Student Name: _____ Student No: _____

The Hong Kong Polytechnic University
Department of Electronic & Information Engineering
Laboratory 1: Image and Video Processing Using C Language

Objective:

The objectives of this laboratory are to address some issues in processing and displaying images and videos. A basic program is provided to process and display images and videos. You are required to read, modify and display the images, and implement the video play back and random access, using C language.

Required software:

Integrated Development environment (IDE): *Microsoft Visual Studio 2015* or *2017*
Note: It should be also workable with *Microsoft Visual Studio 2008*, *2010*, *2012*, and *2013*, but the procedure steps are slightly different when creating the project.

Required media clips:

YUV 4:2:0 Raw image: *FourPeople_1280x720.raw*
YUV 4:2:0 Raw video: *BasketballPass_416x240_50.yuv*
(Required media clips are zipped into "*ImageVideoProcessingLab.zip*" which can be downloaded from website: http://www.eie.polyu.edu.hk/~ylchan/course.htm)

Please go to the lecture notes and appendix if you are not familiar with the image format.

Submission:

Demonstrations to the lab tutor are required for this laboratory exercise.
This laboratory sheet together with the answers should be submitted to the lab tutor before the end of the lab session.

Procedures:

Section A. Reading and Displaying Image

In this section, we are going to read a YUV 4:2:0 raw image from the hard drive according to the parameters in the configuration file, then display the image. Below is the conceptual block diagram for reading and displaying the image:

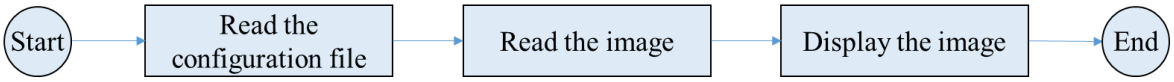

Fig 1. Block diagram for reading and displaying the image.

1. Study the configuration file

   Download the zip file "*ImageVideoProcessingLab.zip*" from website: http://www.eie.polyu.edu.hk/~ylchan/course.htm. Decompress it. Study the configuration file "*image.cfg*". You can simply open it by any text editors such as "*Notepad*". The parameters are listed row by row as follows, please fill in the table below:

Row 1) Name of input file	
Row 2) Width	
Row 3) Height	

2. Create a new project

   For Visual Studio 2015:

   a) Open the visual studio 2015 IDE. Choose "File" > "New" > "Project…" from the menu bar.
   b) At the left-hand side of the pop-up window, choose "Installed" > "Template" > "Visual C++" > "General".
   c) At the middle part, choose "Empty Project".
   d) At the bottom part, for the text field "Name", type "RawDisplay" for the project name. For the text field "Location", choose a proper directory for your convenience.
   e) Click "OK".

   You have created a new empty project successfully.

   For Visual Studio 2017:

   a) Open the visual studio 2017 IDE. Choose "File" > "New" > "Project…" from the menu bar.
   b) At the left-hand side of the pop-up window, choose "Installed" > "Visual C++" > "General".
   c) At the middle part, choose "Empty Project".
   d) At the bottom part, for the text field "Name", type "RawDisplay" for the project name. For the text field "Location", choose a proper directory for your convenience.
   e) Click "OK".

   You have created a new empty project successfully.

3. Add the files

a) Copy the source file "*RawDisplay.c*", the configuration file "*image.cfg*", the raw image file "*FourPeople_1280x720.raw*" that just decompressed from the zip file at step 1, to the project directory "RawDisplay\RawDisplay".

b) Go back to IDE, right click the "Source Files" from the "Solution Explorer". Choose "Add" > "Existing Item…", and click to add the source file "*RawDisplay.c*".

c) At the "Solution Explorer", double click the "*RawDisplay.c*".

Source codes in "*RawDisplay.c*" should be shown.

4. Run the program

a) Choose "Build" > "Build Solution" from the menu bar. Build should be succeeded.

b) Choose "Debug" > "Start Without Debugging" from the menu bar to run the program.

The raw image mentioned in "*image.cfg*" should be displayed in a pop-up window.

5. Understand the code

Study the ***main*** function for understanding the codes. (As our main objective is to understand the formation of the raw image file, we would only focus on the manipulation related to image file, rather than the window operation such as how to create a window and the message loop handling. But if you are interested, you can visit the Microsoft MSDN website for more information.)

```
// Main function
int main()
{
	char szKey;
	FILE *fpInputFile;
	int iWidth, iHeight, iFrameRate, iMaxFrameNum;
	unsigned char * pYuvBuf, * pBmpBuf;

	// init ////////////////////////////////////////////////////////////////
	// read parameter file
	if(readImageConfigFile("image.cfg", &fpInputFile, &iWidth, &iHeight) == -1)
		return -1;
	// init display and thread stuff
	if(initDisplayThread(iWidth, iHeight, &pYuvBuf, &pBmpBuf) == -1)
		return -1;

	// read ////////////////////////////////////////////////////////////////
	// display the image
```

```
        displayImage(fpInputFile, iWidth, iHeight, pYuvBuf, pBmpBuf);

        // close ///////////////////////////////////////////////////////////////////
        //printf(" Please press [Enter] to close the window.\n");
        getchar();

        freeBmp();
        if (pYuvBuf)
                free(pYuvBuf);
        if (pBmpBuf)
                free(pBmpBuf);
        if (fpInputFile)
                fclose(fpInputFile);

        return 0;
}
```

a) First of all, ***readImageConfigFile*** is the function to read the parameters from "*image.cfg*" and return the file pointer for the image ***fpInputFile***, the image width ***iWidth***, and the image height ***iHeight***.

b) Then, ***initDisplayThread*** is the function to allocate the memory buffer to store the pixels for the purpose of displaying the image. Therefore, this function returns the pointer for YUV buffer ***pYuvBuf***, and pointer for flipped version RGB buffer ***pBmpBuf***. The YUV buffer is for reading the Y, U, and V components from the image file. After conversion from YUV to flipped RGB, the flipped version RGB buffer is for storing the flipped RGB for display. Moreover, this function also creates a window, message loop, and thread, etc.

c) After all initialization, ***displayImage*** is the function to display the image. If we dive deeper into the ***displayImage*** function, we can see that it reads the pixels from the image file ***fp*** to the YUV buffer ***pYuvBuf*** by using the basic I/O function ***fread***. Then ***yuv2bmp*** function is called to convert and flip the YUV buffer ***pYuvBuf*** to the flipped version RGB buffer ***pBmpBuf***, then display the image at the window.

d) After displaying the image, some memory free and file close activities are done at the end of the ***main*** function.

6. Backup the project

Backup your project (Save a new copy) before continuing the following Sections.

Section B. Image Processing and Storing Image

In this section, we are going to perform image processing and store the modified YUV 4:2:0 raw image to the hard drive. Below is the conceptual block diagram for modifying and storing the image:

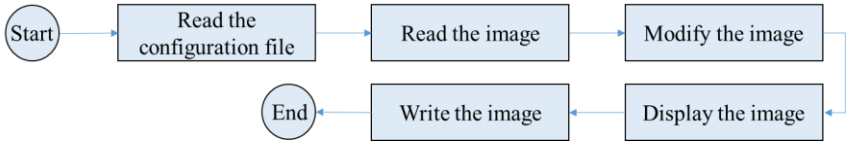

Fig 2. Block diagram for modifying and writing the image.

1. Study the raw image file format structure

   Before modifying and storing the image, understanding of the raw image file format structure is needed. The raw image *FourPeople_1280x720.raw* has the color format of **YUV 4:2:0**. Calculate the sizes of the Y, U and V components by filling in the following table.

   (Please read the lecture notes and appendix if you are not clear about the color format.)

a) Size of Y component (in bytes)	
b) Size of U component (in bytes)	
c) Size of V component (in bytes)	
d) Total Size = Size of Y component + Size of U component + Size of V component (in bytes)	

   e) Check the properties of the raw video file. Check the file size (Not the "Size on disk" but "Size") of the raw image *FourPeople_1280x720.raw*, is it the same as the total size?

   _____

2. Image processing: gray-scale image

   a) Go to the ***main*** function, replace the ***displayImage*** function by the ***displayGrayScaleImage*** function.

   b) Build and run the program. (Make sure you have closed the program.) A gray-scale image should be displayed and it is stored as "*grayscale.raw"*.

   c) Study the ***displayGrayScaleImage*** function. You can see that there is one for-loop to set U and V components to value of 128 to make the image gray-scale before displaying and writing out as "*grayscale.raw*".

3. Image processing: flipped image

   a) Go to the ***main*** function, replace ***displayGrayScaleImage*** function by ***displayFlipImage*** function.

   b) Build and run the program. (Make sure you have closed the program.) A flipped image should be displayed and it is written out as "*flip.raw"*.

   c) Look into the ***displayFlipImage*** function. You can see that there are three for-loops to flip

Y, U and V components of the image before displaying and writing out as "*flip.raw*".

4. Demonstrations

Tasks	Check by tutors
Display the flipped image (by running the program)	

5. Backup the project

   Backup your project (Save a new copy) before continuing the following Sections.

6. Wrap up

   At this step, you should already know the basic image processing techniques. Actually all the image processing tools such as cropping, contrast adjustment, white balancing, and filters, are working in the same manner inside the core by manipulating the pixels.

Section C. Video Playback

In this section, we are going to read and display the raw video from the hard drive. Below is the conceptual block diagram for reading and displaying the video:

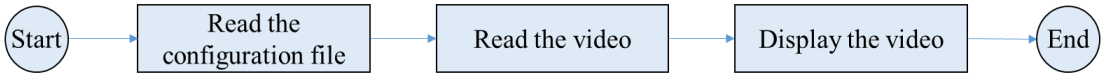

Fig 3. Block diagram for reading and displaying the video.

1. Study the configuration file

   Study the configuration file "*video.cfg*". You can simply open it by any text editors such as "*Notepad*". The parameters are listed row by row as follows, please fill in the table below:

Row 1) Name of input file	
Row 2) Width	
Row 3) Height	
Row 4) Frame Rate	

2. Study the raw video file format structure

   A raw video is a sequence of raw images (frames) concatenated. The raw video *BasketballPass_416x240_50.yuv* has the color format of **YUV 4:2:0**. Calculate the sizes of the Y, U and V components by filling in the following table.

a) Size of Y component for one frame (in bytes)	
b) Size of U component for one frame (in bytes)	
c) Size of V component for one frame (in bytes)	
d) Size for one frame = Size of Y component + Size of U component + Size of V component (in bytes)	

e) Check the properties of the raw video file *BasketballPass_416x240_50.yuv*. What is the total file size (Not the "Size on disk" but "Size") of the raw video in bytes?

---

f) Based on the above, compute the total number of frames in *BasketballPass_416x240_50.yuv*?

---

3. Add the files

Copy the configuration file "*video.cfg*", and the raw image file "*BasketballPass_416x240_50.yuv*" to the project directory "RawDisplay\RawDisplay".

4. Play the video

a) Go to the ***main*** function, replace the ***readImageConfigFile*** function by the ***readVideoConfigFile*** function as follows:

```
…
// init ////////////////////////////////////////////////////////////////////////////////////
// read parameter file
//if(readImageConfigFile("image.cfg", &fpInputFile, &iWidth, &iHeight) == -1)
//	return -1;
if(readVideoConfigFile("video.cfg", &fpInputFile, &iWidth, &iHeight, &iFrameRate,
&iMaxFrameNum) == -1)
	return -1;
…
```

The ***readVideoConfigFile*** is the function to read the parameters from "*video.cfg*" and return the file pointer for the image ***fpInputFile***, the image width ***iWidth***, the image height ***iHeight***, the frame rate ***iFrameRate***, and the maximum frame number ***iMaxFrameNum***.

b) Go to the ***main*** function, replace the ***displayImage*** function by the ***playVideo*** function as follows:

```
…
// read ////////////////////////////////////////////////////////////////////////////////////
// display YUV
//displayImage(fpInputFile, iWidth, iHeight, pYuvBuf, pBmpBuf);
playVideo(fpInputFile, iWidth, iHeight, iFrameRate, iMaxFrameNum, pYuvBuf, pBmpBuf);
…
```

c) The ***playVideo*** is the function to play the raw video. In the ***playVideo*** function, there is a for-loop to display the video frame-by-frame with the ***Sleep*** function*. Build and run the program. Write down your observation, and the use of the ***Sleep*** function.
*The details of the ***Sleep*** function can be found in appendix.

d) Change ***Sleep(100)*** to ***Sleep(10)***, build and run the program. Highlight the difference compared with the case of ***Sleep(100)***.

5. Play the video at controlled frame rate

To have precise control of the video playback rate, several factors, which affects the frame rate during playback of the video, are taken into consideration. They include:

(i) Time to load the video frame from the source (file / network), t_{source}.

(ii) Time to pre-process the image (e.g. conversion of image format, YUV4:2:0 to RGB 24-bit, decompress the image, etc), $t_{process}$.

(iii) Time to display the image, $t_{display}$.

(iv) Time period to display the next video frame, T.

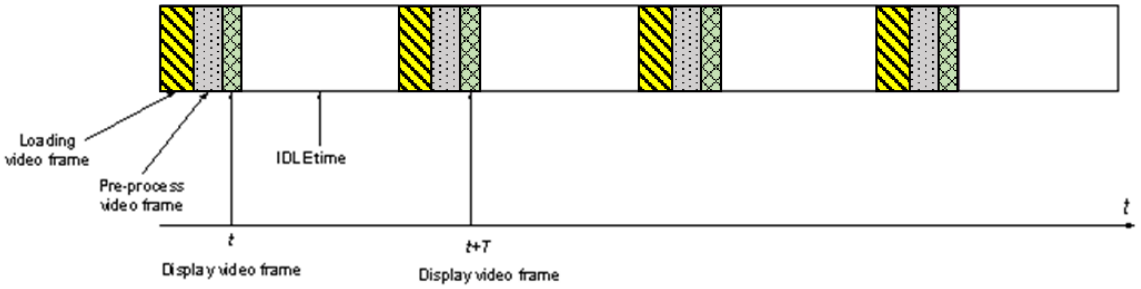

Fig 4. Illustration for displaying the video with controlled frame rate by inserting an idle time.

The above diagram illustrates the sequence of displaying a video frame with frame rate controlled by inserting an idle time between two consecutive video frames. The idle time should be varied for each video frame to adjust the actual display rate of the video playback. The idle time is calculated using the following equation:

$$t_{IDLE} = \frac{1}{frame\ rate} - t_{source} - t_{process} - t_{display}$$

In the above equation, the idle time is the remaining time of the interval between two consecutive video frames (*1/frame rate*) after spending on t_{source}, $t_{process}$ and $t_{display}$, where t_{IDLE} is the idle time for the current video frame, t_{source} is the time to load the video frame from source, $t_{process}$ is the time to process the video frame before being displayed and $t_{display}$ is the time used to display the video frame.

Since the video frame interval time is fixed and depends on *1/frame rate*. t_{source}, $t_{process}$ and $t_{display}$ can be measured. Then the idle time t_{IDLE} can be easily calculated. The following codes show how to estimate the time being spent in t_{source}, $t_{process}$ and $t_{display}$, and the computation of the idle time t_{IDLE}. As shown in the codes, the ***clock()*** function returns the relative system time in millisecond which will be used as a reference for measuring the time before and after the processes of t_{source}, $t_{process}$ and $t_{display}$. Then we can obtain the idle time by computing the difference of the fixed video frame interval time (***1/frame rate***) and ***timeElapse*** by t_{source}, $t_{process}$ and $t_{display}$.

a) Go to the ***playVideo*** function, replace the codes as follows, and fill in the new parameter in the ***Sleep*** function:

```
void playVideo(FILE * fp, int iWidth, int iHeight, int iFrameRate, int iMaxFrameNum, unsigned char *
pYuvBuf, unsigned char * pBmpBuf)
{
	int i;
	for (i = 0; i<iMaxFrameNum; i++) // loop each frame
	{
		long timeElapse, t1;
		t1 = clock();
		// read YUV
		fread(pYuvBuf, sizeof(unsigned char), iWidth * iHeight * 3 / 2, fp);
		// convert and display
		yuv2bmp(pYuvBuf, pBmpBuf, iWidth, iHeight);

		timeElapse = clock() - t1;
		Sleep(_____);
	}

	printf("\n Finished playback.\n");
}
```

b) Run the program. Right now, the video should have the playback in a controlled frame rate.

6. <u>Backup the project</u>

Backup your project (Save a new copy) before continuing the following Sections.

Section D. VCR Functionalities

To enrich the viewers' experience, interactive video player is desired by adding VCR (Video Cassette Recording) functionalities such as random access, fast forward, stop and pause. In this section, we are going to add some simple VCR functionalities with menu as follows:

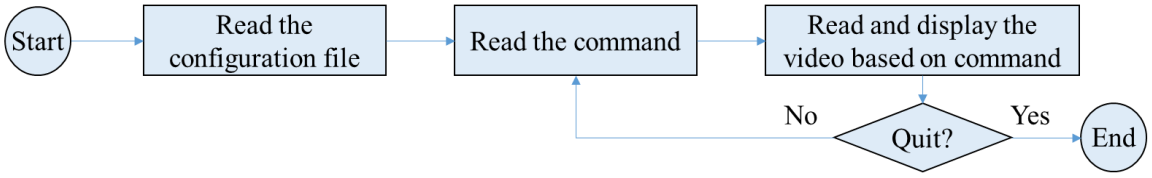

Fig 5. Block diagram for reading and displaying the video based on user commands.

1. Add the user menu to play and quit

   a) Go to the ***main*** function, replace the ***playVideo*** function by the codes as follows:

```
…
// read ////////////////////////////////////////////////////////////////////////////////////////
// display the first frame
//playVideo(fpInputFile, iWidth, iHeight, iFrameRate, iMaxFrameNum, pYuvBuf, pBmpBuf);
displayImage(fpInputFile, iWidth, iHeight, pYuvBuf, pBmpBuf);
// menu
while (1)
{
        // user input
        printf(" Menu: [p] play, [q] quit\n");
        printf(" Please enter the [player command]  : ");
        scanf(" %c", &szKey);
        // player action
        if (szKey == 'p')
        {
                playVideo(fpInputFile, iWidth, iHeight, iFrameRate, iMaxFrameNum, pYuvBuf,
pBmpBuf);
                fseek(fpInputFile, 0, SEEK_SET);
        }
        else if (szKey == 'q')
                break;
        else
                printf("\n Incorrect player command.\n\n");
        printf("\n");
}
…
```

Since we allow the user to have playback again, we need to reset the file pointer to make it point to the beginning of the raw video file after the playback. The basic I/O function ***fseek*** helps to do that.

The details of the **fseek function can be found in the appendix.*

b) Run the program. You can enter "p" to play the video, or "q" to quit the program.

2. Add the user menu for random access (seek)

It is usual that we would like to perform random access for videos to the timestamp we want.

a) Go to the ***playVideo*** function, replace the function with the following codes:

*Note that the parameters in the **Sleep** function should be replaced by your answer in Section C.5.*

```
void playVideo(FILE * fp, int iWidth, int iHeight, int iFrameRate, int iMaxFrameNum, unsigned char *
pYuvBuf, unsigned char * pBmpBuf)
{
	int i;
	for (i = g_iCurFrame; i<iMaxFrameNum; i++) // loop each frame
	{
		long timeElapse, t1;
		t1 = clock();
		// read YUV
		fread(pYuvBuf, sizeof(unsigned char), iWidth * iHeight * 3 / 2, fp);
		// convert and display
		yuv2bmp(pYuvBuf, pBmpBuf, iWidth, iHeight);

		timeElapse = clock() - t1;
		Sleep(Answer in Section C.5);

		g_iCurFrame = i;
	}

	printf("\n [p] Finished playback.\n");
}
```

As shown in the code, we record the current frame number to the variable ***g_iCurFrame***. And the playback starts at the frame based on ***g_iCurFrame***.

b) Go to the ***main*** function, replace the menu by the codes as follows:

```
…
// menu
while (1)
{
        // user input
        printf(" Menu: [p] play, [s] seek, [q] quit\n");
        printf(" Please enter the [player command]  : ");
        scanf(" %c", &szKey);
        // player action
        if (szKey == 'p')
        {
                playVideo(fpInputFile, iWidth, iHeight, iFrameRate, iMaxFrameNum, pYuvBuf,
pBmpBuf);
                if(g_iCurFrame == iMaxFrameNum - 1)
                {
                        fseek(fpInputFile, 0, SEEK_SET);
                        g_iCurFrame = 0;
                }
        }
        else if (szKey == 's')
                seekVideo(fpInputFile, iWidth, iHeight, iMaxFrameNum, pYuvBuf, pBmpBuf);
        else if (szKey == 'q')
                break;
        else
                printf("\n Incorrect player command.\n\n");
        printf("\n");
}
…
```

c) Run the program. You can enter “s” to seek the video as well.

3. Add the user menu to pause the video

Pause is another common VCR functionality.

a) Go to the ***playVideo*** function, replace the function with the following codes:

*Note that the parameters in the **Sleep** function should be replaced by your answer in Section C.5.*

```
void playVideo(FILE * fp, int iWidth, int iHeight, int iFrameRate, int iMaxFrameNum, unsigned char *
pYuvBuf, unsigned char * pBmpBuf)
{
	int i;
	g_iPlayerStatus = STATUS_PLAY;
	for(i=g_iCurFrame; i<iMaxFrameNum; i++) // loop each frame
	{
		long timeElapse, t1;
		t1 = clock();

		if (g_iPlayerStatus != STATUS_PLAY)
			return;

		// read YUV
		fread(pYuvBuf, sizeof(unsigned char), iWidth * iHeight * 3 / 2, fp);
		// convert and display
		yuv2bmp(pYuvBuf, pBmpBuf, iWidth, iHeight);

		timeElapse = clock() - t1;
		Sleep(Answer in Section C.5);

		g_iCurFrame = i;
	}
	g_iPlayerStatus = STATUS_PAUSE;

	printf("\n [p] Finished playback.\n");
}
```

As shown in the code, we record the player status (play/pause) to the variable ***g_iPlayerStatus***. When the video is playing, it is set to ***STATUS_PLAY***. If it is paused, it is set to ***STATUS_PAUSE***.

Look at ***wndProc*** function. This is a callback function to handle the message loop for the window which displays the video. Within the switch case, *WM_LBUTTONDOWN* handles the event when mouse left click is detected. And you can see that we have already added the codes for you. When the video is having playback and the mouse left click is detected, ***g_iPlayerStatus*** would be set to ***STATUS_PAUSE***.

b) Run the program. You can left click the video to pause the video and return back to the menu.

4. Demonstrations

Tasks	Check by tutors
Video playback	
Pause during video playback	
Random access to frame *n*	
Playback from frame *n* after random access to frame *n*	

5. Hand in the lab sheet to tutor

   Make sure you have written down all the answers before submitting the lab sheet.

6. Wrap up

   The image processing techniques, mentioned in Section B, can be applied to videos. Besides normal playback, random access and pause, you can also implement other VCR functionalities such as fast forward. This is the basic concept of video players and video editing tools of how they manipulate the pixels after the decoding of compressed videos which you will study later. If you know developing GUI well, e.g.: Visual C++ or C#, you are able to write a raw player software on your own.

- END-

Appendix: Sleep Function

Sleep function suspends the execution of the current thread until the time-out interval elapses.

```
VOID WINAPI Sleep(
  _In_ DWORD dwMilliseconds
);
```

Parameters

dwMilliseconds: The time interval for which execution is to be suspended, in milliseconds.

For more information:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms686298%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396

Appendix: fseek Function

Move the file pointer to a specified location.

```
int fseek(
   FILE *stream,
   long offset,
   int origin
);
```

Parameters

stream: Pointer to FILE structure.
offset: Number of bytes from origin.
origin: Initial position.
SEEK_CUR: Current position of file pointer.
SEEK_END: End of file.
SEEK_SET: Beginning of file.

Thus, file pointer pointing to the *stream* would be moved to the position which is the relative *offset* from the *origin*.

For more information:
https://msdn.microsoft.com/en-us/library/75yw9bf3.aspx

Appendix: Raw Image Format

This chapter describes the format of the (a) ***RGB 24-bit*** and (b) ***YUV 4:2:0*** image formats respectively. The raw image is organized as *M* x *N* pixels where *M* and *N* are its width and height, respectively.

We label pixels in an image for display as shown below.

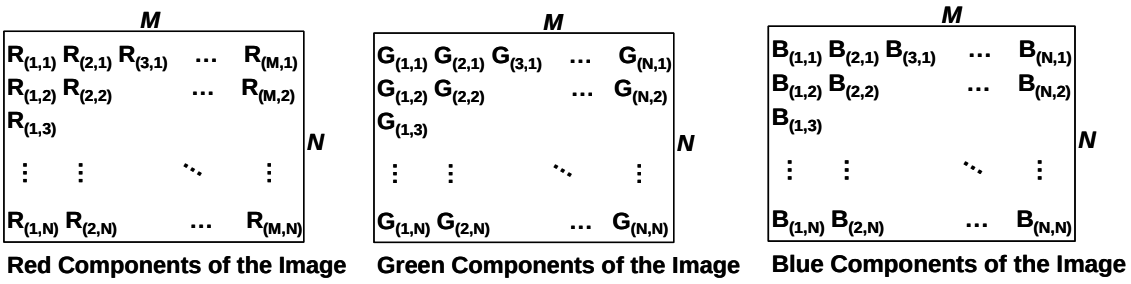

Fig A1. Pixel pattern in an image for display.

A. RGB Raw Image Format

For the RGB 24-bit image, each pixel composes of R, G and B color components. The raw RGB 24-bit image stored in **Windows Bitmap format is stored in an upside down direction** which means that the image needs to be flipped vertically before any image processing. Besides the vertical flipping of the image, the color components are also stored in a reversed stored, i.e. B, G, R, which is shown below.

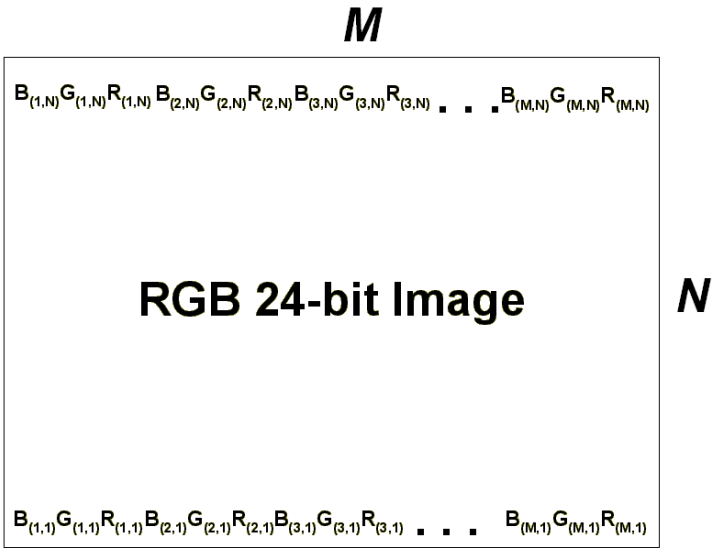

Fig A2. RGB Image Format.

Hence, the raw RGB 24-bit image stored in memory is in the following order:

$$B_{(1,N)}G_{(1,N)}R_{(1,N)}B_{(2,N)}G_{(2,N)}R_{(2,N)}\ldots\ldots B_{(M,N)}G_{(M,N)}R_{(M,N)}B_{(1,N-1)}G_{(1,N-1)}R_{(1,N-1)}\ldots\ldots$$
$$B_{(M,N-1)}G_{(M,N-1)}R_{(M,N-1)}\ldots\ldots B_{(M,1)}G_{(M,1)}R_{(M,1)}$$

B. YUV Raw Image Format

For the YUV 4:4:4 image, we label pixels in this image as shown below.

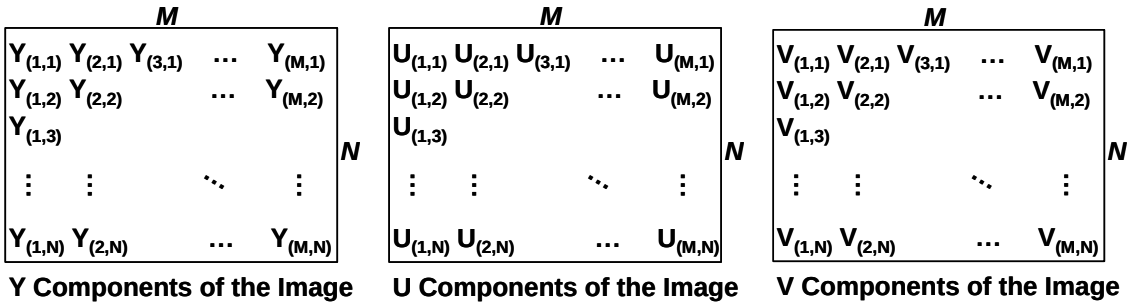

Fig B1. Pixel pattern in the image for display.

After converting the above YUV 4:4:4 images into the YUV 4:2:0 images, the color components, U and V, are decimated in both horizontal and vertical directions. The new pixel pattern of the YUV 4:2:0 is shown below.

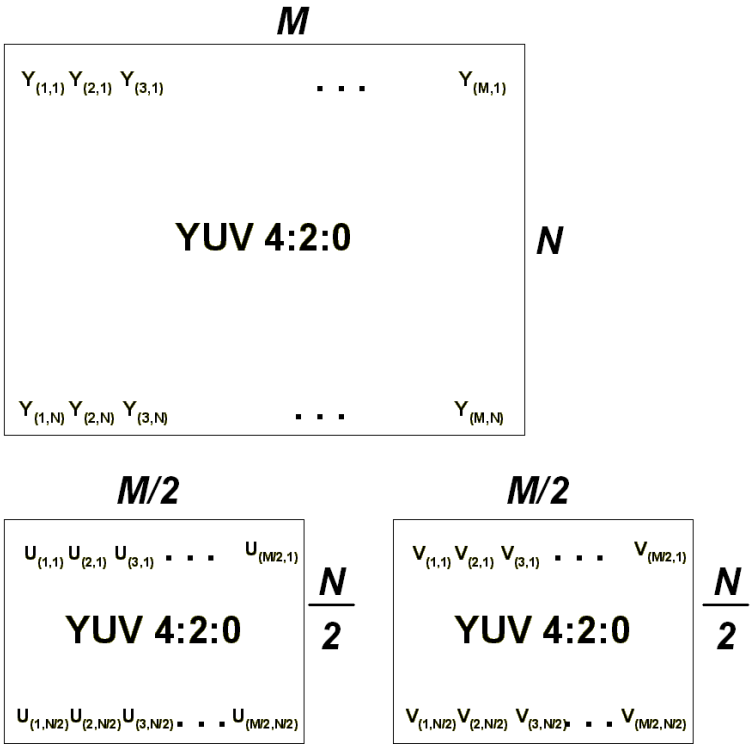

Fig B2. YUV 4:2:0 image format.

The raw YUV 4:2:0 image stored in memory is in the following order:

$$Y_{(1,1)}Y_{(2,1)}Y_{(3,1)}Y_{(4,1)}Y_{(5,1)}\ldots\ldots Y_{(M,N)}U_{(1,1)}U_{(2,1)}U_{(3,1)}U_{(4,1)}U_{(5,1)}\ldots\ldots$$

$$U_{(M/2,N/2)}V_{(1,1)}V_{(2,1)}V_{(3,1)}V_{(4,1)}\ldots\ldots V_{(M/2,N/2)}$$