CSIT 691 Independent Project

Video indexing

Student: Zhou Zigan

Email: zzhouag@ust.hk

Supervisor: Prof. David Rossiter

Contents

1. Introduction

Today, most of video image indexing are still description-based, the indexing only can be achieved by key words matching or text matching, so it contain less information about the content of videos and images. It is not an effective way to search videos and images. On the other hand, if it is still depending on description-based, it can just provide very limited information when people want to know what's in a video without watching the whole video.

Since, the purpose of this project is to do video summary using images. I wish to display the key-frame images with their significance, the significance of each key-frame is determined by the length of the corresponding shot. By doing this, I wish achieve the goal that people can obtain more useful and clear information about the video without play the video.

All of the following algorithms are implemented based on both C++ and C Language, and using the functions of OpenCV 2.3.1 library to do video and image processing.

OpenCV (Free Open Source Computer Vision) is a library of programming functions mainly aimed at real time computer vision. It has a BSD license (free for commercial or research use). OpenCV was originally written in C but now has a full C++ interface and all new development is in C++.

2. Basic concept

Before we go further about the project, we need to have a basic sense about the concept of video and image processing.

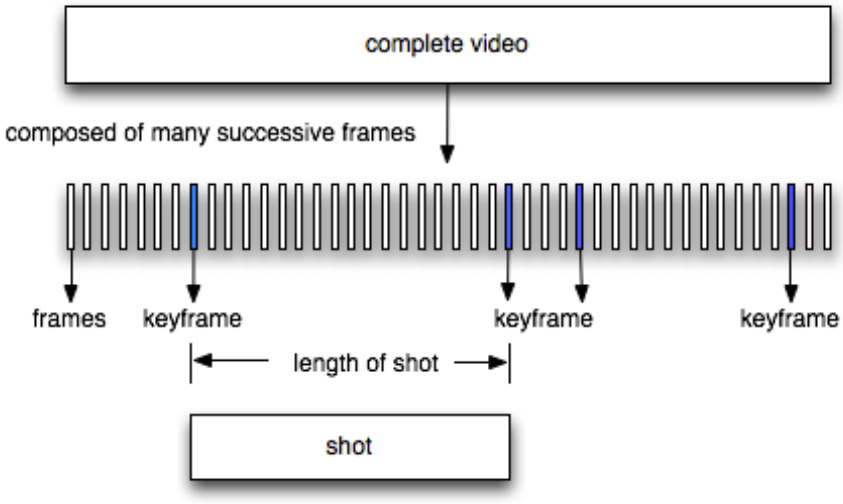


Figure 1. Video structure

2.1 Video Frame

A video or movie frame is a single picture or still shot, that is shown as part of a larger video or movie. Many single pictures are run in succession to produce what appears to be a seamless piece of film or videotape. Each frame can be selected on its own to print out a single photograph.

As we know each video is composed of many still images running in succession, we can do video processing via process the still images. It is a most common way to do the video processing.

2.2 Frame Per Second (Often note as fps)

Fps means the number of frame running in succession in one second. As of now, there are current three main frames rate standards in the TV and movie-making business: 24fps, 25fps, and 30fps. It means that there are only tens of milliseconds between two consecutive frames, there may be a huge project to extract all of the frames from a video with its original length. So, we must control the extract rate when we are capturing frames from video.

2.3 Key-Frame of video

In general, key-frame is the frame that can represent the content of a section of a video. Assume the fps is 1, for a video with 20 minutes long, we still have to capture about 1200 frames. It is inefficient and useless to display all of them, people prefer to play the whole video directly rather than go through the thousands of images. Thus, we need to extract key-frame to summarize the video.

2.4 Shot

In video or film, a shot is a continuous strip of motion picture film, created of a series of frames that runs for an uninterrupted period of time. A shot is defined by the beginning and end of a capturing frame process.

After we have a basic understanding about key-frame, we can simply define shot as a series of frames that runs for an uninterrupted period of time between two key-frames.

2.5 Pixel

Pixel (picture element) is the smallest unit of picture that can be represented or controlled. Each pixel has its own address. The address of a pixel corresponds to its two-dimensional coordinates in image. Each pixel is a sample of an original image; more samples typically provide more accurate representations of the original. The intensity of each pixel is variable. In color image systems, a pixel can be composed of different color model, a color is typically represented by three or four component intensities such as red, green, and blue, or cyan, magenta, yellow, and black.

As we know each frame is a single image, actually, we do video processing mainly based on image processing through looking inside the images and do some operation on the value of pixels.

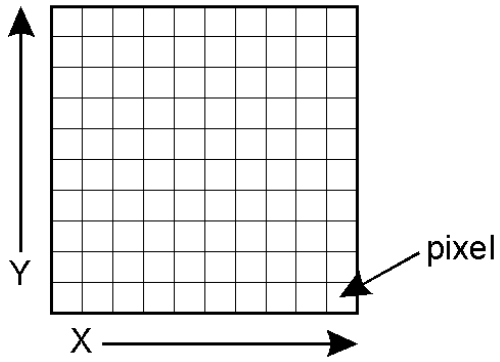


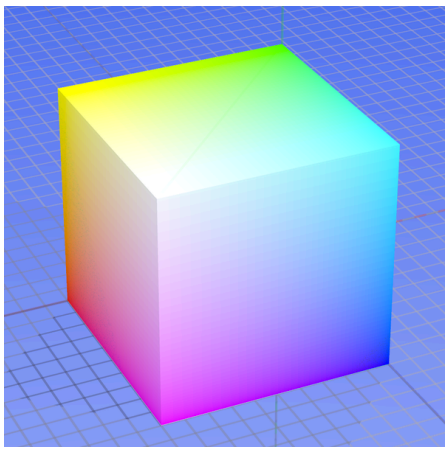
Figure 2. Pixels structure in images

2.6 RGB Color Model

The RGB color model is an additive color model in which red, green, and blue light is added together in various ways to reproduce a broad array of colors.

Each pixel on the computer screen is built by driving three RGB light sources. All the pixels together arranged in the rectangular screen surface conforms the color image.

In computing, the component values are often stored as integer numbers in the range 0 to 255, the range that a single 8-bit byte can offer.



(The RGB color model mapped to a cube. The horizontal x-axis as red values increasing to the left, y-axis as blue increasing to the lower right and the vertical z-axis as green increasing towards the top. The origin, black, is hidden behind the cube.)

Figure 3. RGB color model

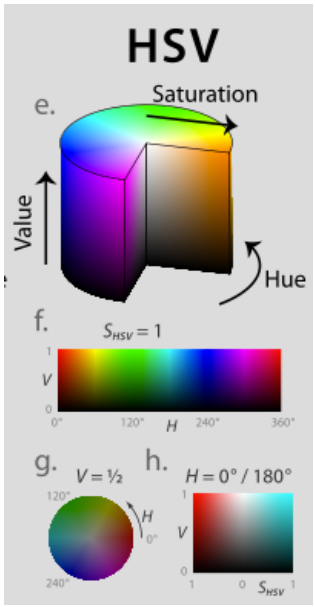
Figure 4. RGB color values

2.7 HSV Color Model

HSV is cylindrical-coordinate representations of points in an RGB color model, which rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cube representation.

HSV stands for hue, saturation, and value, and is also often called HSB (B for brightness).

In each cylinder, the angle around the central vertical axis corresponds to "hue", the distance from the axis corresponds to "saturation", and the distance along the axis corresponds to "lightness", "value" or "brightness".



For hue(H), starting at the red primary at 0°, passing through the green primary at 120° and the blue primary at 240°, and then wrapping back to red at 360°. The central vertical axis(V) comprises the neutral, achromatic, or gray colors, ranging from black at lightness 0 or value 0, the bottom, to white at lightness 1 or value 1, the top. The additive primary and secondary colors – red, yellow, green, cyan, blue, and magenta – and linear mixtures between adjacent pairs of them, sometimes called pure colors, are arranged around the outside edge of the cylinder with saturation 1; In HSV, mixing these pure colors with white reduces saturation, while mixing them with black leaves saturation unchanged.

Figure 5. HSV color model

3. Key-Frame Extraction

3.1 Capture Frame From A Video

The flow diagram as Figure 6:

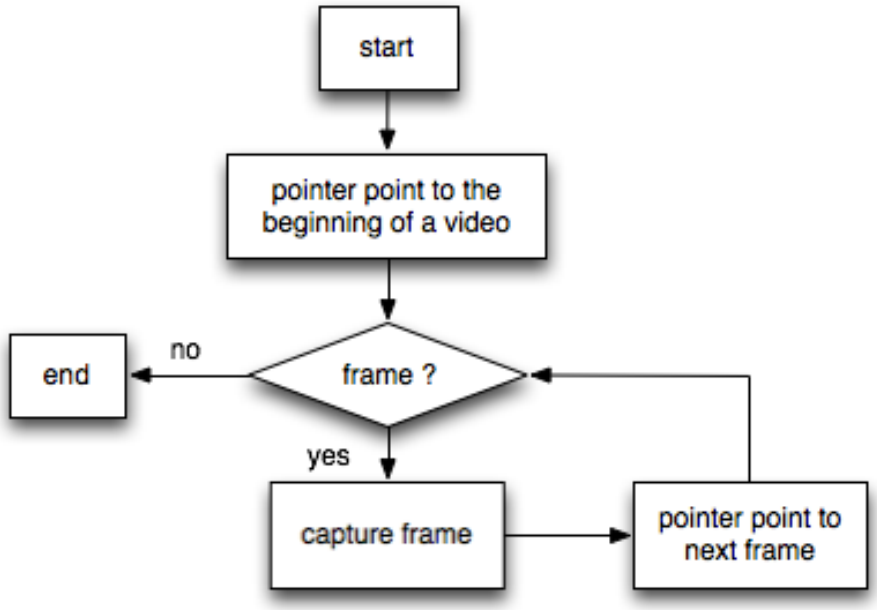


Figure 6. Flow diagram of frame extraction

Using the functions from OpenCV library, we can easily implement the process above:

```
cvCapture * capture = cvCaptureFromFile("/Users/zzg/Desktop/test.avi");
//Set capture pointer to the beginning of the video under a specific directory.
IplImage *frame;
//IplImage define a image structure which the format is native.
while(1){
frame = cvQueryFrame(capture);
//Capturing data from decode and return the just grabbed frame. It is a combination of
//cvGrabFrame() and cvRetrieveFrame().
if(!frame) {    //Out of the loop if no frame left.
                    break;
                }
            }
cvReleaseImage(&frame);
cvReleaseCapture(&capture);
```

When running the program above, pointer point to each frame from the video one by one until the last frame of video has been extracted. We can show and save all of frames in our computer to make sure the process is works as our expectation indeed.

```
Save frame: {cvSaveImage(filename, image); // Saves an image to a specified file.
            cvReleaseImage(&image)}

Show frame: {cvNamedWindow(name of window, CV_WINDOW_AUTOSIZE);
            // Creates a window.
            cvShowImage(name of window, image);
            // Displays an image in the specified window.
            cvDestroyWindow(name of window);
            cvReleaseImage(&image)}
```

3.2 Select Key-Frame

We can easily design an algorithm to extract key-frames from a video without considering the quality of key-frames. But it is not easy to achieve the target that key-frames can represent the most useful information about the video exactly.

So far, we know that pixel is the smallest unit of picture that can be controlled. And in computer vision, a pixel can be composed of different color model. For this reason, I decide to determine key-frame using color comparison method.

3.2.1 In RGB Color Format

As we know, in RGB color format, each pixel is composed of three components of red, green and blue. And component values are often stored as integer numbers in the range 0 to 255.

- **Difference between two RGB color**

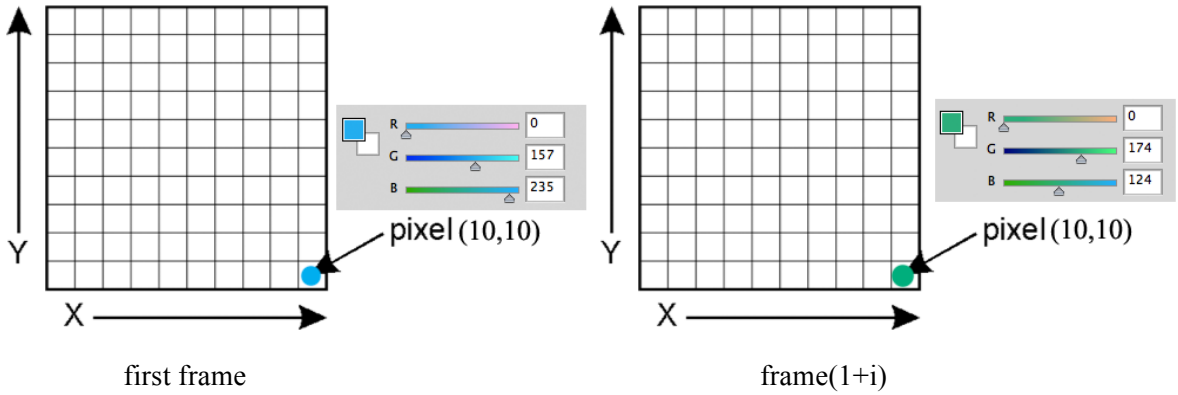
In chapter 2, we know that the RGB color model is mapped to a cube. Each color corresponds to a point in cube. So I think to define the difference as the distance between the two points in cube is more accuracy.

Assume there are two pixels p1 and p2, the color are p1(r1,g1,b1) and p2(r2,g2,b2), so the difference between these two pixels is

$$\text{distance} = \sqrt{(r1 - r2)^2 + (g1 - g2)^2 + (b1 - b2)^2} \quad (1)$$

The maximum distance between two pixels is $d_{max} = \sqrt{3 * 255 * 255} \approx 441$.

- **The basic idea**



first frame frame(1+i)

Figure 7. RGB color structure in images

As the pictures show above (i is a positive integer):

1. Firstly, we choose the first frame as the standard frame that is used to compare with the following frames.
2. Get the corresponding pixel value in both frames one by one, and computing their difference respectively.
3. After finishing 2, add the results in 2 altogether. The sum will be the difference between these two frames.
4. Finally, if sum is larger than a threshold we set, select frame(1+i) as a key-frame, then frame(1+i) become the standard frame. Redo 1 to 4 until there is no frame can be captured.

   (The maximum distance between two frames is d_{max} *(frame size), it is too large to use. So, we can use the sum in 3 and divided by (frame size) to get a smaller between 0 to 441, it is more easy to do calculation and comparison.)

- **Flow diagram**

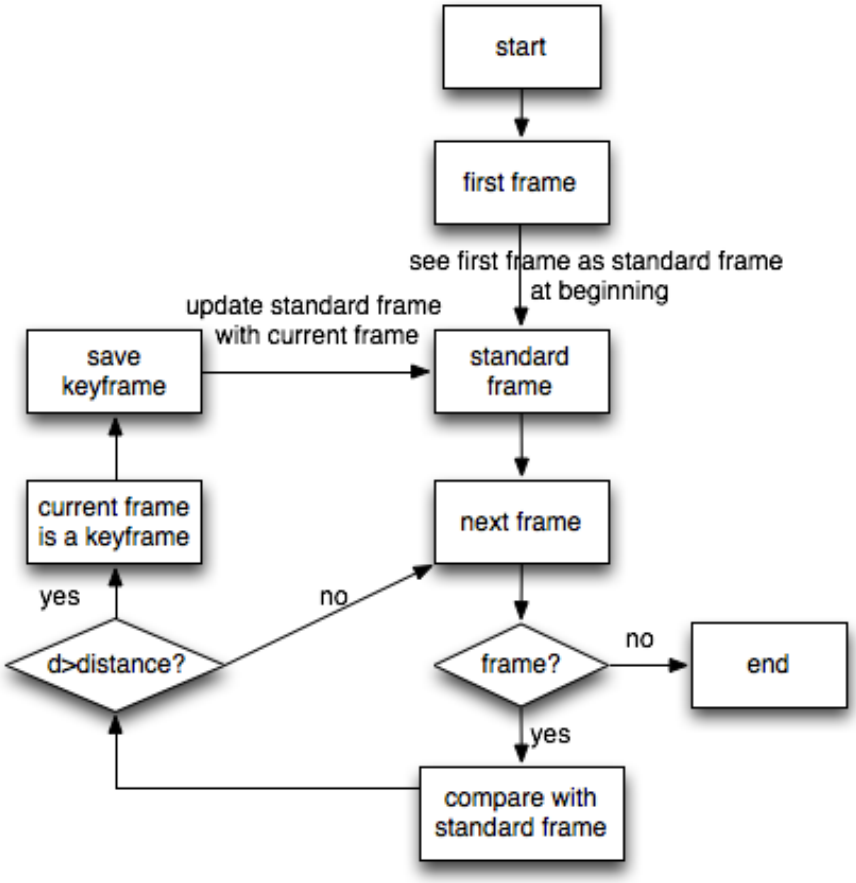


Figure 8. Flow diagram of key-frame extraction

- **Implementation**

```
Int const Distance=XXX;
void extractkeyframe()
{
      …… // initialize
      while(1){
      …… // capture frame
      if(!frame) {      break;
                  }
      If(nFrm==1){
      …… // consider first frame as standard frame to start the comparison and save it
                  }
      If(nFrm!=1){ //if not the first frame
            if(d>Distance){ // set a difference threshold in order to select key-frame,
```

```
                    // at this place, it means if the key-frame.
                ..... // set current frame as the standard frame and save key-frame
            }
        }
    ..... //free memory
}
```

3.2.2 In HSV Color Format

- **Difference between two HSV color**

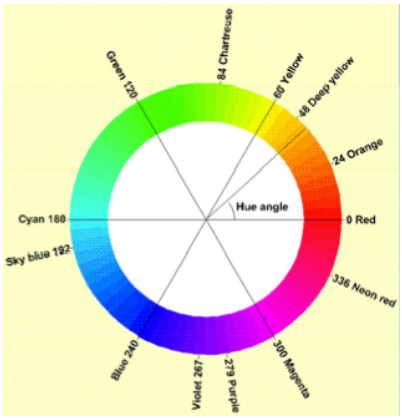


Figure 9. Hue

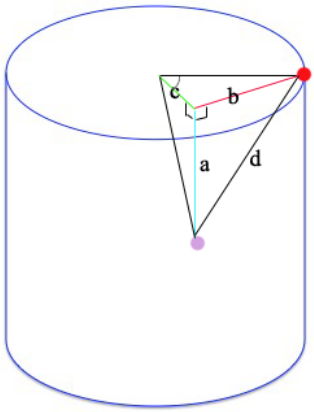


Figure 10. Distance between two pixels in HSV color model

If we only consider the value of hue, the difference between two pixels maybe describe as follow:

if abs(h1-h2)>180°, difference(h) = 360°-abs(h1-h2);
if abs(h1-h2)<=180°, difference(h) = abs(h1-h2);
the maximum contrast of two hue values is 180°;

But after careful consideration, I think it is inappropriate to compute the difference like above. Because of the cylindrical geometries of HSV, which is also a three-dimensional space, I suppose that the better idea to compute the contrast between two pixels is still to compute the distance between them.

Assume there are two pixels p1 and p2 in HSV color format, the color are p1(h1,s1,v1) and p2(h2,s2,v2), we can compute the difference between p1 and p2 as follow:

As Figure 10 shows, d is the distance between two pixels:

$$difference(h)=h1-h2 \quad (2)$$

$$b^2=(s1^2 + s2^2 - 2 * s1 * s2 * \cos(\text{difference(h)}) \quad (3)$$

$$d = \sqrt{b^2 + (v1 - v2)^2} \quad (4)$$

So the maximum distance between two pixels is $\sqrt{5}$ ≈2.236.

- **Convert RGB to HSV**

The algorithm:

$$M = \max(R, G, B)$$
$$m = \min(R, G, B)$$
$$C = M - m$$

$$H' = \begin{cases} \text{undefined}, & \text{if } C = 0 \\ \frac{G-B}{C} \bmod 6, & \text{if } M = R \\ \frac{B-R}{C} + 2, & \text{if } M = G \\ \frac{R-G}{C} + 4, & \text{if } M = B \end{cases}$$

$$H = 60^\circ \times H'$$

$$S_{HSV} = \begin{cases} 0, & \text{if } C = 0 \\ \frac{C}{V}, & \text{otherwise} \end{cases}$$

$$V = M$$

- **Key-frame extraction in HSV color model**

Except the difference of distance computation, the basic idea is the same as RGB. And we need to do a color format change from RGB to HSV before comparison. About the flow diagram and implementation structure, please refer to the RGB section above.

3.2.3 Improvement of key-frame extraction

By now, we can obtain key-frames from a video, but if we take the following states into account, there are still many functions can be improved.

1. In general, fps of a video is larger than 24, it means there are at least 24 frames in one second, if we process all of the frames of a video, it must be a huge procedure.
2. There are some frames with all black color, and it is often considered as the key-frame because the algorithm to extract key-frames is based on color comparison. Thus, these key-frames are worthless.
3. After using some videos to do testing, I found that some of the videos contain some short shots that composed of several quickly change frames, such as the prologue or epilogue of a video what is meaningless. And if all of these quickly change frames are considered as key-frames, there is not the result what we want to see.
4. Further more, the edge of the video is also less significant, and if we take the pixel of edge into account, sometimes it can occurs errors.

For the defects listing above, I make some improvement:

1. Set new fps. We can get the fps of a video by using the functions as follow:

```
int fps = cvGetCaptureProperty(capture, CV_CAP_PROP_FPS);
```

Since we have known the fps, we can set the new fps indirectly by altering the condition in the capture loop, like:

```
if(nFrm%fps==0){ //equals to capture a frame per second
……                    //do key-frame extraction procedure
}
```

2. The RGB color (0,0,0) corresponding to black, and smaller the RGB values, the picture is closer to black. In order to filter out the black frames from key-frames, we set a threshold in loop condition to discard the black images. For example:

```
if(d>Distance && color>threshold ){
        …… // do keyframe extraction
}
```

3. To avoid including too many frames about prologue or epilogue of videos, we must keep the distance of two adjacent key-frames larger than an threshold, in other words, we can reduce the minimum fps again to avoid a short time quickly change frames which is meaningless. It is inaccurate to do this via altering the fps directly, I accomplish that through adding a conditional loop. For example:

```
if(d>Distance && distance between two keyframe>threshold ){
        …… // do keyframe extraction
    }
```

4. The important information of video is mostly presented around the middle of the screen, making an appropriate pruning of the edge of images maybe good for improving the quality of key-frames and also can avoid some errors that are occurring when we try to compute the edge pixels (I still have no idea about why it happens).

```
for (int i=10; i<frame->height-10; ++i) {
    for (int j=10; j<frame->width-10; ++j) {
                        ……      // extract pixel values
                            }
                        }
```

We can sum up all the improvement together to get a final scheme.

3.3 Brief Summary

Until now, we can obtain key-frames with better quality from a video. Contrasting the key-frames that are extracted from RGB color format and HSV color format, the HSV color format produce a better result. However, it is also early to make a conclusion between RGB and HSV. The only thing what is specific is HSV is more intuitive and sensitive than RGB.

4. The Importance of Key-Frame

The aim at considering the importance of key-frame is to give a more clear sight on the result of video summary. In this chapter I will explain how to determine the importance of key-frames.

- **The basic idea**

I determine the significance of a key-frame according to the length of shot that which is behind the key-frame. In other words, the measure of importance of a key-frame is the number of frames between itself and the next key-frame. The importance of the last key-frame is measured by the distance from the itself to the last frame.

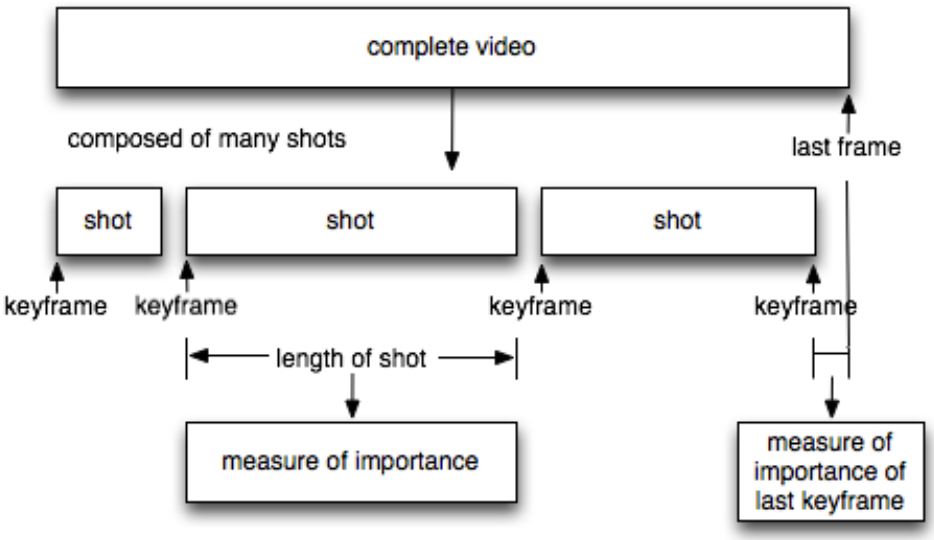


Figure 11. Importance of key-frames

- **Implementation**

We can know a key-frame is the number of which frame in the whole video by adding a counter into the loop, and make a subtraction between two adjacent key-frames, then the results will be the measure of importance of the former key-frame.

```
float scale[];
// to save the importance information of each key-frame.
int dif[];
 if(counter!=1){
                    ……
                    if(d1>20){
                    ……
                   dif[count]=1;
                   // number the first key-frame in the whole video
                    counter++; }
                     }
 if(nFrm%fps==0 && counter==1){
                  ……
                  if(d1>10){
                  ……
                     if(d>Distance)
                                 {
                     dif[count+1]=(int) (nFrm/fps+1);
                     //number of the next key-frame in whole frames.
                     if(dif[count+1]-dif[count]>10)
                                    {
                       ……
                     scale[count]=dif[count+1]-dif[count];
                      //compute the importance of the former key-frame.
                       ……
                     count++;
                                    }
                               }
                          }
                       }
```

5. Key-Frame Display

5.1 Display many images in one page

There are many ways to display many images in one page, we can use Visual Studio MFC or Xcode COCOA to do that directly, but I think it will be more intelligent and easy to do that through some preprocessing.

- **The basic idea**

Create a background image, then, select regions on the background image that we will use to display the key-frame images. Load the key-frame images and copy them to the corresponding regions.

The number of key-frame of each video is unfixed, if it is too large to display in one page, we can create a new page to display them. So a threshold will be set in order to turn to next page automatically if the current page is full.

In this project, I plan to display 12 images in one page with a fixed model. The number of images in the last page will range from 1 to 12.

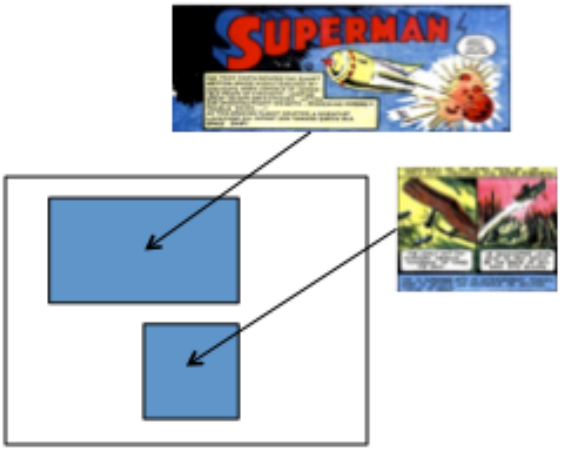


Figure 12. Method to display images

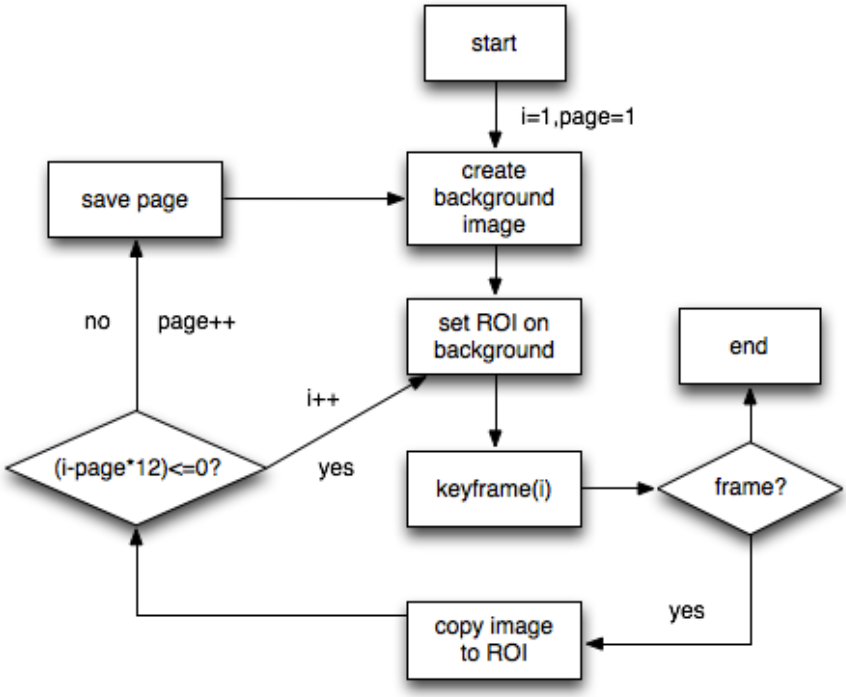


Figure 13. Flow diagram of display

- **Implementation**

```
void displaykeyframe(){
int index=1,page=1; //index=(page-1)*12+1
    while(1){
        if(count<=0)
        { break;}
        if (count-12>=0){
            twelveInOne2("",page,index); // display function
            index+=12; //ready to be used in next page
            tempcount-=12; //compute how many key-frame are not being displayed.
            page++;}
        else if(count>0){ //if the remaining key-frames is less than 12.
            twelveInOne2("",page,index);
            index+=12;
            tempcount-=12;
            page++;}
  }

  void twelveInOne2(char * title,int page, int index){
   ……
    DispImage = cvCreateImage( cvSize(1100 , 700), 8, 3 ); //create background image
```

```
CvRect rect; //Store coordinates of a rectangle
……
        for(int j=0;j<3;j++) //row
        {
                for(int i=0;i<4;i++) //column
                {
                sprintf(str,"/Users/zzg/Desktop/image/%d.jpg",i+j*4+index);
                //transfer string and (i+j*4+index) to str
                frame=cvLoadImage(str); //load key-frame images
                if(!frame){break;
                    }
                 rect.height=125;
                 rect.width=200;
                  //set the height and width of rectangle
                 rect.y=j*(200+10)+10; //choose row
                 rect.x=i*(200+50)+50; //choose column
                 cvSetImageROI(DispImage,rect);
                 // Sets an image Region Of Interest (ROI) for a given rectangle
                 cvResize(frame,DispImage);
                 //resize and copy key-frame images to background image
                 cvResetImageROI(DispImage);
                 cvReleaseImage(&frame); }
        }
      ……
}
```

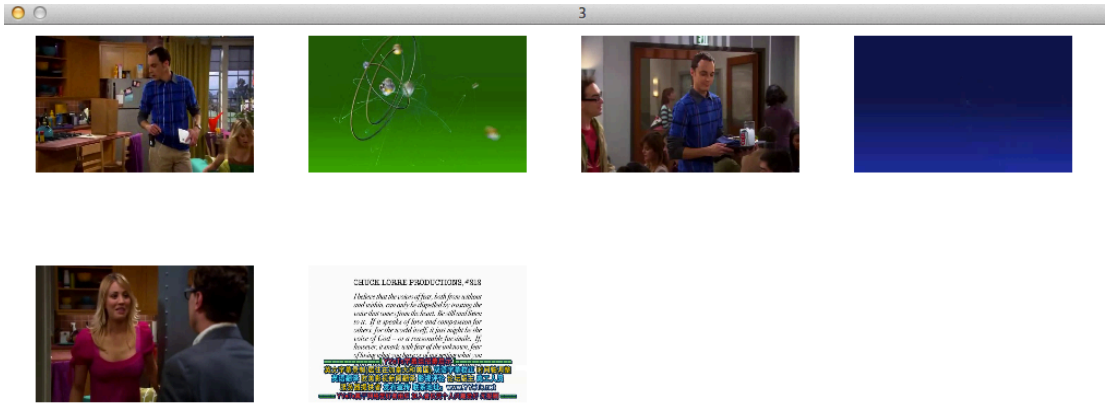


Figure 14. Key-frames display result(last page)

5.2 Improvement of key-frame display

- **The importance of key-frame**

1. Display key-frames and show their importance by the size of images. The larger the size, the greater the key-frame is. To achieve this goal, we need to link the size of images to the measure of importance that is obtained in chapter4. Some of the key-frame's measures are too large so that the images size is overloading. So I set a threshold measure, the measures are all equal to the threshold measure if the actual measures are larger than that. In additional, we use relative measure instead of absolute measure to moderate the fluctuation of the image size.

Implementation:

```
if(measure[i+j*4+index]<80){ // 80 is the threshold measure I set
        rect.height=125-125*(measure_max-measure[i+j*4+index])/200;
        rect.width=200-( measure_max -measure[i+j*4+index]);
        // measure_max is the maximum measure of all
        rect.y=j*(200+10)+10;
        rect.x=i*(200+50)+50;
        ……}
if(measure[i+j*4+index]>=80){ //measure larger than the threshold measure
        rect.height=125;
        rect.width=200; // all of them have the maximum size
        rect.y=j*(200+10)+10;
        rect.x=i*(200+50)+50;
        ……}
```

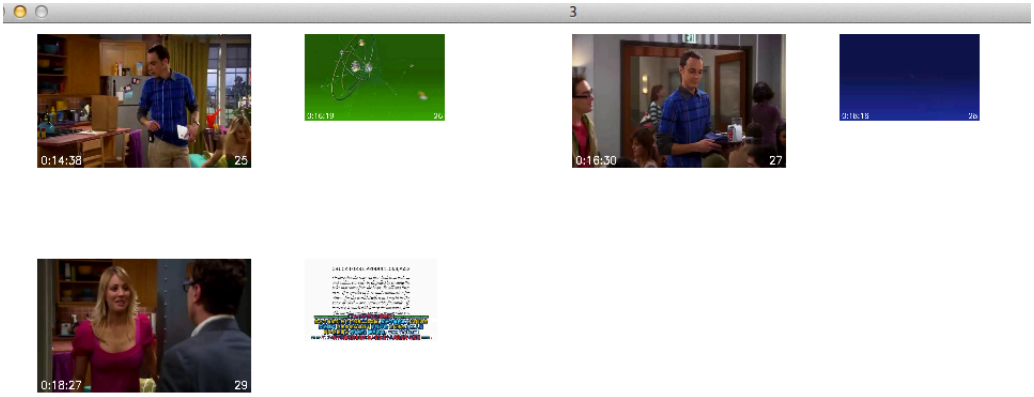


Figure 15. Key-frames display result

However, it still has the problem that some of the key-frames are too small to watch.

2. Display key-frames and show their importance by the thickness of images. The thicker the image, the more important the key-frame is. All of key-frames have the same size but the difference thickness. By this method, we can avoid the problem that which occurs above. The thickness is made by moving the image with a small deviation in both x and y axis for a fixed time. The time is determined by the measure of importance directly.

Figure 16. Images moving

Implementation:

```
if(measure[i+j*4+index]<80){
  for(int k=0;k<measure[i+j*4+index];++k) //set a loop to moving the image
  {
        rect.height=125;
        rect.width=200;
        rect.y=j*(200+30)+10+k; // deviation in y axis
        rect.x=i*(200+50)+80-k/3; // deviation in x axis
        ……    }
}
if(measure[i+j*4+index]>=80){
      for(int k=0;k<80;++k) //set a loop to moving the image
      {
            rect.height=125;
            rect.width=200;
            rect.y=j*(200+30)+10+k; // deviation in y axis
            rect.x=i*(200+50)+80-k/3; // deviation in x axis
        ……}
}
```

Figure 17. Key-frames display result

- **Some text contents**

1. Adding time to key-frames. We know the total frames we have captured under the current fps. For example, we use 1fps in this project, so we can get the seconds of each key-frame. And through the following program, we can change seconds into hours.

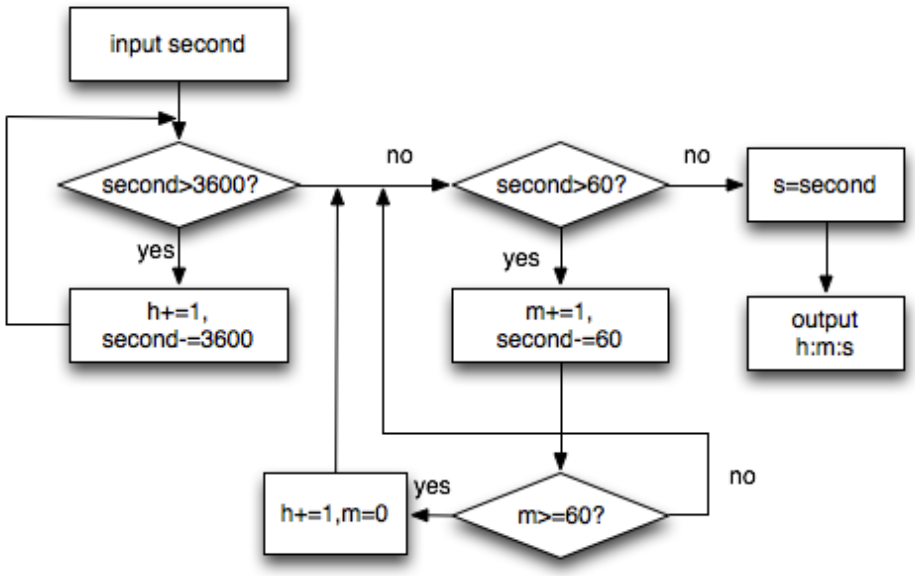


Figure 18. Flow diagram of changing time model

2. Number the key-frames, it easy to do that by using a counter.
3. Show these text contents on the images.

```
void cvText1(IplImage* img, string text, int x, int y)
{
CvFont font;
double hscale = 1.0;    //horizontal scale of text
double vscale = 1.0;    //vertical scale of text
int linewidth = 2;
cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX |
CV_FONT_ITALIC,hscale,vscale,0,linewidth); //Initializes font structure
CvScalar textColor =cvScalar(255,255,255);
CvPoint textPos =cvPoint(x, y); //position to put the text
cvPutText(img, text.c_str(), textPos, &font, textColor);// Draws a text string
}

{
……
sprintf(str2,"%d",i+j*4+index);
cvText1(frame, str2, frame->width-50, frame->height-10);
float h=0,m=0,s=0;
time(tempdifcount[i+j*4+index],&h,&m,&s);
sprintf(str3, "%.0f%s%.0f%s%.0f", h,":",m,":",s);
cvText1(frame, str3, 10, frame->height-10);
……}
```

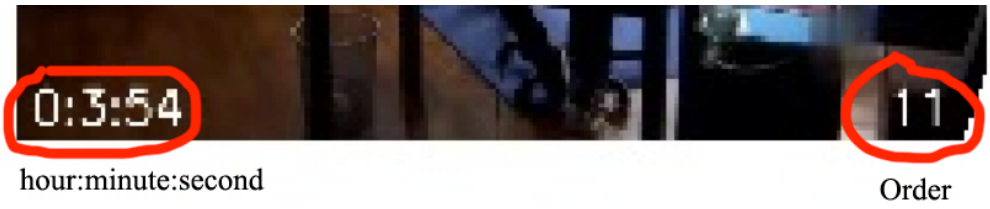


Figure 19. Text contents display

6. Conclusion

Content-based video indexing is an emerging research area that has received growing attention in the research community over the past decade. And with the rapid growth of the internet, a consequence of the growing consumer demand for visual information about the video. Because of the precision, summarize video not only based on the keywords will become important in video retrieval.

This whole program can extract key-frames from a video through compare the color of frame, and display them with the information of their importance in two ways. This project do video summary mainly based on the color information.

The challenge ahead is how to do the high-level video indexing, such as focus on the real content of video rather than its appearance content, and the difference between description of content and how the user perceives the content are the key to implement.

7. Reference

[1] WIKIPEDIA. http://en.wikipedia.org/wiki/HSL_and_HSV. Retrieved on 10 December 2011

[2] OpenCV 2.3 Documentation. http://opencv.itseez.com/index.html. Retrieved on 10 December 2011

[3] WIKIPEDIA. http://en.wikipedia.org/wiki/RGB_color_model. Retrieved on 10 December 2011

[4] Introduction to programming with OpenCV.
http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html#SECTION00050000000000000000. Retrieved on 10 December 2011

[5] View pixel values of image using OpenCV for debugging.
http://blog.adnannoon.com/tag/access-pixel-values-of-opencv-image. Retrieved on 10 December 2011

8. Appendix

Minutes of the 1st Project Meeting

Date: 30 September 2011 (Friday)

Time: 10:40 AM

Place: Rm. 3512

Attending: Prof. Rossiter, Zhou Zigan

Absent: None

Recorder: Zhou Zigan

1. Approval of minutes

This is first formal group meeting, so there were no minutes to approve.

2. Report on Progress

Since this is the first meeting, there is no progress to be reported.

3. Discussion Items

- Looking into a video.
- Accessing pixel values of an image.

4. Meeting adjournment and next meeting

The meeting was adjourned at 11:00 AM.

Minutes of the 2nd Project Meeting

Date: 7 October 2011 (Friday)

Time: 10:45 AM

Place: Rm. 3512

Attending: Prof. Rossiter, Zhou Zigan

Absent: None

Recorder: Zhou Zigan

1. Approval of minutes

The minutes of the last meeting were approved without amendment.

2. Report on Progress

The program can capture frames from a video, and learning about how to access pixel values of frames is ongoing.

3. Discussion Items

- Save frames that have been captured in order to further processing.
- Accessing pixel values of an image.

4. Meeting adjournment and next meeting

The meeting was adjourned at 11:05 AM.

Minutes of the 3rd Project Meeting

Date: 21 October 2011 (Friday)

Time: 10:45 AM

Place: Rm. 3512

Attending: Prof. Rossiter, Zhou Zigan

Absent: None

Recorder: Zhou Zigan

1. Approval of minutes

The minutes of the last meeting were approved without amendment.

2. Report on Progress

The program can use some color information of frames to do some simple image processing. And learning about how to get pixel values with a high operability is ongoing.

3. Discussion Items

- Algorithm about key-frame extraction
- Method about getting the importance information of every key-frame.

4. Meeting adjournment and next meeting

The meeting was adjourned at 11:05 AM.

Minutes of the 4th Project Meeting

Date: 11 November 2011 (Friday)

Time: 11:15 AM

Place: Rm. 3512

Attending: Prof. Rossiter, Zhou Zigan

Absent: None

Recorder: Zhou Zigan

1. Approval of minutes

The minutes of the last meeting were approved without amendment.

2. Report on Progress

The program can obtain the RGB color values and HSV color values of each pixel. The key-frame extraction step is ongoing.

3. Discussion Items

- Method to display the final result of key-frame extraction.
- Idea of combining the importance of every key-frame with key-frame display.

4. Meeting adjournment

The meeting was adjourned at 11:45 AM.

Minutes of the 5th Project Meeting

Date: 15 December 2011 (Thursday)

Time: 2:15 PM

Place: Rm. 3512

Attending: Prof. Rossiter, Zhou Zigan

Absent: None

Recorder: Zhou Zigan

1. Approval of minutes

The minutes of the last meeting were approved without amendment.

2. Report on Progress

Complete the whole program that can extract key-frames from most of video, and display them with their importance information. The final report is nearly finished.

3. Discussion Items

- Finish the final project report.

4. Meeting adjournment

The meeting was adjourned at 2:45 PM.