EECS 22: Assignment 5

Prepared by: Che-Wei Chang, Prof. Rainer Doemer

November 20, 2014

Due on Thursday 12/09/2014 11:00pm.

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.

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

1.1 Introduction

A movie is basically a sequence of images with different contents but same fixed size. Playing a movie is actually showing 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, we will work on a movie with a fixed number of frames (150) and resolution (352×288 pixels/frame), but your program should be able to handel other sizes as well. The color space of the images in the movie is **YUV** format (http://en.wikipedia.org/wiki/YUV) instead of **RGB**.

In YUV format, the color of each pixel is still represented by 3 components, now referred to as *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 handle the images using the **RGB** color space, conversion is needed to change the **YUV** 3-tuple into a **RGB** tuple for each pixel. The **YUV** color space is very common for video streams. Our input and outputs 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 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. Please integrate the DIP functions defined in the DIPs.c and Advanced.c from the Assignment 4 into a new DIPs.c for Assignment 5.

3. Copy the provided files from the *eecs22* account on the *zuma* or *crystalcove* 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 Assignment 4. 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 Assignment 4);
- **MovieLab.c** is the template file with sample code for command-line argument parsing, and the basic file I/O functions.

4. Create a symbolic link to the input movie stream file from the *eecs22* account on the *zuma* or *crystalcove* server.

```
ln -s ˜eecs22/hw5/anteater.yuv
```

anteater.yuv here is symbolic link to the input movie stream in our *eecs22* account. Since we have space limitations for each account on the servers, it is helpful to save disk space for each account by sharing the read-only input file.

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

You will need a *YUV* player to view the movie files. Our provided *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 to the Linux server. For Mac users, 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 `http://www.geo.mtu.edu/geoschem/docs/putty_install.html`.

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 yourfilename.yuv
```

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

```
˜eecs22/bin/yay -s 352x288 anteater.yuv
```

1.3   Design the MovieLab Program

In this assignment, we will design the movie representation in a C program. Fig 1 illustrates the double linked list data sructure for the movie in this assignment.

1.3.1   The Image.c module (provided)

In Assignment 4, 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 Assignment 4. In order to represent the different color representations, we maintain two sets of rename the pointer member variables for *IMAGE*, one for YUV arrays and one for RGB arrays. Now the structure *IMAGE* and the function signatures are as follows: :

```
typedef struct {
  unsigned int Width; /* image width */
  unsigned int Height;  /* image height */
  union {
    struct s_rgb {
      unsigned char *R ; /*pointer to the memory storing R intensities for all pixels*/
      unsigned char *G ; /*pointer to the memory storing G intensities for all pixels*/
      unsigned char *B ; /*pointer to the memory storing B intensities for all pixels*/
    } RGB ;
    struct s_yuv {
      unsigned char *Y ; /*pointer to the memory storing Y values for all pixels*/
      unsigned char *U ; /*pointer to the memory storing U values for all pixels*/
      unsigned char *V ; /*pointer to the memory storing V values for all pixels*/
    } YUV  ;
  } Color;
}IMAGE;

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

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

/*Get the color intensity of the Blue channel of pixel (x, y) in image */
unsigned char GetPixelB(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 SetPixelR(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 SetPixelG(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 SetPixelB(IMAGE *image, unsigned int x,  unsigned int y, unsigned char b);

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

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

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

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

/*Set the color intensity of the U channel of pixel (x, y) in image with value yuv_u */
void SetPixelU(IMAGE *image, unsigned int x, unsigned int y, unsigned char yuv_u);
```

```
/*Set the color intensity of the V channel of pixel (x, y) in image with value yuv_v */
void SetPixelV(IMAGE *image, unsigned int x, unsigned int y, unsigned char yuv_v);



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

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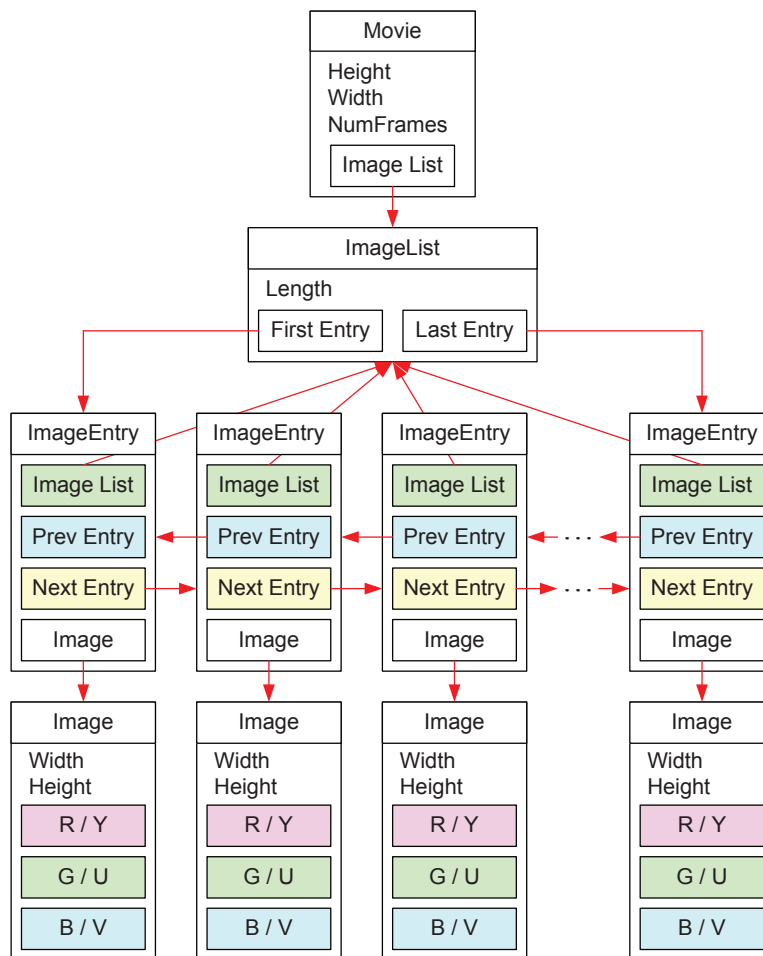

Figure 1: Double Linked List for the Movie

1.3.2 The ImageList.c module

We are going to design a double-linked list to store the frames (images) for the 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 easy traversal of the list.

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

In this module, 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, 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 frame into a list at the end*/
void AppendImage(ILIST *l, IMAGE *image);

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

/*Crop frames from the list starting from SFrame to EFrame*/
void CropImageList(ILIST *l, unsigned int SFrame, unsigned int EFrame) ;

/*Resize the images in the list*/
void ResizeImageList(ILIST *l, unsigned int percentage) ;

/*Fast forward the video*/
unsigned int FastImageList(ILIST *l, unsigned int ff_factor) ;
```

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

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{
    unsigned int Width;     /* movie frame width          */
    unsigned int Height;    /* movie frame height         */
    unsigned int NumFrames; /* total number of frames     */
    ILIST *Frames;          /* the pointer to the frame list */
}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 a YUV image into a RGB image */
void YUV2RGBImage(IMAGE *Image);

/* convert a RGB image into a YUV image */
void RGB2YUVImage(IMAGE *Image);
```

- 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:
    $Y = clip(((66 * R + 129 * G + 25 * B + 128) >> 8) + 16)$
    $U = clip(((-38 * R - 74 * G + 112 * B + 128) >> 8) + 128)$
    $V = clip(((112 * R - 94 * G - 18 * B + 128) >> 8) + 128)$
  - 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 (saturated arithmetic).
  More specifically,
  $clip(x) = x$, if $0 \leq x \leq 255$;

6

$clip(x) = 0$, if $x \leq 0$;
$clip(x) = 255$, if $x \geq 255$.
NOTE: 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).

  Here, 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 [file_name]** to provide the input file name
    - **-o [file_name]** to provide the output file name
    - **-f [no_frames]** to determine how many frames are read from the input stream
    - **-s [WidthxHeight]** to set the resolution of the input stream (widthxheight)
    - **-bw** to activate the conversion to black and white
    - **-vflip** to activate vertical flip
    - **-hmirror** to activate horizontal mirror
    - **-edge** to activate edge filter
    - **-poster** to activate posterize filter
    - **-cut [Start-End]** to crop the frames from the video from frame Start to frame End
    - **-resize [factor]** to resize the video with the provided factor [1-100]
    - **-fast [factor]** to fast forward the video with the provided factor [1+]
    - **-rvs** to reverse the frame order of the input stream
    - **OPTIONAL**: **-m** to generate the movie with Mandelbrot sequences
    - **-h** to 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 can perform multiple operations in an execution except for Mandelbrot movie generation. If the user gives more than one option other than option **"-m"**, please perform the selected options in the following order: **"-bw"**, **"-vflip"**, **"-hmirror"**, **"-edge"**, **"-poster"**, **"-resize"**, **"-cut"**, **"-fast"**, and then **"-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 Mandelbrot movie (option **"-m"**), then option **"-i"** is not mandatory.

Please show proper warning messages and terminate the execution of *MovieLab* if any of the mandetory options are 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 [option]
-i [file_name]    to provide the input file name
-o [file_name]    to provide the output file name
-f [no_frames]    to specify the no. of frames to be read
-s [WidthxHeight] to set resolution of the input stream (widthxheight)
-bw               to activate the conversion to black and white
-vflip            to activate vertical flip
-hmirror          to activate horizontal flip
-edge             to activate edge filter
-poster           to activate posterize filter
-cut [Start-End]  to crop the frames from frame[Start] to frame[End]
-resize [factor]  to resize the video with the provided factor [1-100]
-fast   [factor]  to fast forward the video with the provided factor
-rvs              to reverse the frame order of the input stream
-m                (OPTIONAL) to generate the movie with Mondelbrot sequences
-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 anteater -o out -f 150 -s 352x288 -bw
The movie file anteater.yuv has been read successfully!
Operation BlackNWhite is done!
The movie file out.yuv has been written successfully!
150 frames are written to the file out.yuv in total
```

- Read and write movie files

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

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

- **IMAGE* ReadOneFrame(const char* fname, int nFrame, unsigned int W, unsigned int H):**
  reads the file with name *fname.yuv*, and loads the color intensities for channel Y, U and V of the *n*-th frame into the memory space pointed to by *frame→Color.YUV*.

- **int SaveMovie(const char *fname, MOVIE *movie):**
  opens the movie file with name *fname.yuv*, and stores the frames of the movie into *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 loads the frames of this movie file. Please call the *ReadOneFrame()* and *AppendImage()* functions to implement this function.

For *ReadMovie()* function, you need to allocate the memory space to the *movie* before you call this function to get the content of the input movie file. The *ReadOneFrame* function will take the file name of the image, the resolution of the image, and the frame number to be read as the input, and return a IMAGE pointer to the memory space storing the input frame. At the end of your program, you need to free these memory space to avoid memory leakage.

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

  - Change a Color Movie to Black and White (**"-bw"** option):
    The execution of our program should be like:

    ```
    %./MovieLab -i anteater -o out -f 150 -s 352x288 -bw
    The movie file anteater.yuv has been read successfully!
    Operation BlackNWhite is done!
    The movie file out.yuv has been written successfully!
    150 frames are written to the file out.yuv in total
    ```

  - Flip the movie Vertically (**"-vflip"** option):
    Traverse the frame list of the movie, and perform *Vflip()* operation for each frame image. The execution of our program should be like:

    ```
    %./MovieLab -i anteater -o out -f 150 -s 352x288 -vflip
    The movie file anteater.yuv has been read successfully!
    Operation VFlip is done!
    The movie file out.yuv has been written successfully!
    150 frames are written to the file out.yuv in total
    ```

  - Mirror the movie Horizontally (**"-hmirror"** option):
    Traverse the frame list of the movie, and perform *Hmirror()* operation for each frame image. The execution of our program should be like:

    ```
    %./MovieLab -i anteater -o out -f 150 -s 352x288 -hmirror
    The movie file anteater.yuv has been read successfully!
    Operation HMirror is done!
    The movie file out.yuv has been written successfully!
    150 frames are written to the file out.yuv in total
    ```

  - Create a edge-detected movie (**"-edge"** option):
    Traverse the frame list of the movie, and perform *Edge()* operation for each frame image. The execution of our program should be like:

    ```
    %./MovieLab -i anteater -o out -f 150 -s 352x288 -hmirror
    The movie file anteater.yuv has been read successfully!
    Operation Edge is done!
    The movie file out.yuv has been written successfully!
    150 frames are written to the file out.yuv in total
    ```

– Create a posterized movie (**"-poster"** option):
  Traverse the frame list of the movie, and perform *Posterize()* operation for each frame image. Note that for this assignment, *rbits*, *gbits*, and *bbits* specifying the number of the significant bits that needs to be posterized should be hard coded to **6**, and the user does not have to input the posterize bit information anymore. The execution of our program should be like:

```
%./MovieLab -i anteater -o out -f 150 -s 352x288 -poster
The movie file anteater.yuv has been read successfully!
Operation Posterize is done!
The movie file out.yuv has been written successfully!
150 frames are written to the file out.yuv in total
```

NOTE: All DIP operations asked in this assignemt have the same parameter lists and return values, therefore we will define one function *Movie_DIP_Operation()* and use function pointers to perform different operations onto the frame images in the *Movie_DIP_Operation()* function for the following DIP function.

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

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

/* mirror image horizontally */
void HMirror(IMAGE *image);

/* Edge Detection */
void Edge(IMAGE *image);

/* Posterize */
void Posterize(IMAGE *image);
```

The type definition of the function pointer is shown below:

```
/* 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);
```

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);
```

Please refer to the slides of **Lecture 15 or later** for more detail and examples of function pointers.

• Crop frames from the movie (**"-cut"** option):
  Perform the *CropImageList()* operation for the ImageList in the movie structure. Fig 2 illustrates the concept of cropping operation. In this example, the program will take frame 71 to frame 140 to to generate the new movie with 70 frames.
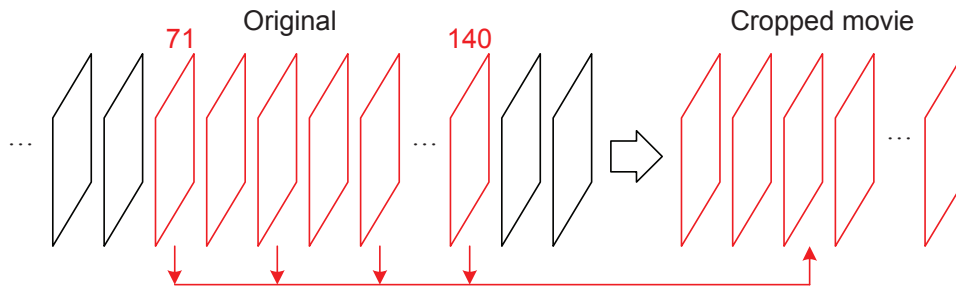
  The execution of our program should be like:

Figure 2: Cropping Operation

```
%./MovieLab -i anteater -o out2 -f 150 -s 352x288 -cut 71-140
The movie file anteater.yuv has been read successfully!
Operation Frame Cropping is done!
The movie file out2.yuv has been written successfully!
70 frames are written to the file out.yuv in total
```

- Resize the image in the movie (**"-resize"** option):
  Perform the *ResizeImageList()* operation for the ImageList in the movie structure with the given resize factor. Note that your program should also print the frame size of the image after resizing. The execution of our program should be like:

```
%./MovieLab -i anteater -o out -f 150 -s 352x288 -resize 75
The movie file anteater.yuv has been read successfully!
Operation Resize is done! New Width/Height = 264x216
The movie file out.yuv has been written successfully!
150 frames are written to the file out.yuv in total
```

- Create a fast forwarded movie (**"-fast"** option):
  Perform the *FastImageList()* operation for the ImageList in the movie structure with the given fast-forward factor. Note that your program should also print the number of frames after fast-forwarded. Fig 3 illustrates the concept of fast forward operation. In this example of fast forwarding by 3, the program will take every third frame from the original one to generate the new movie.
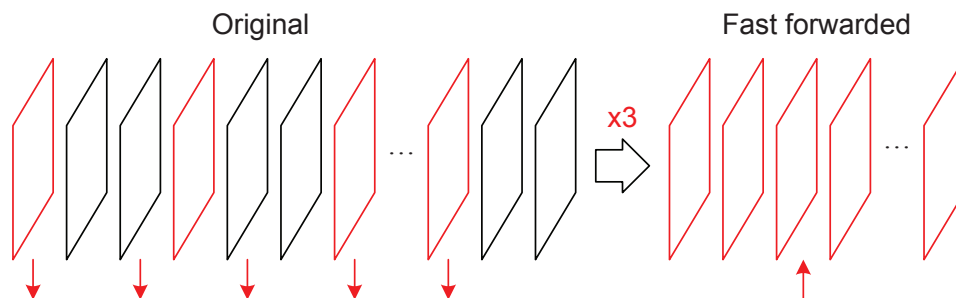

Figure 3: Fast Forward Operation

The execution of our program should be like:

```
%./MovieLab -i anteater -o out -f 150 -s 352x288 -fast 3
The movie file anteater.yuv has been read successfully!
Operation Fast Forward is done! Number of frames = 50
The movie file out.yuv has been written successfully!
50 frames are written to the file out.yuv in total
```
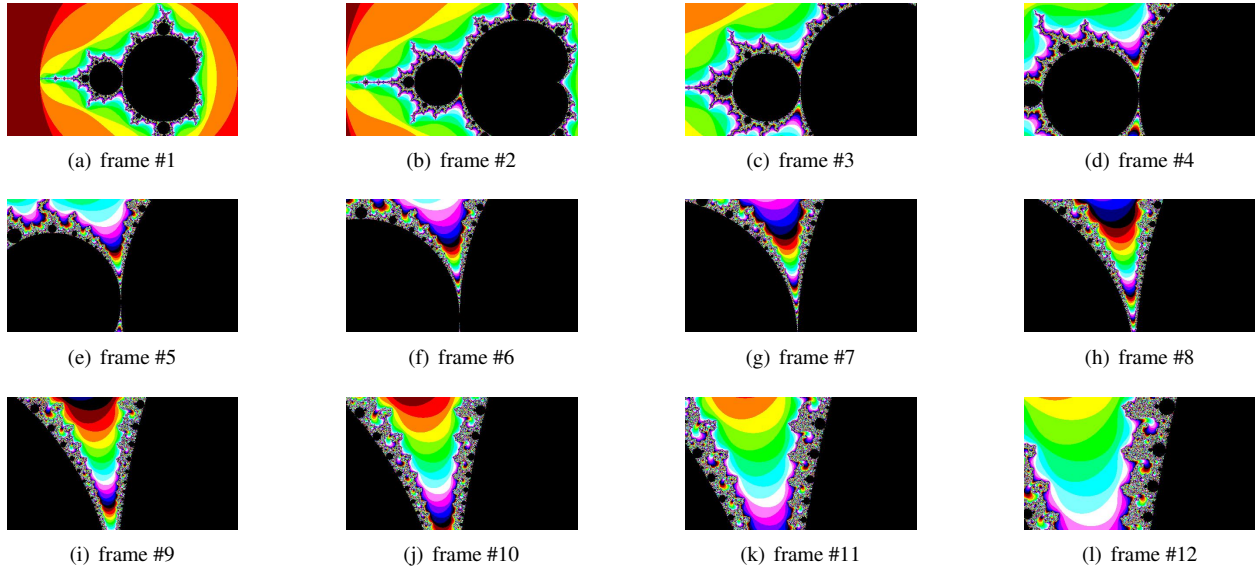
(a) frame #1   (b) frame #2   (c) frame #3   (d) frame #4

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

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

Figure 4: The first 12 images in the Mandelbrot zoom sequence where the zoom ratio is 1.17

- Reverse the frame order in the movie (**"-rvs"** option):
  Perform the *ReverseImageList()* operation for the ImageList in the movie structure. The execution of our program should be like:

```
%./MovieLab -i anteater -o out -f 150 -s 352x288 -rvs
The movie file anteater.yuv has been read successfully!
Operation ReverseMovie is done!
The movie file out.yuv has been written successfully!
150 frames are written to the file out.yuv in total
```

1.3.5 BONUS: Create the movie with Mandelbrot zoom sequence (10 pts)

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

As indicated in `http://en.wikipedia.org/wiki/Mandelbrot_set`, the Mandelbrot shows more intricate detail the closer one looks or magnifies the image, usually called "zooming in". Fig. 4 shows the first twelve images of the Mandelbrot zoom sequence with the zoom-in ratio of 1.17.

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 order to get a sequence of images with nice fractal shapes, we need to move the center of the image to **-0.7 + 0.27015i**. This can be done by setting the **cRe** and **cIm** in the pseudocode of Mandelbrot provided in the handout of assignment 4 to **-0.7** and **0.27015**.

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 zoom-in ratio of 1.17, the *n*-th frame of Mandelbrot sequence is generated with variable **zoom** in the provided pseudocode to 1.17 to the power of *n*.

You can extend your *Mandelbrot()* function in Assignment 4 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,
                  unsigned int H,
                  unsigned int max_iteration,
                  long double zoom
) ;


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

Note that you may need to use the variables of *long double* to get sufficient 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 40 frames with 352x288 pixel per frame, and set the zoom-in ratio to be 1.17. 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 40 -s 352x288 -m
Creating Mandelbrot frame #1
Creating Mandelbrot frame #2
Creating Mandelbrot frame #3
...
Creating Mandelbrot frame #40
% ~eecs22/bin/yay -s 352x288 mandelbrot.yuv
```

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

```
% ~eecs22/bin/yay -s 352x288 ~eecs22/hw5/mandelbrot.yuv
```

NOTE: We will reuse some function defined in *DIP.c* and *Advanced.c* module from Assignment 4. Please integrate these two modules into a new *DIP.c* module for this assignment. You also have to adjust your Makefile accordingly with proper target and dependencies and include the header file (*DIPs.h*) properly in your source code.
NOTE: Due to the space limitation for the account on the Linux server, please use the same output file name, i.e. out.yuv, when you test your program so as to save disk space.

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

Note: For the grading purpose, following the interfaces of the function prototypes defined in this section is mandatory.

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{
    unsigned int Width;     /* movie frame width            */
    unsigned int Height;    /* movie frame height           */
    unsigned int NumFrames; /* total number of frames       */
    ILIST *Frames;          /* the pointer to the frame list */
}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 an image into a list */
void AppendImage(ILIST *l, IMAGE *image);

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

/*Crop frames from the list starting from SFrame to EFrame*/
void CropImageList(ILIST *l, unsigned int SFrame, unsigned int EFrame) ;

/*Resize the images in the list*/
```

```
void ResizeImageList(ILIST *l, unsigned int percentage) ;

/*Fast forward the video*/
unsigned int FastImageList(ILIST *l, unsigned int ff_factor) ;
```

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

```
/* 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 *Image);

/* convert the RGB image into the YUV image */
void RGB2YUVImage(IMAGE *Image);
```

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);

int MandelbrotMovie(int nFrame, unsigned int W, unsigned H, MOVIE *movie);

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

The DIP functions (BlackNWhite, VFlip, HMirror, Edge, Posterize, Resize) should be defined in the **DIPs.c**, and you can reuse those functions from the previous assignment. You may want to define other functions as needed as well.

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.

IMPORTANT: Don't try to finish the whole problem in one step, or the number of compilation errors could be overwhelming. Try to give this assignment more time, and finish it step-by-step.

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 -i anteater -o out -f 100 -s 352x288 -bw - vflip*

5. run the program: *% MovieLab -i anteater -o out -f 100 -s 352x288 -edge*

6. run the program: *% MovieLab -i anteater -o out -f 100 -s 352x288 -poster*

7. run the program: *% MovieLab -i anteater -o out1 -f 150 -s 352x288 -cut 1-70* under the monitor of *Valgrind*

8. run the program: *% MovieLab -i anteater -o out2 -f 150 -s 352x288 -cut 80-150* under the monitor of *Valgrind*

9. run the program: *% MovieLab -i anteater -o out -f 150 -s 352x288 -fast 3* under the monitor of *Valgrind*

10. run the program: *% MovieLab -i anteater -o out -f 150 -s 352x288 -resize 75* under the monitor of *Valgrind*

11. OPTIONAL: run the program: *% MovieLab -o mandelbrot -f 40 -s 352x288 -m*

12. Clean all the object files, generated .yuv file and executable program by using your **Makefile**.

13. Stop the script by typing the command: *exit*.

14. 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*
- *DIPs.c*
- *DIPs.h*
- *Movie.c*
- *Movie.h*
- *Makefile*