are “red”, “green”, or “blue”.

-s
apply a sepia transformation

-m
vertically mirror the first half of the image to the second half

-t:
reduce the input image to a thumbnail based on the given scaling factor [1-8].

-n:
tile thumbnails of the input image based on the given scaling factor [1-8].

-o:
write output image to the specified file. Existent output files will be overwritten.

Examples

`ppmcvt -o out.pbm in.ppm`
convert the PPM image in in.ppm to a PBM image in out.pbm

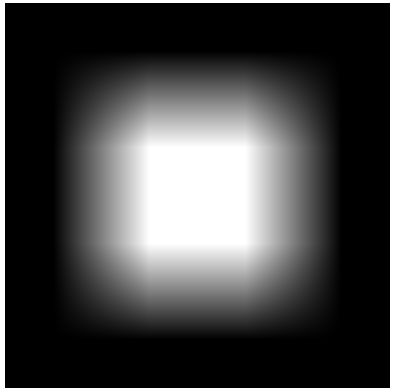
`ppmcvt -g 16 -o out.pgm in.ppm`
convert the PPM image in.ppm to a PGM image in out.pgm

`ppmcvt -s -o out.ppm in.ppm`
apply a sepia transformation to the PPM image in in.ppm and output the new image to out.ppm

`ppmcvt -n 4 -o out.ppm in.ppm`
tile 4 1:4-scaled (quarter-sized) thumbnails of the image in in.ppm into a new PPPM image in out.ppm.

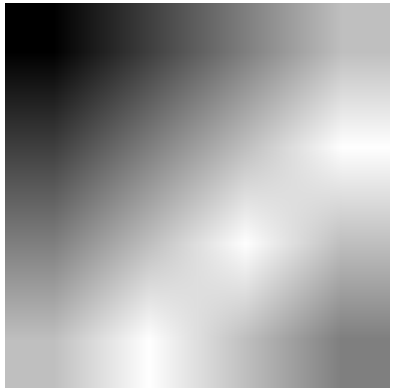
PBM, PGM and PPM Files

```
P1
4 4
0 0 0 0
0 1 1 0
0 1 1 0
0 0 0 0
```



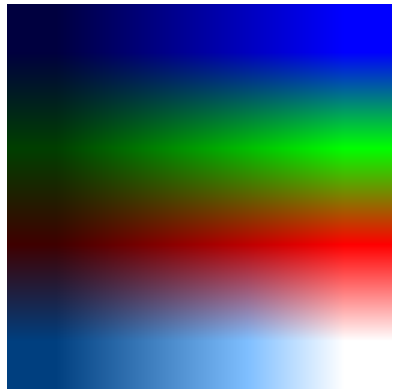
square.pbm

```
P2
4 4
4
0 1 2 3
1 2 3 4
2 3 4 3
3 4 3 2
```



square.pgm

```
P3
4 4
4
0 0 1  0 0 2  0 0 3  0 0 4
0 1 0  0 2 0  0 3 0  0 4 0
1 0 0  2 0 0  3 0 0  4 0 0
0 1 2  1 2 3  2 3 4  4 4 4
```



square.ppm

Transformations

- **Bitmap:**
  $Average(R + G + B) < PPMMax/2$
- **Grayscale**:
  $Average(R + G + B) \times PGMMax\ PPMMax$
- **Sepia**:
  $NewR = 0.393(OldR) + 0.769(OldG) + 0.189x(OldB)$
  $NewG = 0.349(OldR) + 0.686(OldG) + 0.168x(OldB)$
  $NewR = 0.272(OldR) + 0.534(OldG) + 0.131x(OldB)$
- **Mirror**:
  Vertically reflect the left half of the image onto the right half.
- **Thumbnail**:
  Shrink image by scaling factor
- **Nup**:
  Tile thumbnail across entire image

What I Did*

1. Defined “Options” struct: {mode, arg, infile-name, outfile-name}
2. Implemented function to process command line (returns “Options”)
   - command line error checking done in this function
3. Implemented image allocation/deallocation routines
4. Implemented a function for each mode (transformation)
   - read input file
   - create output struct
   - for each input pixel, update respective output pixel based on mode
   - write output file
   - destroy all image structs and any other allocated memory
5. Called appropriate transformation function from main()

*Not necessarily what you must or should do

PBM Library (pbm.h/pbm.c)

structs for PBM, PGM and PPM image types

```
typedef struct {
  unsigned int ** pixmap[3];
  unsigned int height, width, pixmax;
} PPMImage;
```

```
typedef struct {
  unsigned int ** pixmap;
  unsigned int height, width, pixmax;
} PGMImage;
```

```
typedef struct {
  unsigned int ** pixmap;
  unsigned int height, width;
} PBMImage;
```

I/O routines to read/write images from/to a PBM, PGM or PPM file.

```
PPMImage * read_ppmfile( const char * filename );
void write_pbmfile( PBMImage *image, const char * filename );
void write_pgmfile( PGMImage *image, const char * filename );
void write_ppmfile( PPMImage *image, const char * filename );
```

Declares memory allocation/deallocation routines for image structs. YOU MUST IMPLEMENT!

```
PPMImage * new_ppmimage( unsigned int width, unsigned int height, unsigned int max);
PGMImage * new_pgmimage( unsigned int width, unsigned int height, unsigned int max);
PBMImage * new_pbmimage( unsigned int width, unsigned int height );

void del_ppmimage( PPMImage * );
void del_pgmimage( PGMImage * );
void del_pbmimage( PBMImage * );
```

new_ppmimage() is called by read_ppmfile()

```
typedef struct {
  unsigned int ** pixmap[3];
  unsigned int height, width, pixmax;
} PPMImage;
```

`pixmap`: Three `height` x `width`, 2-dimensional pixel arrays, for 'R', 'G', 'B' values

`height`: image height (number of rows)

`width`: image width (number of columns)

`pixmax`: maximum pixel value of image

PGM and PBM structs and routines work analogously

```
typedef struct {
  unsigned int ** pixmap[3];
  unsigned int height, width, pixmax;
} PPMImage;
```

```
//read image from mypic.ppm: read_ppmfile() calls new_ppmimage()
PPMImage * p = read_ppmfile( "mypic.ppm" );
```

//p->pixmap[0]: 'R' pixmap array

//p->pixmap[1][7]: 8th row of pixels of 'G' pixmap array

//p->pixmap[2][4][10]: 11th pixel in 5th row of 'B' pixmap array

```
//write image to mypic-copy.ppm
write_ppmfile( "mypic-copy.ppm" );
```

```
//deallocate all memory associated with p
del_ppmimage( p );
```

PGM and PBM structs and routines work analogously

```
PBMImage * new_pbmimage( unsigned int width, unsigned int height );
```

1. Define PBMImage pointer
2. Allocate storage for PBMIMage struct for PBMImage pointer
3. Initialize PBMImage struct height and width
4. Initialize PBMImage struct pixmap (by allocating the required storage*):
    1. pixmap should point to an array of pointers, one pointer for each row of pixmap
    2. Each pixmap row pointer should point to an array of unsigned integers, one unsigned integer for each column
5. return pointer to PBMImage struct

```
typedef struct {
  unsigned int ** pixmap;
  unsigned int height, width;
} PBMImage;
```

*Analogous to the 2D example in the malloc lecture

Other Hints and Tips

- Keep it simple! Implement easiest transformations first.
  - Consider “null” transformation as first test: read image; copy obj; write new obj
- Use small .ppm files you can inspect manually for initial testing
- Correct deallocation of pixmap array will reverse allocation order
- You may need special consideration for odd numbers of rows/columns
- Use `strtol()` to convert strings to numbers
- Use `strcmp()` to compare 2 strings
- For thumbnail/Nup, # rows/# cols may not be multiple of scaling factor
- For many transformations, it is possible to update the input image struct in place – without a separate output image struct*.
- A memory debugger, e.g. valgrind, is recommended
- UNIX `diff` program identifies differences (if any) between two files

*I did not do this