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 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

1 Introduction 1

2 Method and Procedure 1
   2.1 General Design ................................................................. 1
   2.2 Data Structure ................................................................. 2
       2.2.1 Pixel Buffer ............................................................. 3
   2.3 Control Flow ................................................................. 3
   2.4 Player Functions ............................................................. 3
       2.4.1 Pause ................................................................. 4
       2.4.2 Replay ................................................................. 4
       2.4.3 Fast-Forward .......................................................... 4
       2.4.4 Backward ............................................................... 4
       2.4.5 File Choose ............................................................ 4
       2.4.6 Quit ................................................................. 4

3 User Manual 4

4 Conclusion and Future Work 5

References 5

5 Appendix 6
   5.1 Source Code ................................................................. 6
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YUV video structure</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>data structure</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Choose File</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Choose File</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Play Video Successfully</td>
<td>5</td>
</tr>
</tbody>
</table>
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

![Data Structure Diagram](image)

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](image-url)
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”.

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.

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


5 Appendix

The Appendix section will show the BEST Player code section.

5.1 Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include <string.h>
#include <assert.h>
#include <time.h>
#include <math.h>
#include <sys/time.h>

#include "Movie.h"
#include "Constants.h"
#include "Image.h"
#include "ImageList.h"
#include "FileIO.h"

/* 0 is play, 1 is pause */
int pausePlay = 0;
/* 0 do nothing, 1 repeat */
int repeat = 0;

int image_width = 0;
int image_height = 0;
int quit = 0;
int fps = 33000;

GtkWidget *pausePlaybtn;
GtkWidget *repeatbtn;
GtkWidget *fastForwardbtn;
GtkWidget *backwardbtn;
GtkWidget *window;
GdkPixbuf *pixelBuffer;
GtkWidget *frame;
GtkWidget *quitbtn;
GtkWidget *loadbtn, *enter;
GtkWidget *fileChoosebtn;
GtkWidget *FPSbutton;
MOVIE *movie;

GtkWidget *Vbox, *Hbox, *Hbox2;
```
/* Load one movie frame from the input file */
YUVIMAGE *LoadOneFrame(const char *fname, int n,
                        unsigned int width, unsigned height);

/* Load the movie frames from the input file */
MOVIE *LoadMovie(const char *fname, int frameNum,
                 unsigned int width, unsigned height);

/* Saves one movie frame to the output file */
void SaveOneFrame(YUVIMAGE *image, const char *fname, FILE *file);

/* Save the movie frames to the output file */
int SaveMovie(const char *fname, MOVIE *movie);

/* Set the RGB in each pixel of pixel buffer */
void set_pixel(GdkPixbuf *pixbuffer, int w, int h, uchar R, uchar G,
               uchar B, uchar alpha);

/* Set the pixel in one frame */
void drawOneFrame(int image_width, int image_height);

/* Pause the video */
void pausePlayMode(GtkWidget *Widget, gpointer Data);

/* Repeat the video and reset the pointer */
void repeatMode(GtkWidget *Widget, gpointer Data);

/* Adjust the pointer and fast-forward the video */
void fastForwardMode(GtkWidget *Widget, gpointer Data);

/* Adjust the pointer and backward the video */
void backwardMode(GtkWidget *Widget, gpointer Data);

/* Exit this tool */
void quitMode(GtkWidget *Widget, gpointer Data);

/* Load the movie */
void loadMovie(GtkWidget *Widget, gpointer Data);

/* Generate the dialog windows */
void dialogStart(GtkWidget *Widget, gpointer Data);

/* Get the width and height from the name of the video file */
void get_Width_And_Height(char name[], int widthAndHeight[]);

/* Exit the tool */
void closePlayer(GtkWidget *Widget, gpointer Data);

/* 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. */
void dialogStart(GtkWidget *Widget, gpointer Data)
{
  gint response;
  GtkWidget *dialog;
  char *fileName;
  const char *fileNameCst;
  FILE *file;
  int widthAndHeight[2];
  int fileSize;
  int numOfFrame;
  char totalNum[256];

  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);
  GtkWidgetFilter *filter = gtk_file_filter_new();
  gtk_file_filter_add_pattern(filter, "*.yuv");
  gtk_file_chooser_add_filter(GTK_FILE_CHOOSER(dialog), filter);

  gtk_widget_show_all(dialog);
  response = gtk_dialog_run(GTK_DIALOG(dialog));
  if (response == GTK_RESPONSE_ACCEPT)
  {
    fileName = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog))
      ;

    /* get the width and height of the video */
    memset(widthAndHeight, 0, sizeof(widthAndHeight));
    get_Width_And_Height(fileName, widthAndHeight);
    // printf("%d, %d\n", widthAndHeight[0], widthAndHeight[1]);

    fileNameCst = fileName;
    /* get the size of the video */
    file = fopen(fileNameCst, "r");
    if (file == NULL)
    {
      gtk_entry_set_text(GTK_ENTRY(enter), "File Error");
gtk_widget_destroy(dialog);
return;
}
fsseek(file, 0, SEEK_END);
fileSize = ftell(file);
// printf("filesize: %d\n", fileSize);

/* calculate the number of frame */
if (widthAndHeight[0] * widthAndHeight[1] != 0)
{
    numOfFrame = fileSize / (widthAndHeight[0] * widthAndHeight[1] * 1.5);
}
else
{
    gtk_entry_set_text(GTK_ENTRY(enter), "Frame Size Error");
gtk_widget_destroy(dialog);
    return;
}
if (numOfFrame != (int)numOfFrame)
{
/* incorrect the size of the frame */
    gtk_entry_set_text(GTK_ENTRY(enter), "Frame Size Error");
    gtk_widget_destroy(dialog);
    return;
}
// printf("numOfFrame: %d\n", numOfFrame);

movie = LoadMovie(fileNameCst, numOfFrame, widthAndHeight[0],
        widthAndHeight[1]);
YUV2RGBMovie(movie);
movie->currentFrame = movie->Frames->First;
// printf("Load success\n");

/* update the size of the frame */
pixelBuffer = gdk_pixbuf_new(GDK_COLORSPACE_RGB, FALSE, 8,
        widthAndHeight[0], widthAndHeight[1]);
memset(totalNum, 0, sizeof(totalNum));
sprintf(totalNum, "Total Frames: %d", numOfFrame);
gtk_entry_set_text(GTK_ENTRY(enter), totalNum);

/* close the file after loading */
fclose(file);
file = NULL;
// printf("out success\n");
else
{
    gtk_entry_set_text(GTK_ENTRY(entry), "Please choose a file!");
}

gtk_widget_destroy(dialog);
/

*/ 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. */
void drawOneFrame(int image_width, int image_height)
{
    int w = 0;
    int h = 0;
    for (w = 0; w < image_width; w++)
    {
        for (h = 0; h < image_height; h++)
        {
            set_pixel(pixelBuffer, w, h, GetPixelR(movie->currentFrame->RGBImage, w, h), GetPixelG(movie->currentFrame->RGBImage, w, h), GetPixelB(movie->currentFrame->RGBImage, w, h), 0);
        }
    }

} /* This method will change the status of video to pause or play the video.

void pausePlayMode(GtkWidget *Widget, gpointer Data)
{
    if (pausePlay == 0)
    {
        gtk_button_set_label(GTK_BUTTON(pausePlayBttn), "Play");
        pausePlay = 1;
    }
    else
    {
        gtk_button_set_label(GTK_BUTTON(pausePlayBttn), "Pause");
        pausePlay = 0;
    }

} /* By changing the current pointer, the first frame of this video will be set. */
void repeatMode(GtkWidget *Widget, gpointer Data)
{
if (movie != NULL) {
    movie->currentFrame = movie->Frames->First;
    drawOneFrame(image_width, image_height);
    repeat = 1;
    gtk_image_set_from_pixbuf(GTK_IMAGE(frame), pixelBuffer);
}

/* This method will use loop to achieve the fast forward function. */
void fastForwardMode(GtkWidget *Widget, gpointer Data) {
    int i = 0;
    // printf("enter\n");
    while ((movie != NULL) && (movie->currentFrame != NULL) && (movie->currentFrame->Next != NULL) && (i < 5)) {
        movie->currentFrame = movie->currentFrame->Next;
        i ++;
    }
    // printf("out\n");
    if ((movie != NULL) && (movie->currentFrame != NULL)) {
        drawOneFrame(image_width, image_height);
        gtk_image_set_from_pixbuf(GTK_IMAGE(frame), pixelBuffer);
    }
}

/* This method will use loop to achieve the backward function. */
void backwardMode(GtkWidget *Widget, gpointer Data) {
    int i = 0;
    while ((movie != NULL) && (movie->currentFrame != NULL) && (movie->currentFrame->Prev) && (i < 10)) {
        movie->currentFrame = movie->currentFrame->Prev;
        i ++;
    }
    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     // printf("close player\n");
270     quit = 1;
271 }
272
273 /* The real method will assign each pixel's RGB value */
274 void set_pixel(GdkPixbuf *pixbuffer, int w, int h, guchar R, guchar G,
275                 guchar B, guchar alpha)
276 {
277     int rowstride;
278     int channels;
279     guchar *pixels;
280     guchar *target;
281     channels = gdk_pixbuf_get_n_channels(pixbuffer);
282     rowstride = gdk_pixbuf_get_rowstride(pixbuffer); // define the height
283                 // position of a pixel.
284     pixels = gdk_pixbuf_get_pixels(pixbuffer); // return a pointer
285                 // to point the address of pixel data in buffer.
286     target = pixels + w * channels + h * rowstride; // find the pixel
287                 // 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 {
gtk_init(&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(window), "BEST player");
g_signal_connect(window, "destroy", G_CALLBACK(closePlayer), NULL);

/* vertical boxes */
Vbox = gtk_vbox_new(FALSE, 0);
gtk_container_add(GTK_CONTAINER(window), Vbox);

/* create frame */
image_height = 360;
image_width = 480;
pixelBuffer = gdk_pixbuf_new(GDK_COLORSPACE_RGB, FALSE, 8, image_width , image_height);
frame = gtk_image_new_from_pixbuf(pixelBuffer);
gtk_box_pack_start(GTK_BOX(Vbox), frame, TRUE, TRUE, 0);

/* create Hbox for buttons */
Hbox = gtk_hbox_new(TRUE, 0);
gtk_box_pack_start(GTK_BOX(Vbox), Hbox, FALSE, FALSE, 0);

/* create Hbox2 for buttons */
Hbox2 = gtk_hbox_new(TRUE, 0);
gtk_box_pack_start(GTK_BOX(Vbox), Hbox2, FALSE, FALSE, 0);

pausePlaybtn = gtk_button_new_with_label("Pause");
// gtk_container_add(GTK_CONTAINER(Hbox), pausePlaybtn);
g_signal_connect(G_OBJECT(pausePlaybtn), "clicked", G_CALLBACK( 
pausePlayMode), NULL);
gtk_box_pack_start(GTK_BOX(Hbox), pausePlaybtn, TRUE, TRUE, 0);

repeatbtn = gtk_button_new_with_label("Replay");
// gtk_container_add(GTK_CONTAINER(Hbox), repeatbtn);
g_signal_connect(G_OBJECT(repeatbtn), "clicked", G_CALLBACK( repeatMode ), NULL);
gtk_box_pack_start(GTK_BOX(Hbox), repeatbtn, TRUE, TRUE, 0);

fastForwardbtn = gtk_button_new_with_label("Forward");
// gtk_container_add(GTK_CONTAINER(Hbox), fastForwardbtn);
g_signal_connect(G_OBJECT(fastForwardbtn), "clicked", G_CALLBACK( fastForwardMode), NULL);
gtk_box_pack_start(GTK_BOX(Hbox), fastForwardbtn, TRUE, TRUE, 0);

backwardbtn = gtk_button_new_with_label("Backward");
// gtk_container_add(GTK_CONTAINER(Hbox), backwardbtn);
```c
337 g_signal_connect(G_OBJECT(backwardbtn), "clicked", G_CALLBACK(
338       backwardMode), NULL);
339 gtk_box_pack_start(GTK_BOX(Hbox), backwardbtn, TRUE, TRUE, 0);
340
341 quitbtn = gtk_button_new_with_label("Quit");
342 // gtk_container_add(GTK_CONTAINER(Hbox), quitbtn);
343 g_signal_connect(G_OBJECT(quitbtn), "clicked", G_CALLBACK(quitMode),
344       NULL);
345 gtk_box_pack_start(GTK_BOX(Hbox), quitbtn, TRUE, TRUE, 0);
346
347 /* initialize the fileChoosebtn */
348 fileChoosebtn = gtk_button_new_with_label("ChooseFile");
349 g_signal_connect(G_OBJECT(fileChoosebtn), "clicked", G_CALLBACK(
350       dialogStart), NULL);
351 gtk_box_pack_start(GTK_BOX(Hbox2), fileChoosebtn, TRUE, TRUE, 0);
352
353 /* enter bar and load button */
354 enter = gtk_entry_new();
355 gtk_box_pack_start(GTK_BOX(Hbox2), enter, TRUE, TRUE, 0);
356
357 gtk_widget_show_all(window);
358
359 /**********GTK Initialization END**********/
360
361 struct timeval start;
362 struct timeval end;
363 int timeDifference;
364
365 /* Play the movie */
366 while (quit == 0)
367 {
368     while ((movie != NULL) && (movie->currentFrame) && (quit == 0))
369     {
370         image_width = movie->currentFrame->RGBImage->W;
371         image_height = movie->currentFrame->RGBImage->H;
372         if (pausePlay == 0)
373             {
374                 gettimeofday(&start, NULL);
375                 /* Play */
376                 drawOneFrame(image_width, image_height);
377                 movie->currentFrame = movie->currentFrame->Next;
378                 /* Update the event */
379                 gtk_image_set_from_pixbuf(GTK_IMAGE(frame), pixelBuffer);
380             }
381     }
382 }
383 ```
gettimeofday(&end, NULL);

timeDifference = (end.tv_sec * 1000000 + end.tv_usec) -
                      (start.tv_sec * 1000000 + start.tv_usec);

  // printf("FPS: %d\n", timeDifference);
  if (timeDifference > 0) {
      usleep(timeDifference);
  }
  else
  {
      usleep(33333);
  }

while (gtk_events_pending())
{
    gtk_main_iteration();
}

usleep(33333);
if (gtk_events_pending())
{
    gtk_main_iteration();
}

  // printf("Player tool has been exited successfully !\n");
  return 0;

/* Load one movie frame from the input file */
YUVIMAGE *LoadOneFrame(const char *fname, int n,
                      unsigned int width, unsigned height)
{
    FILE *file;
    unsigned int x, y;
    unsigned char c;
    YUVIMAGE *YUVimage;

    /* Check errors */
    assert(fname);
    assert(n >= 0);

    YUVimage = CreateYUVImage(width, height);
    if (YUVimage == NULL)
    {
        return NULL;
    }
/* Open the input file */
file = fopen(fname, "r");
if (file == NULL)
{
    DeleteYUVImage(YUVimage);
    return NULL;
}

/* Find the desired frame */
seek(file, 1.5 * n * width * height, SEEK_SET);
for (y = 0; y < height; y++)
{
    for (x = 0; x < width; x++)
    {
        c = fgetc(file);
        SetPixelY(YUVimage, x, y, c);
    } /* rof */
}
for (y = 0; y < height; y += 2)
{
    for (x = 0; x < width; x += 2)
    {
        c = fgetc(file);
        SetPixelU(YUVimage, x, y, c);
        SetPixelU(YUVimage, x + 1, y, c);
        SetPixelU(YUVimage, x, y + 1, c);
        SetPixelU(YUVimage, x + 1, y + 1, c);
    }
}
for (y = 0; y < height; y += 2)
{
    for (x = 0; x < width; x += 2)
    {
        c = fgetc(file);
        SetPixelV(YUVimage, x, y, c);
        SetPixelV(YUVimage, x + 1, y, c);
        SetPixelV(YUVimage, x, y + 1, c);
        SetPixelV(YUVimage, x + 1, y + 1, c);
    }
}
/* Check errors */
assert(err(file) == 0);

/* Close the input file and return */
fclose(file);
file = NULL;
return YUVimage;
}

/* Load the movie frames from the input file */
MOVIE *LoadMovie(const char *fname, int frameNum, unsigned int width, unsigned height)
{
    MOVIE *movie;
    movie = CreateMovie();
    assert(movie);
    int i = 0;
    YUVIMAGE *yuvimage;

    for (i = 0; i < frameNum; i++)
    {
        yuvimage = LoadOneFrame(fname, i, width, height);
        AppendYUVImage(movie->Frames, yuvimage);
    }
    // printf("The movie file EngPlaza.yuv has been read successfully!\n");
    return movie;
}

/* Save the movie frames to the output file */
int SaveMovie(const char *fname, MOVIE *movie)
{
    FILE *outputfile;
    outputfile = fopen(fname, "w");
    int linklength = 0;
    IENTRY *f, *l;
    f = movie->Frames->First;
    assert(f->YUVImage);
    for (linklength = 0; linklength < movie->Frames->Length; linklength++)
    {
        l = f->Next;
        SaveOneFrame(f->YUVImage, fname, outputfile);
        f = l;
    }

fclose(outputfile);
outputfile = NULL;

// printf("The movie file out.yuv has been written successfully!\n");
// printf("%d frames are written to the file out.yuv in total.\n", movie->Frames->Length);
return 0;
}

/* Saves one movie frame to the output file */
void SaveOneFrame(YUVIMAGE *image, const char *fname, FILE *file)
{
    int x, y;
    for (y = 0; y < image->H; y++)
    {
        for (x = 0; x < image->W; x++)
        {
            fputc(GetPixelY(image, x, y), file);
        }
    }

    for (y = 0; y < image->H; y += 2)
    {
        for (x = 0; x < image->W; x += 2)
        {
            fputc(GetPixelU(image, x, y), file);
        }
    }

    for (y = 0; y < image->H; y += 2)
    {
        for (x = 0; x < image->W; x += 2)
        {
            fputc(GetPixelV(image, x, y), file);
        }
    }

    /* This method will get the width and height of a video by checking the key words in the name of the file. */
void get_Width_And_Height(char name[], int widthAndHeight[2])
{
    int i = 0;
    int start = -1;
    int end = -1;
    char chars[256];
    memset(chars, 0, sizeof(chars));
for (i = 0; i < strlen(name); i++)
{
    if (name[i] == '_')
    {
        if ((start > 0) && (start < i))
        {
            start = start;
        }
        else
        {
            start = i;
        }
    }
    else if (name[i] == '.')
    {
        end = i;
    }
}
if (start < 0)
{
    widthAndHeight[0] = 0;
    widthAndHeight[1] = 0;
}
else
{
    strncpy(chars, name + start, end - start);
    sscanf(chars, "_%d_%d", widthAndHeight, widthAndHeight + 1);
}