Center for Embedded and Cyber-Physical Systems
University of California, Irvine

A Linux-based YUV Video Player

Zhuoqi Li, Rainer Dömer

Technical Report CECS-23-02
May 19, 2023

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

zhuoqil3@uci.uci.edu
http://www.cecs.uci.edu

A Linux-based YUV Video Player

Zhuoqi Li, Rainer Dömer

Technical Report CECS-23-02
May 19, 2023

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
(949) 824-8919

zhuoqil3@cecs.uci.edu
http://www.cecs.uci.edu

Abstract

A video player named "BEST" will be introduced in this report. This tool can play YUV format video in a Linux environment, and the player also implements some basic operations, which can make the users more convenient to control the video playback. At the same time, the tool is based entirely on the C programming language and GTK, which makes it more compatible with the Linux environment. Therefore, this report will describe the details of the development, features, and basic usage of this tool, so that users can better understand the BEST player.

Contents

List of Figures

A Linux-based YUV Video Player

Zhuoqi Li, Rainer Dömer
Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA
zhuoqil3@uci.edu
http://www.cecs.uci.edu

Abstract

A video player named "BEST" will be introduced in this report. This tool can play YUV format video in a Linux environment, and the player also implements some basic operations, which can make the users more convenient to control the video playback. At the same time, the tool is based entirely on the C programming language and GTK, which makes it more compatible with the Linux environment. Therefore, this report will describe the details of the development, features, and basic usage of this tool, so that users can better understand the BEST player.

1 Introduction

With the development of computer technology, an embedded system has been ubiquitous in today's living environment. For example, some intelligent household appliances, refrigerators, televisions, and intelligent systems in automobiles all require embedded systems to assist and control them. At the same time, the embedded system plays an important role in computer control and operation. To be more specific, embedded systems can be divided into hardware and software. In the field of hardware, hardware about embedded applications, such as processors and memory, are an important part of the computer. In addition, operating systems, applications, and other embedded software also give the computer more functions and improve the user's work efficiency. Among them, Linux, as a common operating system, provides a large number of applications for computers to help users improve their work efficiency.

The BEST player in this study also belongs to an embedded model design, used as a software program in the Linux operating system environment. The main purpose of the BEST player is to play the YUV video. However, different from other common YUV players, the BEST player is based directly on the GTK[3] and C programming languages[5], so this player will be more compatible with Linux servers. Moreover, while some traditional YUV players[6] can also do similar jobs, their operations are more complicated, such as coding the command line. Therefore, this new player tool can help to eliminate these unnecessary steps, making video operations easier and greatly increasing user efficiency. In addition, the BEST player also includes the basic operated function of the video, such as pause, fast forward, backward, and replay. These functions also can help the users to manage and edit their videos more efficacious.

2 Method and Procedure

2.1 General Design

As the information provided in Figure 1, YUV videos are composed of multiple frames, so the BEST player has to analyze each frame of the video first and store the corresponding pixel to the pixel buffer. Then, the pixel buffer can be displayed in the image by the usage of the set image methods of GTK. By displaying each frame quickly on the screen in sequence, the viewer can see a coherent sequence of actions or scenes being represented.

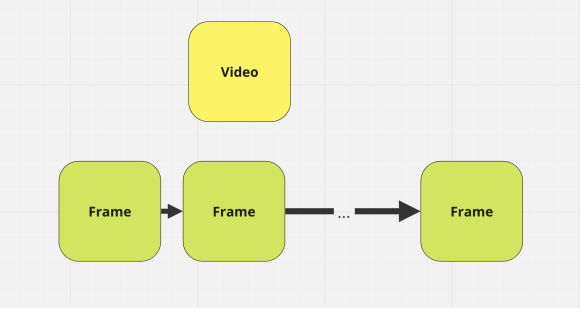

Figure 1: YUV video structure

At the same time, YUV is a common color model and an efficient video storage format[4]. To be more specific, this color model controls the brightness and color of each pixel through three attribute variables Y, U, and V. Thus, users can modify videos by adjusting the YUV variables contained in each pixel, and some YUV players examine these YUV variables to analyze video files and play them. Based on this theorem, the BEST player will also store the color variables from the video file, and then set each variable to the corresponding position in the pixel buffer[2] which is generated by using the method in GTK. After that, the image method "gtk_image_set_from_pixbuf" to obtain the image from the pixel buffer and display it on the window.

However, as mentioned before, the BEST player has to ensure Linux servers are adapted as well as possible, BEST player fully adopts C language and GTK2.0 to implement video functions. Therefore, due to the limitations of GTK2.0, the pixel buffer can only store images in RGB format. Therefore, using YUV variables directly will be difficult to show through GTK. To solve this problem, BEST player will convert each frame in the video from YUV format to RGB using the existing conversion formula before storing the video.[4] After that, the pixel buffer will be filled by the RGB values of each image, and these images will also be rendered on the screen one by one in the loop.

2.2 Data Structure

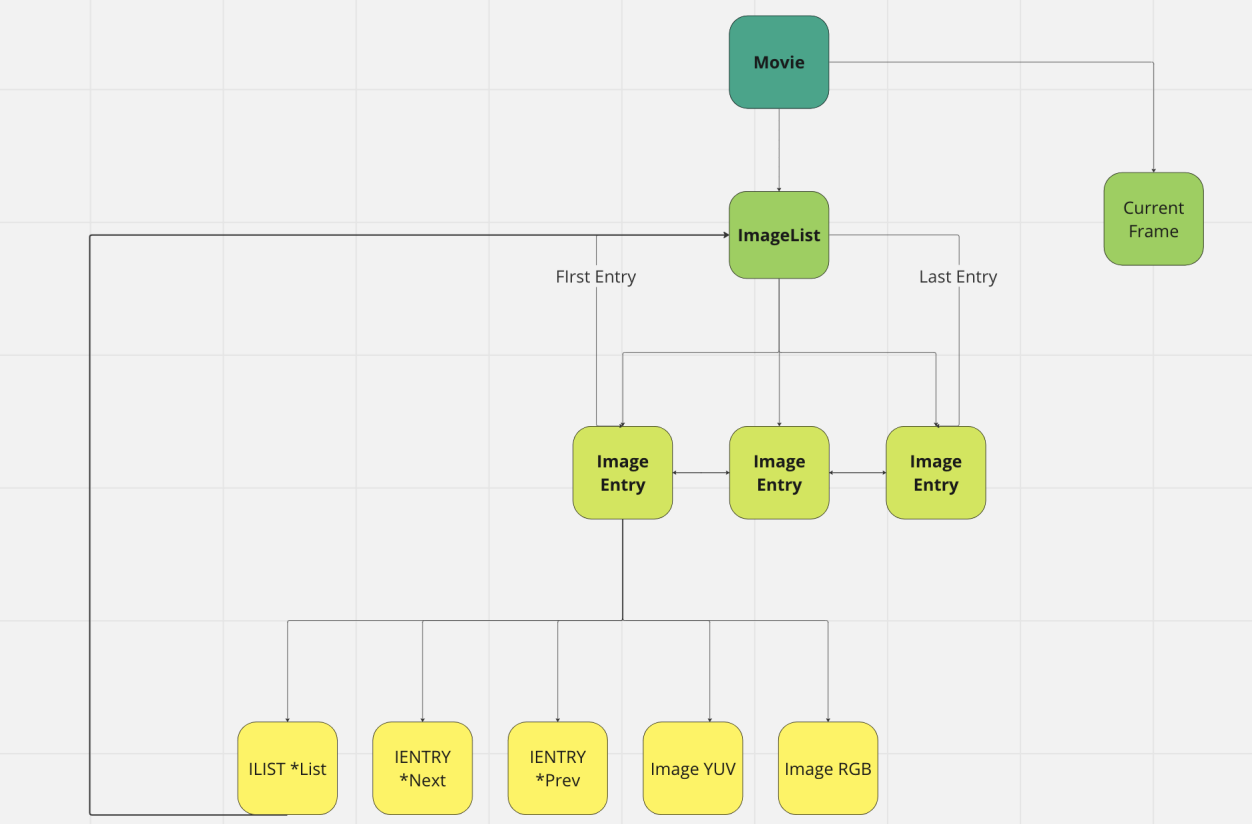

Figure 2: data structure

Since the player needs to do the conversion between YUV and RGB, a special data structure needs to be designed to attain this requirement. Fortunately, the data structure in EECS 22 assignment[1] can be revised and used into this project. As figure 2 shown above, after the YUV video is loaded, the entire video

file is stored in a data structure called Movie. The movie structure consists of ImageList and CurrentFrame. CurrentFrame is a pointer that will point to a frame or an image, and the ImageList will point to the EntryList. At the same time, the first and the last entry will be pointed by ImageList. The EntryList is a linked list that will have a pointer to point to the next or previous entry, and each entry also will point to its parent ImageList. In addition, each entry will store the pixels of the YUV image and the converted RGB image, and these pixels are stored separately in a one-dimensional array of YUV and RGB. In this data structure storage, we can distinguish and save the frames and pixels of the loaded video file.

2.2.1 Pixel Buffer

In addition to storing the video, we also need to store the converted RGB image in the pixel buffer in GTK2.0. According to the method in GdkPixBuf introduction of the GDK website, it introduces the structure of the pixel buffer exactly. Similar to the one-bit array that stores RGB images, a pixel buffer stores the red, green, and blue values of RGB in a one-dimensional form. Besides to RGB value, the pixel buffer also will store the alpha value which will set the transparency of the image. However, since this research will only implement the common situation, the alpha variable will not be considered and used in this tool. After using the method in GTK2.0, we can get the corresponding Channel and RowStride for the pixel buffer. Channel is the size of one pixel, and the RowStride is the size of one row. Therefore, according to the position of the pixel in a frame, the position of the pixel in the pixel buffer also will be calculated. In other words, by allowing the number of rows in which the pixel in the frame is located to be multiplied by a RowStride and adding the product of the number of pixel columns and the number of channels, the player can manipulate the corresponding position of the pixel in the pixel buffer. Hence, we can calculate and store the value of RGB in the corresponding pixel buffer.

2.3 Control Flow

After the video file is loaded, the whole process will enter the Main Control Loop, so that the window and the picture can continue to display. When a video is loaded, every pixel of data is stored in a Movie structure. After that, the YUV data will be converted to RGB. Meanwhile, the second loop which is inside the Main Control Loop in the Main method will detect the state of the pointer in the Movie. If a video is detected to be loaded, the process will enter a second nested Loop and conduct other following assignments. In the second nested loop, the pixel value of the CurrentFrame will be transferred to the pixel buffer and the image on the screen will be updated after the loop is finished.

Also, an If statement checks the current task state in the second nested loop. For example, if the user clicks the Pause button, the process state goes into a paused state ”1”. In this case, the data transfer process will not proceed until the user clicks the Pause button again to restore the process state to ”0”. For other function buttons, they do not affect the running of the main process. When the user clicks the function buttons, such as Forward and Backward, the process will enter the corresponding function. Finally, when the task finishes, the process returns the Main method.

2.4 Player Functions

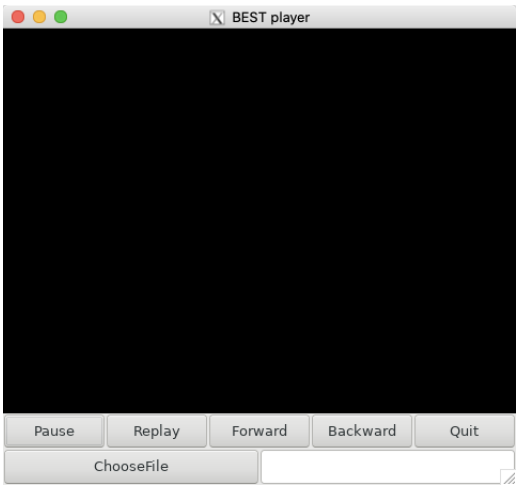

Figure 3: Choose File

The player provides users with some common control functions. As the figure 3 shows, each button in the layout has its own function, and clicking on them allows you to manipulate the video.

2.4.1 Pause

In addition to playing the video, the BEST player offers a range of control options to manipulate the video. The pause option stops the video and keeps the current video screen displayed. When the user clicks the Pause button, the CurrentFrame pointer in the movie structure will point to the current frame which is playing now. At the same time, the status of the video will be set to "1" which represents the pause status. Similarly, the user can select the option to continue playing in the paused state. At this point, the video will continue to play the remaining frames from the paused point. Therefore, the frame which is pointed by the CurrentFrame pointer will be transferred to the pixel buffer, and then this frame will be printed on the screen.

2.4.2 Replay

The replay function allows you to replay the current video at any time. When the user clicks the Replay button, the CurrentFrame pointer will point directly to the first frame of the ImageList, so the video will start playing from the first frame.

2.4.3 Fast-Forward

The video fast-forward feature allows you to jump the video forward several frames to fast-forward the effect. When the user clicks the Fast Forward button, the CurrentFrame will enter a loop to achieve the fast forward function. Because ImageList is LinkedList, we need to loop the CurrentFrame pointer to the next frame over and over again. When the loop is ended, the frame which is pointed by the CurrentFrame will be transferred to the pixel buffer, and the rest of the video continues to play.

2.4.4 Backward

Backward is similar to fast-forward, which also let the CurrentFrame pointer enter a loop and complete the backward assignment. However, different from the fast forward, the CurrentFrame pointer in the Backward mode will ceaselessly point to the previous frame. Therefore, after finishing the loop, the CurrentFrame will point to a frame which is played before. Then, the video will start from this frame and continue to play the rest of the video.

2.4.5 File Choose

The ChooseFile button can help the user to find their YUV video file automatically. After the users click the button, a window will be generated by GTK and it will look through the content library of the user on the server. At the same time, this method will filter the other file and only list the YUV file on the screen. When the user has found the file and clicked the button "OK", the video will be loaded and played by the BEST player.

In addition, the users do not need to input any command line, since the BEST player will help to find the size of the file and the number of frames. The only thing that needs to be done by the user is to provide the resolution of the video. In this way, since the size of each frame in the video is the same, we can allow the total size of the video to be divided by the product of the size of one frame and 1.5, which will help to get the total number of frames. After that, the video can be loaded successfully. Furthermore, the BEST player also provides some error detection functions to check the availability of the video. Therefore, if a video has some problems, the BEST player will print some messages on the screen.

2.4.6 Quit

The Quit button will stop all the processes and close the windows. Then, the BEST player will be exited.

3 User Manual

Before users can run BEST player, they need to ensure the name format of the YUV video. Because when the BEST player loads the video, the name of the video will be detected first so that the resolution

of the video can be obtained. For example, if the width of a video named "example" is 480 and the height is 360, the name of the video would be "example_480_360.yuv".

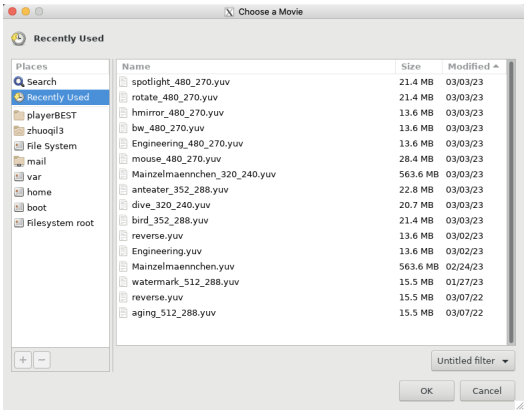

Figure 4: Choose File

Later, as the figure 4 presented above, when the user starts running BEST player, they need to first click the ChooseFile button to select the video file. After clicking, the user's file directory will be displayed on the screen, so the user just needs to find the file they want and click OK to play the video. Then, as figure 5 performed, the video will be played successfully.

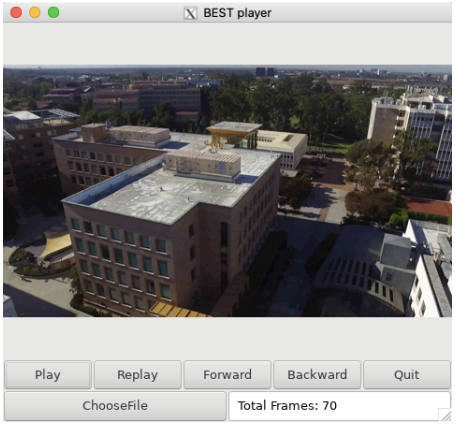

Figure 5: Play Video Successfully

In addition, when the user clicks Pause, the video stops playing. Meanwhile, the Pause button will become the Play button. In other words, if the user continues to click the play button, the video will continue to play. Moreover, users can click the Forward and Backward buttons to show the video going forward and backward. At the same time, users can tap the two buttons at any time a video is playing, regardless of whether the video is paused or playing.

Finally, if users want to quit the video player, they can either click the Quit button or the Close option in the upper left corner of the window. These two closing methods have the same effect and impact, both will end the current task and exit the video player.

4 Conclusion and Future Work

The current version of BEST player can play the YUV files normally. Also, without the help of external extension packs or other resources, this tool can adapt to the Linux operating environment better. At the same time, this tool also provides users with some essential operation modes, which also greatly improves the users' efficiency. However, the BEST player has the potential for more progress. For example, while the BEST player can play normal-resolution video smoothly, it can not guarantee the same smoothness when playing high-resolution video. In future updates, the BEST player will do more consideration and improvement on these issues, and continue to enhance the functionality of the software, so that it can better serve users.

References

[1] EECS 22. https://canvas.eee.uci.edu/courses/41966/pages/assignments.

[2] GdkPixbuf. https://docs.gtk.org/gdk-pixbuf/class.Pixbuf.html.

[3] GTK. https://docs.gtk.org/gtk3/.

[4] Intel® Integrated Performance Primitives Developer Reference. https://www.intel.com/content/www/us/en/docs/ipp/developer-reference/2021-7/color-models.html.

[5] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Pearson Education, 1988.

[6] yay. https://github.com/mattzzw/yay.

5 Appendix

The Appendix section will show the BEST Player code section.

5.1 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <gtk/gtk.h>
4  #include <string.h>
5  #include <assert.h>
6  #include <time.h>
7  #include <math.h>
8  #include <sys/time.h>
9
10 #include "Movie.h"
11 #include "Constants.h"
12 #include "Image.h"
13 #include "ImageList.h"
14 #include "FileIO.h"
15
16 /* 0 is play, 1 is pause */
17 int pausePlay = 0;
18 /* 0 do nothing, 1 repeat */
19 int repeat = 0;
20
21 int image_width = 0;
22 int image_height = 0;
23 int quit = 0;
24 int fps = 33000;
25
26 GtkWidget *pausePlaybtn;
27 GtkWidget *repeatbtn;
28 GtkWidget *fastForwardbtn;
29 GtkWidget *backwardbtn;
30 GtkWidget *window;
31 GdkPixbuf *pixelBuffer;
32 GtkWidget *frame;
33 GtkWidget *quitbtn;
34 GtkWidget *loadbtn, *enter;
35 GtkWidget *fileChoosebtn;
36 GtkWidget *FPSbutton;
37 MOVIE *movie;
38
39 GtkWidget *Vbox, *Hbox, *Hbox2;
40
```

```
41 /* Load one movie frame from the input file */
42 YUVIMAGE *LoadOneFrame(const char *fname, int n,
43                        unsigned int width, unsigned height);
44
45 /* Load the movie frames from the input file */
46 MOVIE *LoadMovie(const char *fname, int frameNum,
47                  unsigned int width, unsigned height);
48
49 /* Saves one movie frame to the output file */
50 void SaveOneFrame(YUVIMAGE *image, const char *fname, FILE *file);
51
52 /* Save the movie frames to the output file */
53 int SaveMovie(const char *fname, MOVIE *movie);
54
55 /* Set the RGB in each pixel of pixel buffer  */
56 void set_pixel(GdkPixbuf *pixbuffer, int w, int h, guchar R, guchar G,
      guchar B, guchar alpha);
57
58 /* Set the pixel in one frame */
59 void drawOneFrame(int image_width, int image_height);
60
61 /* pause the video */
62 void pausePlayMode(GtkWidget *Widget, gpointer Data);
63
64 /* repeat the video and reset the pointer */
65 void repeatMode(GtkWidget *Widget, gpointer Data);
66
67 /* adjust the pointer and fast-forward the video */
68 void fastForwardMode(GtkWidget *Widget, gpointer Data);
69
70 /* adjust the pointer and backward the video */
71 void backwardMode(GtkWidget *Widget, gpointer Data);
72
73 /* exit this tool */
74 void quitMode(GtkWidget *Widget, gpointer Data);
75
76 /* load the movie */
77 void loadMovie(GtkWidget *Widget, gpointer Data);
78
79 /* generate the dialog windows */
80 void dialogStart(GtkWidget *Widget, gpointer Data);
81
82 /* get the width and height from the name of the video file */
83 void get_Width_And_Height(char name[], int widthAndHeight[]);
84
85 /* exit the tool */
```

```
86  void closePlayer(GtkWidget *Widget, gpointer Data);
87
88
89  /* This method will generate a dialog window and initialize some
       information about the video. At the same time, it will help the users
       to get the size of the video file and calculate the number of the total
        frames.*/
90  void dialogStart(GtkWidget *Widget, gpointer Data)
91  {
92
93      gint response;
94      GtkWidget *dialog;
95      char *fileName;
96      const char *fileNameCst;
97      FILE *file;
98      int widthAndHeight[2];
99      int fileSize;
100     int numOfFrame;
101     char totalNum[256];
102
103     dialog = gtk_file_chooser_dialog_new("Choose a Movie", GTK_WINDOW(
            window), GTK_FILE_CHOOSER_ACTION_OPEN, GTK_STOCK_OK,
            GTK_RESPONSE_ACCEPT, GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL, NULL);
104     GtkFileFilter *filter = gtk_file_filter_new();
105     gtk_file_filter_add_pattern(filter, "*.yuv");
106     gtk_file_chooser_add_filter(GTK_FILE_CHOOSER(dialog), filter);
107
108     gtk_widget_show_all(dialog);
109     response = gtk_dialog_run(GTK_DIALOG(dialog));
110     if (response == GTK_RESPONSE_ACCEPT)
111     {
112
113         fileName = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog))
                ;
114
115         /* get the width and height of the video */
116         memset(widthAndHeight, 0, sizeof(widthAndHeight));
117         get_Width_And_Height(fileName, widthAndHeight);
118         //printf("%d, %d\n", widthAndHeight[0], widthAndHeight[1]);
119
120         fileNameCst = fileName;
121         /* get the size of the video */
122         file = fopen(fileNameCst, "r");
123         if (file == NULL)
124         {
125             gtk_entry_set_text(GTK_ENTRY(enter), "File Error");
```

```
126             gtk_widget_destroy(dialog);
127             return;
128         }
129         fseek(file, 0, SEEK_END);
130         fileSize = ftell(file);
131         //printf("filesize: %d\n", fileSize);
132 
133         /* calculate the number of frame */
134         if (widthAndHeight[0] * widthAndHeight[1] != 0)
135         {
136             numOfFrame = fileSize / (widthAndHeight[0] * widthAndHeight[1]
                    * 1.5);
137         }
138         else
139         {
140             gtk_entry_set_text(GTK_ENTRY(enter), "Frame Size Error");
141             gtk_widget_destroy(dialog);
142             return;
143         }
144         if (numOfFrame != (int)numOfFrame)
145         {
146             /* incorrent the size of the frame */
147             gtk_entry_set_text(GTK_ENTRY(enter), "Frame Size Error");
148             gtk_widget_destroy(dialog);
149             return;
150         }
151         //printf("numOfFrame: %d\n", numOfFrame);
152 
153         movie = LoadMovie(fileNameCst, numOfFrame, widthAndHeight[0],
                widthAndHeight[1]);
154         YUV2RGBMovie(movie);
155         movie->currentFrame = movie->Frames->First;
156         //printf("Load success\n");
157 
158         /* update the size of the frame */
159         pixelBuffer = gdk_pixbuf_new(GDK_COLORSPACE_RGB, FALSE, 8,
                widthAndHeight[0], widthAndHeight[1]);
160         memset(totalNum, 0, sizeof(totalNum));
161         sprintf(totalNum, "Total Frames: %d", numOfFrame);
162         gtk_entry_set_text(GTK_ENTRY(enter), totalNum);
163 
164         /* close the file after loading */
165         fclose(file);
166         file = NULL;
167         //printf("out success\n");
168     }
```

```
169     else
170     {
171         gtk_entry_set_text(GTK_ENTRY(enter), "Please choose a file !");
172     }
173
174     gtk_widget_destroy(dialog);
175 }
176
177 /* In this method, each pixel in the frame is traversed and accessed
       through a nested loop, and the RGB value of each pixel is set. */
178 void drawOneFrame(int image_width, int image_height)
179 {
180     int w = 0;
181     int h = 0;
182     for (w = 0; w < image_width; w++)
183     {
184         for (h = 0; h < image_height; h++)
185         {
186             set_pixel(pixelBuffer, w, h, GetPixelR(movie->currentFrame->
                  RGBImage, w, h), GetPixelG(movie->currentFrame->RGBImage, w
                  , h), GetPixelB(movie->currentFrame->RGBImage, w, h), 0);
187         }
188     }
189 }
190
191 /* This method will change the status of video to pause or play the video
       .*/
192 void pausePlayMode(GtkWidget *Widget, gpointer Data)
193 {
194
195     if (pausePlay == 0)
196     {
197         gtk_button_set_label(GTK_BUTTON(pausePlaybtn), "Play");
198         pausePlay = 1;
199     }
200     else
201     {
202         gtk_button_set_label(GTK_BUTTON(pausePlaybtn), "Pause");
203         pausePlay = 0;
204     }
205 }
206
207 /* By changing the current pointer, the first frame of this video will be
       set. */
208 void repeatMode(GtkWidget *Widget, gpointer Data)
209 {
```

```
210     if (movie != NULL)
211     {
212         movie->currentFrame = movie->Frames->First;
213         drawOneFrame(image_width, image_height);
214         repeat = 1;
215         gtk_image_set_from_pixbuf(GTK_IMAGE(frame), pixelBuffer);
216     }
217 }
218
219 /* This method will use loop to achieve the fast forward function. */
220 void fastForwardMode(GtkWidget *Widget, gpointer Data)
221 {
222
223     int i = 0;
224     //printf("enter\n");
225
226     while ((movie != NULL) && (movie->currentFrame != NULL) && (movie->
            currentFrame->Next != NULL) && (i < 5))
227     {
228
229         movie->currentFrame = movie->currentFrame->Next;
230
231         i++;
232     }
233     //printf("out\n");
234
235     if ((movie != NULL) && (movie->currentFrame != NULL))
236     {
237         drawOneFrame(image_width, image_height);
238         gtk_image_set_from_pixbuf(GTK_IMAGE(frame), pixelBuffer);
239     }
240 }
241
242 /*This method will use loop to achieve the backward function. */
243 void backwardMode(GtkWidget *Widget, gpointer Data)
244 {
245
246     int i = 0;
247     while ((movie != NULL) && (movie->currentFrame != NULL) && (movie->
            currentFrame->Prev) && (i < 10))
248     {
249         movie->currentFrame = movie->currentFrame->Prev;
250         i++;
251     }
252
253     if ((movie != NULL) && (movie->currentFrame != NULL))
```

```
254     {
255         drawOneFrame(image_width, image_height);
256         gtk_image_set_from_pixbuf(GTK_IMAGE(frame), pixelBuffer);
257     }
258 }
259
260 /* This method will exit all of the processes and exit the tool */
261 void quitMode(GtkWidget *Widget, gpointer Data)
262 {
263     quit = 1;
264 }
265
266 /* This method will do the same thing with quitMode and exit the tool  */
267 void closePlayer(GtkWidget *Widget, gpointer Data)
268 {
269
270     //printf("close player\n");
271     quit = 1;
272 }
273
274 /*The real method will assign each pixel's RGB value */
275 void set_pixel(GdkPixbuf *pixbuffer, int w, int h, guchar R, guchar G,
       guchar B, guchar alpha)
276 {
277
278     int rowstride;
279     int channels;
280     guchar *pixels;
281     guchar *target;
282
283     channels = gdk_pixbuf_get_n_channels(pixbuffer);
284     rowstride = gdk_pixbuf_get_rowstride(pixbuffer); // define the hight
          position of a pixel.
285     pixels = gdk_pixbuf_get_pixels(pixbuffer);       // return a pointer
          to point the address of pixel data in buffer.
286
287     target = pixels + w * channels + h * rowstride; // find the pixel
          which need to be modified.
288     target[0] = R;
289     target[1] = G;
290     target[2] = B;
291
292 }
293
294 int main(int argc, char *argv[])
295 {
```

```
296     gtk_init(&argc, &argv);
297     window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
298     gtk_window_set_title(GTK_WINDOW(window), "BEST player");
299     g_signal_connect(window, "destroy", G_CALLBACK(closePlayer), NULL);
300
301     /* vertical boxes */
302     Vbox = gtk_vbox_new(FALSE, 0);
303     gtk_container_add(GTK_CONTAINER(window), Vbox);
304
305     /* create frame */
306     image_height = 360;
307     image_width = 480;
308     pixelBuffer = gdk_pixbuf_new(GDK_COLORSPACE_RGB, FALSE, 8, image_width
            , image_height);
309     frame = gtk_image_new_from_pixbuf(pixelBuffer);
310     gtk_box_pack_start(GTK_BOX(Vbox), frame, TRUE, TRUE, 0);
311
312     /* create Hbox for buttons */
313     Hbox = gtk_hbox_new(TRUE, 0);
314     gtk_box_pack_start(GTK_BOX(Vbox), Hbox, FALSE, FALSE, 0);
315
316     /* create Hbox2 for buttons */
317     Hbox2 = gtk_hbox_new(TRUE, 0);
318     gtk_box_pack_start(GTK_BOX(Vbox), Hbox2, FALSE, FALSE, 0);
319
320     pausePlaybtn = gtk_button_new_with_label("Pause");
321     // gtk_container_add(GTK_CONTAINER(Hbox), pausePlaybtn);
322     g_signal_connect(G_OBJECT(pausePlaybtn), "clicked", G_CALLBACK(
            pausePlayMode), NULL);
323     gtk_box_pack_start(GTK_BOX(Hbox), pausePlaybtn, TRUE, TRUE, 0);
324
325     repeatbtn = gtk_button_new_with_label("Replay");
326     // gtk_container_add(GTK_CONTAINER(Hbox), repeatbtn);
327     g_signal_connect(G_OBJECT(repeatbtn), "clicked", G_CALLBACK(repeatMode
            ), NULL);
328     gtk_box_pack_start(GTK_BOX(Hbox), repeatbtn, TRUE, TRUE, 0);
329
330     fastForwardbtn = gtk_button_new_with_label("Forward");
331     // gtk_container_add(GTK_CONTAINER(Hbox), fastForwardbtn);
332     g_signal_connect(G_OBJECT(fastForwardbtn), "clicked", G_CALLBACK(
            fastForwardMode), NULL);
333     gtk_box_pack_start(GTK_BOX(Hbox), fastForwardbtn, TRUE, TRUE, 0);
334
335     backwardbtn = gtk_button_new_with_label("Backward");
336     // gtk_container_add(GTK_CONTAINER(Hbox), backwardbtn);
```

```
337     g_signal_connect(G_OBJECT(backwardbtn), "clicked", G_CALLBACK(
            backwardMode), NULL);
338     gtk_box_pack_start(GTK_BOX(Hbox), backwardbtn, TRUE, TRUE, 0);
339
340     quitbtn = gtk_button_new_with_label("Quit");
341     // gtk_container_add(GTK_CONTAINER(Hbox), quitbtn);
342     g_signal_connect(G_OBJECT(quitbtn), "clicked", G_CALLBACK(quitMode),
            NULL);
343     gtk_box_pack_start(GTK_BOX(Hbox), quitbtn, TRUE, TRUE, 0);
344
345     /* initialize the fileChoosebtn  */
346     fileChoosebtn = gtk_button_new_with_label("ChooseFile");
347     g_signal_connect(G_OBJECT(fileChoosebtn), "clicked", G_CALLBACK(
            dialogStart), NULL);
348     gtk_box_pack_start(GTK_BOX(Hbox2), fileChoosebtn, TRUE, TRUE, 0);
349
350     /* enter bar and load button */
351     enter = gtk_entry_new();
352     gtk_box_pack_start(GTK_BOX(Hbox2), enter, TRUE, TRUE, 0);
353
354     gtk_widget_show_all(window);
355
356     /*-------GTK Initialization END-------*/
357
358     struct timeval start;
359     struct timeval end;
360     int timeDifference;
361
362     /* Play the movie */
363     while (quit == 0)
364     {
365
366         while ((movie != NULL) && (movie->currentFrame) && (quit == 0))
367         {
368
369             image_width = movie->currentFrame->RGBImage->W;
370             image_height = movie->currentFrame->RGBImage->H;
371             if (pausePlay == 0)
372             {
373                 gettimeofday(&start, NULL);
374                 /* Play */
375                 drawOneFrame(image_width, image_height);
376                 movie->currentFrame = movie->currentFrame->Next;
377                 /* Update the event */
378                 gtk_image_set_from_pixbuf(GTK_IMAGE(frame), pixelBuffer);
379
```

```
380                 gettimeofday(&end, NULL);
381                 timeDifference = (end.tv_sec * 1000000 + end.tv_usec) -
382                                  (start.tv_sec * 1000000 + start.tv_usec);
383                 //printf("FPS: %d\n",timeDifference);
384                 if (timeDifference > 0) {
385                     usleep(timeDifference);
386                 }
387             }
388             else
389             {
390                 usleep(33333);
391             }
392
393             while (gtk_events_pending())
394             {
395                 gtk_main_iteration();
396             }
397         }
398         usleep(33333);
399         if (gtk_events_pending())
400         {
401             gtk_main_iteration();
402         }
403     }
404     //printf("Player tool has been exited successfully !\n");
405     return 0;
406 }
407
408 /* Load one movie frame from the input file */
409 YUVIMAGE *LoadOneFrame(const char *fname, int n,
410                        unsigned int width, unsigned height)
411 {
412     FILE *file;
413     unsigned int x, y;
414     unsigned char c;
415     YUVIMAGE *YUVimage;
416
417     /* Check errors */
418     assert(fname);
419     assert(n >= 0);
420
421     YUVimage = CreateYUVImage(width, height);
422     if (YUVimage == NULL)
423     {
424         return NULL;
425     }
```

```
426
427     /* Open the input file */
428     file = fopen(fname, "r");
429     if (file == NULL)
430     {
431         DeleteYUVImage(YUVimage);
432         return NULL;
433     }
434
435     /* Find the desired frame */
436     fseek(file, 1.5 * n * width * height, SEEK_SET);
437
438     for (y = 0; y < height; y++)
439     {
440         for (x = 0; x < width; x++)
441         {
442             c = fgetc(file);
443             SetPixelY(YUVimage, x, y, c);
444         } /*rof*/
445     }
446
447     for (y = 0; y < height; y += 2)
448     {
449         for (x = 0; x < width; x += 2)
450         {
451             c = fgetc(file);
452             SetPixelU(YUVimage, x, y, c);
453             SetPixelU(YUVimage, x + 1, y, c);
454             SetPixelU(YUVimage, x, y + 1, c);
455             SetPixelU(YUVimage, x + 1, y + 1, c);
456         }
457     }
458
459     for (y = 0; y < height; y += 2)
460     {
461         for (x = 0; x < width; x += 2)
462         {
463             c = fgetc(file);
464             SetPixelV(YUVimage, x, y, c);
465             SetPixelV(YUVimage, x + 1, y, c);
466             SetPixelV(YUVimage, x, y + 1, c);
467             SetPixelV(YUVimage, x + 1, y + 1, c);
468         }
469     }
470
471     /* Check errors */
```

```
472     assert(ferror(file) == 0);
473
474     /* Close the input file and return */
475     fclose(file);
476     file = NULL;
477     return YUVimage;
478 }
479
480 /* Load the movie frames from the input file */
481 MOVIE *LoadMovie(const char *fname, int frameNum,
482                  unsigned int width, unsigned height)
483 {
484     MOVIE *movie;
485     movie = CreateMovie();
486
487     assert(movie);
488     int i = 0;
489     YUVIMAGE *yuvimage;
490
491     for (i = 0; i < frameNum; i++)
492     {
493         yuvimage = LoadOneFrame(fname, i, width, height);
494         AppendYUVImage(movie->Frames, yuvimage);
495     }
496     //printf("The movie file EngPlaza.yuv has been read successfully!\n");
497     return movie;
498 }
499
500 /* Save the movie frames to the output file */
501 int SaveMovie(const char *fname, MOVIE *movie)
502 {
503     FILE *outputfile;
504     outputfile = fopen(fname, "w");
505     int linklength = 0;
506     IENTRY *f, *l;
507     f = movie->Frames->First;
508     assert(f->YUVImage);
509     for (linklength = 0; linklength < movie->Frames->Length; linklength++)
510     {
511
512         l = f->Next;
513         SaveOneFrame(f->YUVImage, fname, outputfile);
514
515         f = l;
516     }
517
```

```
518     fclose(outputfile);
519     outputfile = NULL;
520     //printf("The movie file out.yuv has been written successfully!\n");
521     //printf("%d frames are written to the file out.yuv in total.\n",
            movie->Frames->Length);
522     return 0;
523 }
524
525 /* Saves one movie frame to the output file */
526 void SaveOneFrame(YUVIMAGE *image, const char *fname, FILE *file)
527 {
528     int x, y;
529     for (y = 0; y < image->H; y++)
530     {
531         for (x = 0; x < image->W; x++)
532         {
533             fputc(GetPixelY(image, x, y), file);
534         }
535     }
536
537     for (y = 0; y < image->H; y += 2)
538     {
539         for (x = 0; x < image->W; x += 2)
540         {
541             fputc(GetPixelU(image, x, y), file);
542         }
543     }
544
545     for (y = 0; y < image->H; y += 2)
546     {
547         for (x = 0; x < image->W; x += 2)
548         {
549             fputc(GetPixelV(image, x, y), file);
550         }
551     }
552 }
553
554 /* This method will get the width and height of a video by checking the
       key words in the name of the file. */
555 void get_Width_And_Height(char name[], int widthAndHeight[2])
556 {
557     int i = 0;
558     int start = -1;
559     int end = -1;
560     char chars[256];
561     memset(chars, 0, sizeof(chars));
```

```
562     for (i = 0; i < strlen(name); i++)
563     {
564         if (name[i] == '_')
565         {
566             if ((start > 0) && (start < i))
567             {
568                 start = start;
569             }
570             else
571             {
572                 start = i;
573             }
574         }
575         else if (name[i] == '.')
576         {
577             end = i;
578         }
579     }
580
581     if (start < 0)
582     {
583         widthAndHeight[0] = 0;
584         widthAndHeight[1] = 0;
585     }
586     else
587     {
588         strncpy(chars, name + start, end - start);
589         sscanf(chars, "_%d_%d", widthAndHeight, widthAndHeight + 1);
590     }
591 }
```