EECS 22: Assignment 5

Prepared by: Weiwei Chen, Prof. Rainer Doemer

November 15, 2012

Due on Monday 12/03/2012 11:59pm. Note: this is a two-week assignment.

1 MovieLab [100 points + 10 bonus points]

In this assignment you will learn how to design a program to take command-line arguments and how to design a linked list.

The program, *MovieLab* will be developed to perform digital image processing (DIP) operations on a input movie stream.

A movie is basically a sequence of images with the same size. You will be asked to design a linked list for images to represent a movie in your program, generate the frames of the movie, and then use the DIP operation functions designed in the previous assignments to perform DIP operations onto the images in the movie.

1.1 Introduction

A movie is basically a sequence of images with different contents but same fixed size. Play a movie is actually to show the images one after another at a certain rate, i.e. *fps* (frames per second). Each image in the movie is the same as what we've learned in the previous assignments. It is essentially a two-dimensional matrix, which can be represented in C by an array of pixels. A pixel is still the smallest unit of an image.

In this assignment, you will work on a movie with fixed number of frames (141) and resolution (352×288 pixels/frame). The color space of the images in the movie is **YUV** (`http://en.wikipedia.org/wiki/YUV`) instead of **RGB**.

However, the color of each pixel is still represented by 3 components, now *Y channel*, *U channel* and *V channel*. Here, *Y channel* represents the luminance of the color, while *U channel* and *V channel* represent the chrominance of the color. Each channel for one pixel is still represented by an intensity value between 0 and 255. In order to utilize the DIP functions that handles the images using the **RGB** color space, conversion is needed to change the **YUV** 3-tuple into **RGB** tuple for each pixel (Section 1.3.3). The **YUV** color space is very common for video streams. Our input and output file will both use the **YUV** color space.

1.2 Initial Setup

Before you start working on this assignment, please do the following steps:

1. Create the subdirectory *hw5* for this assignment, and change your current directory to *hw5*.
2. We will reuse some of the source code files from our previous assignments. Please feel free to reuse any of your designs, or reuse the solution files to the previous assignments which are posted on our course website. The files we will reuse are *DIPs.c* and *DIPs.h*. Copy these two files to *hw5*.
3. Copy the provided files from the *eecs22* account on the *ladera* server.

```
cp ~eecs22/hw5/Image.c ./
cp ~eecs22/hw5/Image.h ./
cp ~eecs22/hw5/MovieLab.c ./
```

Here,

- **Image.h** is the header file for the definition of the structure and declarations of the pixel mapping functions we've been using for assignment4. The name of the structure members are changed for representing images using different color spaces (**RGB** and **YUV**);
- **Image.c** is the modified source code file for *Image.h* (will be available after the deadline of assignment4);
- **MovieLab.c** is the template file with sample code for command-line argument parsing, and the basic file I/O functions.

4. Create a symbol link to the input movie stream file from the *eecs22* account on the *ladera* server.

```
ln -s ~eecs22/hw5/bird.yuv
```

bird.yuv here is symbolic link to the input movie stream in our *eecs22* account. Since we have space limitation for each account on *ladera.eecs.uci.edu*, it is helpful to save disk space for each account by sharing the read-only input file.

We will use the *bird.yuv* file as the test input stream for this assignment. Once a movie image operation is done, you can save the output image as *name.yuv* in your working directory by using the *"-o"* option.

You will need a *YUV* player to view the movie files. The *YUV* player requires you to have X window support on your own machine where you use either **PuTTY** (Windows user) or **Terminal** (Mac User) to remote login the Linux server. For Mac user, your system has the X window support installed. Please remember to add the *"-X"* option while using the *"ssh"* command. For windows users, you need to install the X server first and set the configurations of **PuTTY** with proper *X11 forwarding*. A free X server, **Xming**, for Windows system is available from `http://sourceforge.net/projects/xming/`. The detailed instructions on **PuTTY** configuration is available from `https://eee.uci.edu/toolbox/messageboard/m11480/f31314/t361907/p643529/`.

With the X server running properly with your remote login software, you can use the following command to play your movie files (.yuv):

```
cd hw5
~eecs22/bin/yay -s widthxheight -f 1 yourfilename.yuv
```

Specifically, you can play the input movie stream by using:

```
~eecs22/bin/yay -s 352x288 -f 1 bird.yuv
```

1.3 Design the MovieLab Program

In this assignment, we will design the movie representation in a C program.

1.3.1 The Image.c module (provided)

In assignment4, we designed the *Image.c* module for the basic functions of an image. A struct *IMAGE* is defined for the pixels in the **RGB** format. Image creation and deletion functions and basic pixel color getting and setting functions are defined accordingly.

Since the data structure for the **YUV** format is almost the same as the **RGB** format. We will reuse the *Image.c* module from assignment4. In order to represent the different color representations better, we rename the pointer member variables for *IMAGE*. Now the structure *IMAGE* and the function signatures look like:

```
typedef struct {
unsigned int Width; /* image width */
unsigned int Height;  /* image height */
unsigned char *R_Y; /* pointer to the memory storing all the R or Y intensity values */
unsigned char *G_U; /* pointer to the memory storing all the G or U intensity values */
unsigned char *B_V; /* pointer to the memory storing all the B or V intensity values */
}IMAGE;

/*Get the color intensity of the Red channel of pixel (x, y) in image */
unsigned char GetRPixel(IMAGE *image, unsigned int x,  unsigned int y);

/*Get the color intensity of the Green channel of pixel (x, y) in image */
unsigned char GetGPixel(IMAGE *image, unsigned int x,  unsigned int y);

/*Get the color intensity of the Blue channel of pixel (x, y) in image */
unsigned char GetBPixel(IMAGE *image, unsigned int x,  unsigned int y);

/*Set the color intensity of the Red channel of pixel (x, y) in image with value r */
void SetRPixel(IMAGE *image, unsigned int x,  unsigned int y, unsigned char r);

/*Set the color intensity of the Green channel of pixel (x, y) in image with value g */
void SetGPixel(IMAGE *image, unsigned int x,  unsigned int y, unsigned char g);

/*Set the color intensity of the Blue channel of pixel (x, y) in image with value b */
void SetBPixel(IMAGE *image, unsigned int x,  unsigned int y, unsigned char b);

/* allocate the memory spaces for the image              */
/* and the memory spaces for the color intensity values. */
/* return the pointer to the image                       */
IMAGE *CreateImage(unsigned int Width, unsigned int Height);

/*release the memory spaces for the pixel color intensity values */
/*release the memory spaces for the image                        */
void DeleteImage(IMAGE *image);

/*Get the color intensity of the Y channel of pixel (x, y) in image */
#define GetYPixel GetRPixel

/*Get the color intensity of the U channel of pixel (x, y) in image */
#define GetUPixel GetGPixel

/*Get the color intensity of the V channel of pixel (x, y) in image */
#define GetVPixel GetBPixel

/*Set the color intensity of the Y channel of pixel (x, y) in image */
#define SetYPixel SetRPixel

/*Set the color intensity of the U channel of pixel (x, y) in image */
#define SetUPixel SetGPixel

/*Set the color intensity of the V channel of pixel (x, y) in image */
#define SetVPixel SetBPixel
```

1.3.2 The ImageList.c module

We are going to design a double-linked list to store the frames (images) for a movie and keep them in the correct order.

As discussed in **lecture 14**, a double-linked list is a linked data structure that consists of a set of sequentially linked records called *entries*. Each *entry* contains two fields, called links, that are references to the previous (*Prev*) and to the next (*Next*) entry in the sequence of entries. The first (*First*) and last (*Last*) entries' *Prev* and *Next* links, respectively, point to a terminator, *NULL*, to facilitate traversal of the list.

Please add one module **ImageList.c** (**ImageList.h**) to your *MovieLab* program.

In this module please define the following two structures:

- The structure for the image list entry **IENTRY**:

```
typedef struct ImageEntry IENTRY;

struct ImageEntry
{
   ILIST  *List;   /* pointer to the list which this entry belongs to */
   IENTRY *Next;   /* pointer to the next entry, or NULL              */
   IENTRY *Prev;   /* pointer to the previous entry, or NULL          */
   IMAGE  *Image;  /* pointer to the struct for the image             */
};
```

- The structure for the image list **ILIST**:

```
typedef struct ImageList ILIST;

struct ImageList
{
   unsigned int  Length;    /* Length of the list                  */
   IENTRY *First;           /* pointer to the first entry, or NULL */
   IENTRY *Last;            /* pointer to the last entry, or NULL  */
};
```

In the same module, please define the following double-linked list functions:

```
/* allocate a new image list */
ILIST *NewImageList(void);

/* delete a image list (and all entries) */
void DeleteImageList(ILIST *l);

/* insert a student into a list */
void AppendImage(ILIST *l, IMAGE *image);

/* reverse an image list */
void ReverseImageList(ILIST *l);
```

Note: Please refer to the slides of **lecture 14** for the implementation of double-linked list.

1.3.3 The Movie.c module

Please add one module **Movie.c** (**Movie.h**) to handle basic operations on the movie.

- **The MOVIE struct**: We will use a *struct type* to aggregate all the information of one movie. Please define the following struct in **Movie.h**:

```
/* the structure for MOVIE */
typedef struct{
    ILIST *Frames;          /* the pointer to the frame list */
    unsigned int Width;     /* movie frame width             */
    unsigned int Height;    /* movie frame height            */
    unsigned int NumFrames; /* total number of frames        */
}MOVIE;
```

- Define the following functions for basic movie operations. Please use the following function prototypes (in **Movie.h**) and define the functions properly (in **Movie.c**)

```
/* allocate the memory space for the movie         */
/* and the memory space for the frame list.        */
/* return the pointer to the movie                 */
MOVIE *CreateMovie(unsigned int nFrames, unsigned int W, unsigned int H);

/*release the memory space for the frames and the frame list. */
/*release the memory space for the movie.                      */
void DeleteMovie(MOVIE *movie);

/* convert the YUV image into the RGB image */
void YUV2RGBImage(IMAGE *YUVImage, IMAGE *RGBImage);

/* convert the RGB image into the YUV image */
void RGB2YUVImage(IMAGE *RGBImage, IMAGE *YUVImage);
```

- Conversion between YUV and RGB:
  The conversion between YUV pixel formats (used by the image and movie compression methods) and RGB format (used by many hardware manufacturers) can be done by the following formulas. They show how to compute a pixel's value in one format from the pixel value in the other format.

  Please using the following equations for the *YUV2RGBImage* and *RGB2YUVImage* functions.

  - Conversion from RGB to YUV:
    $Y = ((66*R+129*G+25*B+128) >> 8)+16$
    $U = ((-38*R-74*G+112*B+128) >> 8)+128$
    $V = ((112*R-94*G-18*B+128) >> 8)+128$
  - Conversion from YUV to RGB:
    $C = Y-16$
    $D = U-128$
    $E = V-128$
    $R = clip((298*C+409*E+128) >> 8)$
    $G = clip((298*C-100*D-208*E+128) >> 8)$
    $B = clip((298*C+516*D+128) >> 8)$

  Here, $clip()$ denotes clipping a value to the range of 0 to 255.
  More specifically,
  $clip(x) = x$, if $0 \le x \le 255$;
  $clip(x) = 0$, if $x \le 0$;
  $clip(x) = 255$, if $x \ge 255$.
  NOTE: please use type *int* for the variables for the calculation.

1.3.4 The MovieLab.c module

Extend the **MovieLab.c** template module as the top module of the *MovieLab* program.

- Support for command-line arguments:
  The C language provides a method to pass arguments to the main() function. This is typically accomplished by specifying arguments on the operating system command line (console).

  The prototype for *main()* looks like:

```
int main(int argc, char *argv[])
{
    ...
}
```

  There are two parameters for the *main()* function. The first parameter is the number of items on the command line (int argc). Each argument on the command line is separated by one or more spaces, and the operating system places each argument directly into its own null-terminated string. The second parameter of *main()* is an array of pointers to the character strings containing each argument (char *argv[]).

  Please add command-line argument support for the *MovieLab.c* program.

  The following options should be supported:

  - **-i**: provide the input file name
  - **-o**: provide the output file name
  - **-f**: determine how many frames desired in the input stream
  - **-s**: resolution of the input stream
  - **-m**: create a movie with Mandelbrot zoom sequence
  - **-bw**: activate the conversion to black and white
  - **-n**: activate the conversion to negative
  - **-hf**: activate horizontal flip
  - **-vf**: activate vertical flip
  - **-rvs**: reverse the frame order of the input stream
  - **-h**: show this usage information

  The *MovieLab.c* template file contains the sample code for the support of **"-i"**, **"-o"** and **"-h"** options. Please extend the code accordingly to support the rest of the options.

  NOTE: The *MovieLab* program will only perform one operation a time. If the user gives more than one option among **"-h"**, **-m**, **"-bw"**, **"-n"**, **"-hf"**, **"-vf"** and **"-rvs"**, please just perform the one with the highest priority. The option priority from high to low is: **"-h"**, **-m**, **"-bw"**, **"-n"**, **"-hf"**, **"-vf"** and **"-rvs"**.

  The **"-i"**, **"-o"**, **"-f"**, **"-s"** options are mandatory to *MovieLab* with two exceptions:

  1. the user just wants to see the usage information (option **"-h"**)
  2. the user wants to create the movie (option **"-m"**), then option **"-i"** is not mandatory.

  Please show proper warning messages and terminate the execution of *MovieLab* if any of the options is missing as the command-line argument.

  In order to get two integer values for the **"-s"** option, please use the following piece of code: (assume that the *i*th command-line argument contains these two values)

```
unsigned int W, H;
if(sscanf(argv[i], "%dx%d", &W, &H) == 2){
    /* input is correct */
    /* the width is stored in W */
    /* the height is stored in H */
}
else{
    /*input format error*/
}
```

If we run the *MovieLab* with the **"-h"** option, we will have:

```
% ./MovieLab -h

Format on command line is:
MovieLab -i input_file_name -o output_file_name -f number_of_frames
         -s widthxheight -h|-m|-bw|-n|-hf|-vf|-rvs
-i      to provide the input file name
-o      to provide the  output file name
-f      to determine how many frames desired in the input stream
-s      to resolution of the input stream (widthxheight)
-m      to generate the movie with Mandelbrot zoom sequence
-bw     to activate the conversion to black and white
-n      to activate the conversion to negative
-hf     to activate horizontal flip
-vf     to activate vertical flip
-rvs    to reverse the frame order of the input stream
-h      to show this usage information
```

Otherwise, we need to run the *MovieLab* with proper information for the movie and operation options, e.g:

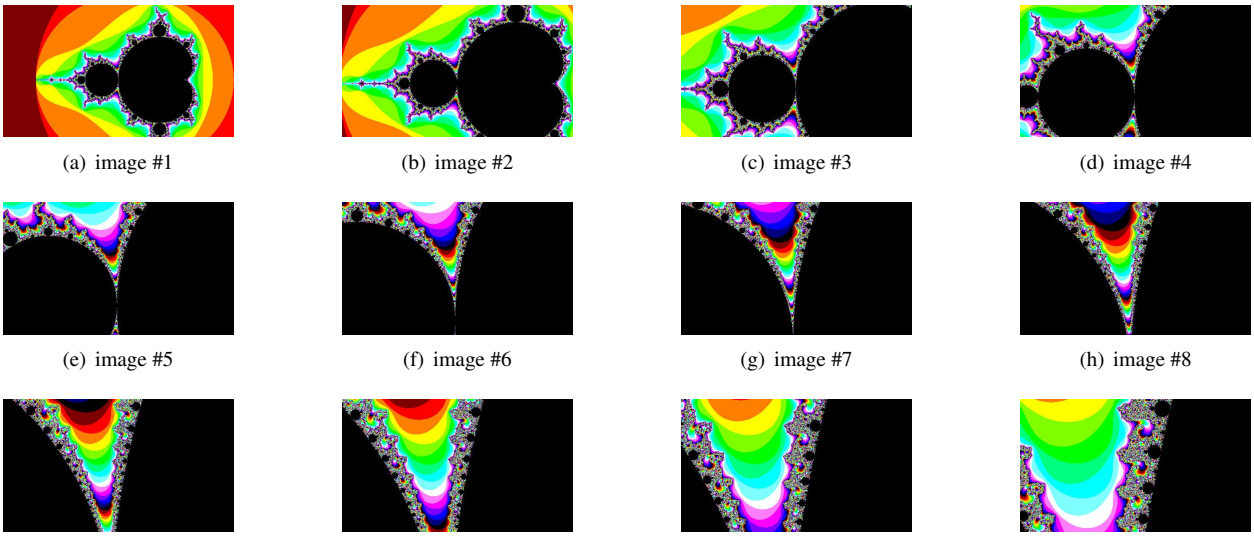
```
% ./MovieLab -i bird -o out -f 141 -s 352x288 -bw
The movie file bird.yuv has been read successfully!
Operation BlackNWhite is done!
The movie file out.yuv has been written successfully!
```

- Read and write movie files

  We have the file I/Os functions defined in *MovieLab.c* module.

  The function signatures for the file I/Os functions are:

  - **int ReadOneFrame(const char* fname, int nFrame, unsigned int W, unsigned int H, IMAGE *frame): (already defined in the template file *MovieLab.c*)**
    reads the file with name *fname.yuv*, and load the color intensities for channel Y, U and V of the *nFrame*th frame into the memory spaces pointed by *frame→R_Y*, *frame→G_U* and *frame→B_V*.
  - **int SaveMovie(const char *fname, MOVIE *movie): (already defined in the template file *MovieLab.c*)**
    opens the movie file with name *fname.yuv*, and stores the frames of the movie to *fname.yuv*.
  - **int ReadMovie(const char *fname, int nFrame, unsigned int W, unsigned int H, MOVIE *movie): (to be defined)**
    reads the file with name *fname.yuv*, and load the frames of this movie file. Please call *ReadOneFrame()* and *AppendImage()* function to implement this function.



(a) image #1 (b) image #2 (c) image #3 (d) image #4

(e) image #5 (f) image #6 (g) image #7 (h) image #8

(i) image #9 (j) image #10 (k) image #11 (l) image #12

Figure 1: The first 12 images in the Mandelbrot zoom sequence where the zoom ratio is 0.7

For *ReadOneFrame()* and *ReadMovie()* function, you need to allocate the memory space to the *image* and *movie* before you call this function to get the content of the input movie file. At the end of your program, you need to free these memory spaces to avoid memory leakage.

- Create the movie with Mandelbrot zoom sequence: Instead of reading an existing movie, we will also add the function to create a movie to show the Mandelbrot zoom sequence.

  As indicated in `http://en.wikipedia.org/wiki/Mandelbrot_set`, the Mandelbrot set shows more intricate detail the closer one looks or magnifies the image, usually called "zooming in". Fig. 1 shows the first twelve images of the Mandelbrot zoom sequence with the zoom-in ratio of 0.7.

  The basic idea for this function is to create an image which belongs to the Mandelbrot zoom sequence, convert the color space from RGB to YUV, append this image as a frame into a movie, generate the next image in the zoom sequence, and repeat the previous steps until there are enough frames for the movie. After finishing generating the sequence, save the movie as a file to the disk by calling the *SaveMovie()* function.

  The following steps are needed to generate a Mandelbrot zoom sequence with fractal shapes

  1. Shift the center of the Mandelbrot space
     In assignment4, we scale the x coordinate of the pixels into the range of [-2.5, 1], and the y coordinate into the range of [-1, 1]. Hence, the complex number that is in the center of the image is -0.75+0i, where -0.75 = (-2.5 + 1)/2 and 0 = (-1 + 1) / 2.
     In order to get a sequence of images with nice fractal shapes, we need to move the center of the image from -0.75+0i to -0.743643887037158704752191506114774L + 0.131825904205311970493132056385139Li.
  2. Use a proper zooming-in ratio
     In order to zoom in the Mandelbrot image, we need to scale the pixels of the image into smaller ranges. For example, given a zooming-in ratio of 0.7, and unshifted Mandelbrot image:
     The range for the real axis (x) and the imaginary axis (y) of the first image is [-2.5, 1], and [-1, 1] respectively;
     The range for the real axis (x) and the imaginary axis (y) of the second image is [-1.75, 0.7], and [-0.7, 0.7] respectively;
     The range for the real axis (x) and the imaginary axis (y) of the third image is [-1.225, 0.49], and [-0.49,

0.49] respectively;
....

3. Scale / map the coordinates of the pixels to the corresponding Mandelbrot complex numbers
   In this step, you need to define the mapping function from the pixel coordinate to the complex number which falls in the complex plane for a specific Mandelbrot image.

You can extend your *Mandelbrot()* function in assignment4 to create one image in the zoom sequence. The following two functions are suggested to define for this operation.

```
/* Mandelbrot in DIPs.c */
IMAGE *Mandelbrot(
  unsigned int W,             /* the width of the image                      */
  unsigned int H,             /* the height of the image                     */
  unsigned int max_iteration, /* the max iteration for Mandelbrot computation */
  long double ratio,          /* the zoom ratio                              */
  long double offsetv,        /* the vertical offset of the complex plane    */
  long double offseth         /* the horizontal offset of the complex plane  */
)

/*Fill the Mandelbrot images as the frames to a movie in MovieLab.c */
int MandelbrotMovie(
  int nFrame,         /* number of the frames in the movie */
  unsigned int W,     /* the width of the movie            */
  unsigned int H,         /* the height of the movie           */
  MOVIE *movie        /* the point to the output movie     */
)
```

Note that, you may need to use the variables of *long double* to get better precisions for the computation in this operation, and you need to manage the memory space for both the images and the movie properly.

For testing, please try to generate a Mandelbrot zoom sequence of 80 frames with 532x304 pixel per frame, and set the zooming-in ratio to be 0.7. Please name the output file as "mandelbrot". More specifically, you can use the following command to create the Mandelbrot zoom sequence and view the movie.

```
% ./MovieLab -o mandelbrot -f 80 -s 532x304 -m
Creating Mandelbrot frame #1
Creating Mandelbrot frame #2
Creating Mandelbrot frame #3
...
Creating Mandelbrot frame #80
% ~eecs22/bin/yay -f 1 -s 532x304 mandelbrot.yuv
```

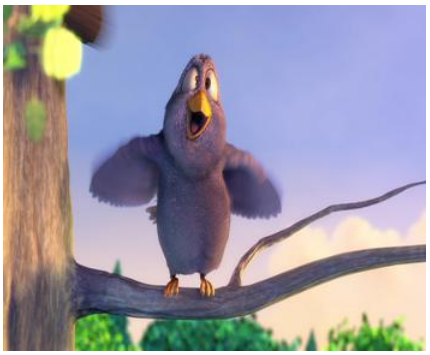
You can also use the following command to check the reference Mandelbrot movie in the *eecs22* account.

```
% ~eecs22/bin/yay -f 1 -s 532x304 ~eecs22/hw5/mandelbrot.yuv
```

- Perform DIP operations on the movie:
  We will add the support for 5 operations onto the movie file:

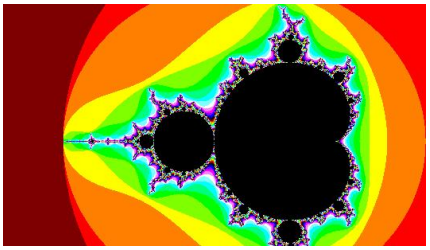
  – Change a Color Movie to Black and White (**"-bw"** option):
    Traverse the frame list of the movie, and perform *BlackNWhite()* operation for each frame images.

    Fig. 2 shows an example of this operation. The execution of our program should be like:

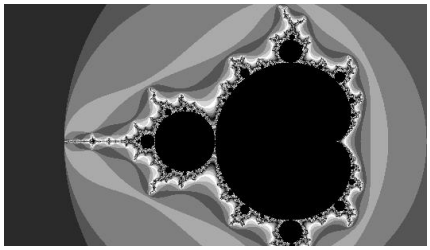


(a) Color movie (bird.yuv, frame: 108)

(b) Black and white movie (bird.yuv, frame: 108)



(c) Color movie (mandelbrot.yuv, frame: 1)



(d) Black and white movie (mandelbrot.yuv, frame: 1)

Figure 2: The color movies and their black and white counterparts.

```
%./MovieLab -i bird -o out -f 141 -s 352x288 -bw
The movie file bird.yuv has been read successfully!
Operation BlackNWhite is done!
The movie file out.yuv has been written successfully!
```

– Make a negative of a movie (**"-n"** option):
Traverse the frame list of the movie, and perform *Negative()* operation for each frame images.
Fig. 3 shows an example of this operation. The execution of our program should be like:

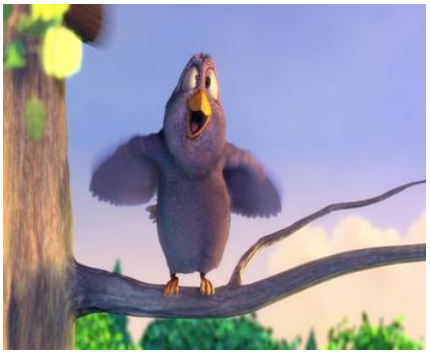
```
%./MovieLab -i bird -o out -f 141 -s 352x288 -n
The movie file bird.yuv has been read successfully!
Operation Negative is done!
The movie file out.yuv has been written successfully!
```

– Flip the movie Horizontally (**"-hf"** option):
Traverse the frame list of the movie, and perform *HFlip()* operation for each frame images.
Fig. 4 shows an example of this operation. The execution of our program should be like:

```
%./MovieLab -i bird -o out -f 141 -s 352x288 -hf
The movie file bird.yuv has been read successfully!
Operation HFlip is done!
The movie file out.yuv has been written successfully!
```

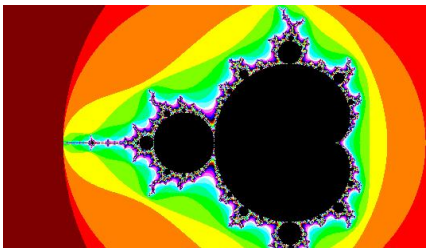
– Flip the movie Vertically (**"-vf"** option):
Traverse the frame list of the movie, and perform *VFlip()* operation for each frame images.
Fig. 5 shows an example of this operation. The execution of our program should be like:

```
%./MovieLab -i bird -o out -f 141 -s 352x288 -vf
The movie file bird.yuv has been read successfully!
Operation VFlip is done!
The movie file out.yuv has been written successfully!
```

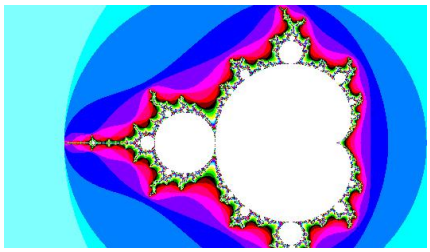


(a) Original movie (bird.yuv, frame: 108)

(b) Negative movie (bird.yuv, frame: 108)

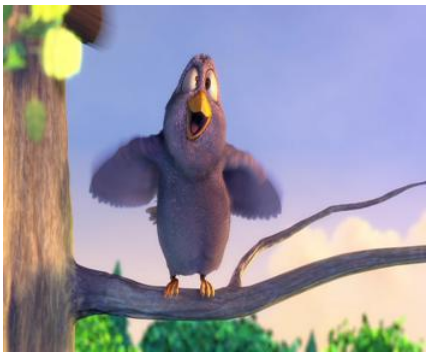


(c) Original movie (mandelbrot.yuv, frame: 1)

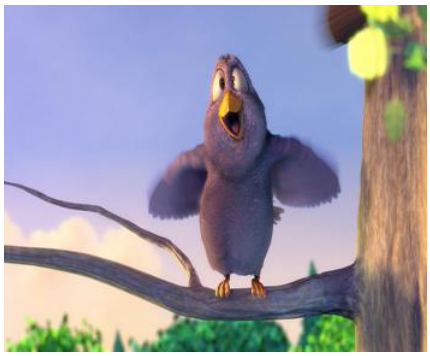


(d) Negative movie (mandelbrot.yuv, frame: 1)

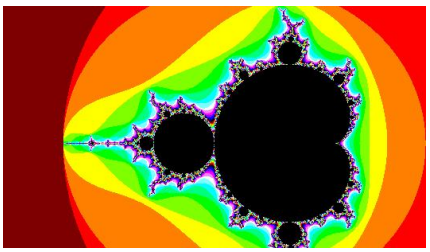
Figure 3: The movies and their negative counterparts.



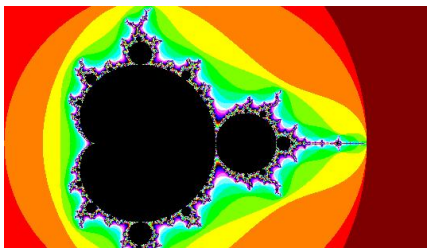
(a) Original movie (bird.yuv, frame: 108)



(b) Horizontally flipped movie (bird.yuv, frame: 108)

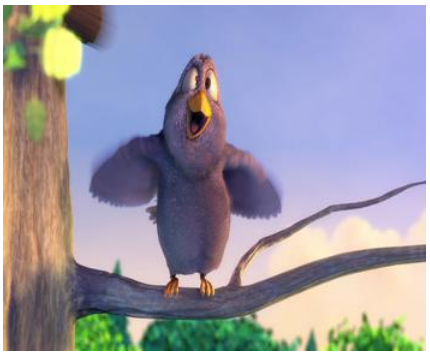


(c) Original movie (mandelbrot.yuv, frame: 1)

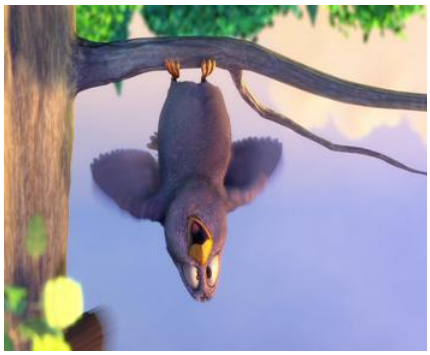


(d) Horizontally flipped movie (mandelbrot.yuv, frame: 1)

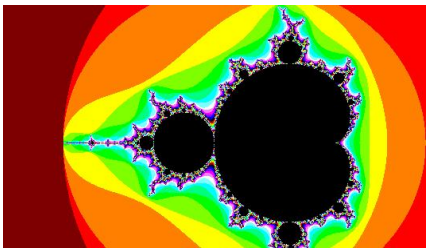
Figure 4: The movies and their horizontally flipped counterparts.



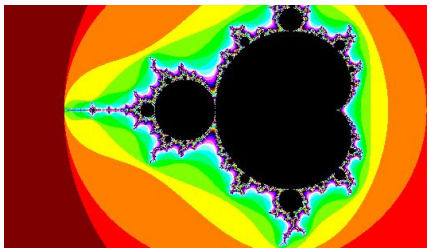
(a) Original movie (bird.yuv, frame: 108)



(b) Vertically flipped movie (bird.yuv, frame: 108)



(c) Original movie (mandelbrot.yuv, frame: 1)



(d) Vertically flipped movie (mandelbrot.yuv, frame: 1)

Figure 5: The movie and their vertically flipped counterparts.

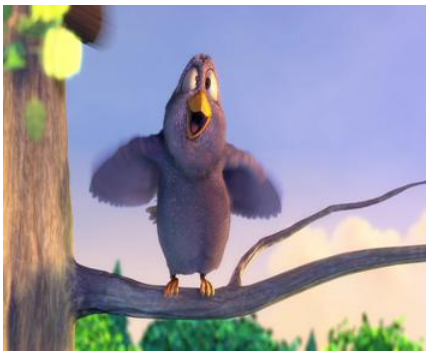
All of the above DIP operations are implemented in assignment3 for the image (in *DIPs.c* module). The function signatures for these image DIP functions are in *DIPs.h*:

```
/* change color image to black & white */
void BlackNWhite(IMAGE *image);

/* reverse image color */
void Negative(IMAGE *image);

/* flip image horizontally */
void HFlip(IMAGE *image);

/* flip image vertically */
void VFlip(IMAGE *image);
```

These four functions have the same parameter lists and return values.

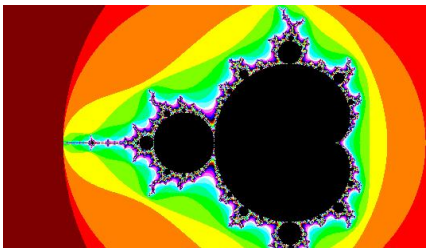
We will define one function *Movie_DIP_Operation()* to traverse the frame list of the movie, and perform the DIP operations on each frame. Function pointers will be used for performing different operations onto the frame images in the *Movie_DIP_Operation()* function:

```
/* type define the function pointer to the DIP function  */
typedef void MOP_F(IMAGE *image);

/* the function for perform DIP operations on the movie*/
void Movie_DIP_Operation(MOVIE *movie, MOP_F *MovieOP);
```



(a) Original movie (bird.yuv, frame: 108)

(b) Frame order reversed movie (bird.yuv, frame: 108)



(c) Original movie (mandelbrot.yuv, frame: 1)

(d) Frame order reversed movie (mandelbrot.yuv, frame: 1)

Figure 6: The movie and their frame order reversed counterparts.

When we need to turn the movie into black and white, we pass the function pointer of *BlackNWhite()* to the *Movie_DIP_Operation()* function as:

```
Movie_DIP_Operation(movie, BlackNWhite);
```

When we need to turn the movie into negative, we pass the function pointer of *Negative()* to the *Movie_DIP_Operation()* function as:

```
Movie_DIP_Operation(movie, Negative);
```

NOTE: Please refer to the slides of **lecture 15 or later** for the examples of function pointers.

NOTE: We will reuse the *DIP.c* module for this assignment. Please adjust your Makefile accordingly with proper target and dependencies. Please include the header file (*DIPs.h*) properly in your source code.

- Reverse the frame order of the movie **(bonus: 10 pts)**:
  Please support the *reverse frame order ("-rvs")* operation for the *MovieLab* program. What you need to do is to reverse the doubly-linked frame list of the movie.

  Fig. 6 shows an example of this operation. The execution of our program should be like:

```
%./MovieLab -i bird -o out -f 141 -s 352x288 -rvs
The movie file bird.yuv has been read successfully!
Operation ReverseMovie is done!
The movie file out.yuv has been written successfully!
```

NOTE: Due to the space limitation for the account on the *ladera* Linux server, please use the same output file name, i.e. out.yuv, when you test your program so as to save disk spaces.

1.4 Build the Makefile

Please create your own **Makefile** with at least the following targets:

- **all:** the dummy target to generate the executable program *MovieLab*.
- **clean:** the target to clean all the intermedia files, e.g. object files, the generated .yuv file, and the executable program.
- ***.o:** the target to generate the object file *.o from the C source code file *.c.
- **MovieLab:** the target to generate the executable program *MovieLab*.

2 Implementation Details

2.1 Structure Definitions

For this assignment, you need to define the following structures in **ImageList.h**:

```
typedef struct ImageList ILIST;
typedef struct ImageEntry IENTRY;

struct ImageEntry
{
   ILIST  *List;   /* pointer to the list which this entry belongs to */
   IENTRY *Next;   /* pointer to the next entry, or NULL              */
   IENTRY *Prev;   /* pointer to the previous entry, or NULL          */
   IMAGE  *Image;  /* pointer to the struct for the image             */
};

struct ImageList
{
   unsigned int  Length;    /* Length of the list                  */
   IENTRY *First;           /* pointer to the first entry, or NULL */
   IENTRY *Last;            /* pointer to the last entry, or NULL  */
};
```

The following structure in **Movie.h**:

```
/* the structure for MOVIE */
typedef struct{
    ILIST *Frames;          /* the pointer to the frame list */
    unsigned int Width;     /* movie frame width             */
    unsigned int Height;    /* movie frame height            */
    unsigned int NumFrames; /* total number of frames        */
}MOVIE;
```

2.2 Function Prototypes

For this assignment, you need to define the following functions in **ImageList.c** module:

```
/* allocate a new image list */
ILIST *NewImageList(void);

/* delete a image list (and all entries) */
void DeleteImageList(ILIST *l);
```

```
/* insert a student into a list */
void AppendImage(ILIST *l, IMAGE *image);

/* reverse an image list */
void ReverseImageList(ILIST *l);
```

The following functions in **Movie.c** module:

```
/* allocate the memory spaces for the movie              */
/* and the memory spaces for the frame list.             */
/* return the pointer to the movie                       */
MOVIE *CreateMovie(unsigned int nFrames, unsigned int W, unsigned int H);

/*release the memory spaces for the frames and the frame list. */
/*release the memory spaces for the movie.                     */
void DeleteMovie(MOVIE *movie);

/* convert the YUV image into the RGB image */
void YUV2RGBImage(IMAGE *YUVImage, IMAGE *RGBImage);

/* convert the RGB image into the YUV image */
void RGB2YUVImage(IMAGE *RGBImage, IMAGE *YUVImage);
```

The following functions in **MovieLab.c** module:

```
/*read the movie frames from the input file */
int ReadMovie(const char *fname, int nFrame, unsigned int W, unsigned int H, MOVIE *movie),

/* the function for perform DIP operations on the movie*/
void Movie_DIP_Operation(MOVIE *movie, MOP_F *MovieOP);

/*main function*/
int main(int argc, char *argv[]);
```

You may want to define other functions as needed.

3 Budgeting your time

You have two weeks to complete this assignment, but we encourage you to get started early. We suggest you budget your time as follows:

- Week 1:
  1. Add the command-line argument support in the *main()* function.
  2. Design the **ImageList.c** (**ImageList.h** as the header file) module.
  3. Build the **Makefile**.
  4. Design the **Movie.c** (**Movie.h** as the header file) module if possible.
- Week 2:
  1. Finish the **Movie.c** (**Movie.h** as the header file) module.
  2. Finish the **MovieLab.c** module.
  3. Finalize the **Makefile**.
  4. Use *Valgrind* to check memory usage. Fix the code if *Valgrind* complains about any errors or memory leaks.
  5. Script the result of your programs and submit your work.

4 Script File

To demonstrate that your program works correctly, perform the following steps and submit the log as your script file:

1. Start the script by typing the command: *script*
2. Compile *MovieLab* by using your **Makefile**
3. run the program: *% MovieLab -h*
4. run the program: *% MovieLab -o mandelbrot -f 80 -s 532x304 -m*
5. run the program: *% MovieLab -i mandelbrot -o out -f 80 -s 532x304 -bw*
6. run the program: *% MovieLab -i mandelbrot -o out -f 80 -s 532x304 -n*
7. run the program: *% MovieLab -i mandelbrot -o out -f 80 -s 532x304 -hf*
8. run the program: *% MovieLab -i mandelbrot -o out -f 80 -s 532x304 -vf*
9. run the program: *% MovieLab -i mandelbrot -o out -f 80 -s 532x304 -rvs*
10. run the program: *% MovieLab -i bird -o out -f 141 -s 352x288 -bw*
11. run the program: *% MovieLab -i bird -o out -f 141 -s 352x288 -n*
12. run the program: *% MovieLab -i bird -o out -f 141 -s 352x288 -hf* under the monitor of *Valgrind*
13. run the program: *% MovieLab -i bird -o out -f 141 -s 352x288 -vf* under the monitor of *Valgrind*
14. run the program: *% MovieLab -i bird -o out -f 141 -s 352x288 -rvs* under the monitor of *Valgrind*
15. run the program: *% MovieLab -o mandelbrot -f 2 -s 532x304 -m* under the monitor of *Valgrind*
16. Clean all the object files, generated .yuv file and executable program by using your **Makefile**.
17. Stop the script by typing the command: *exit*.
18. Rename the script file to *MovieLab.script*.

NOTE: The script file is important, and will be checked in grading; you must follow the above steps to create the script file. ***Please don't open any text editor while scripting !!!***

5 Submission

Use the standard submission procedure to submit the following files as the whole package of your program:

- *MovieLab.c*
- *MovieLab.script*
- *ImageList.c*
- *ImageList.h*
- *Movie.c*
- *Movie.h*
- *Makefile*