Libraries : Video

Video is a series of still images that are displayed consecutively at some reasonable rate. The result is interpreted by our brains as motion. Modern video is displayed at a rate of just a little bit less than 30 images or frames per second.

Using Processing, we can model our work with video very closely to the work we did with images in the previous set of notes. For each frame or image in the video there is:

- a **width** variable
- a **height** variable
- a **pixels** array

We can get the pixel color values copied into the **pixels** array with the **loadPixels( )** function and we can send altered values from the **pixels** array back to memory with the **updatePixels( )** function.

Everything we said about images applies to each frame in a video file.

Shiffman presents a number of examples in Chapter 16 of the book. His first examples work with a camera attached to your computer. These examples have the following code in common:

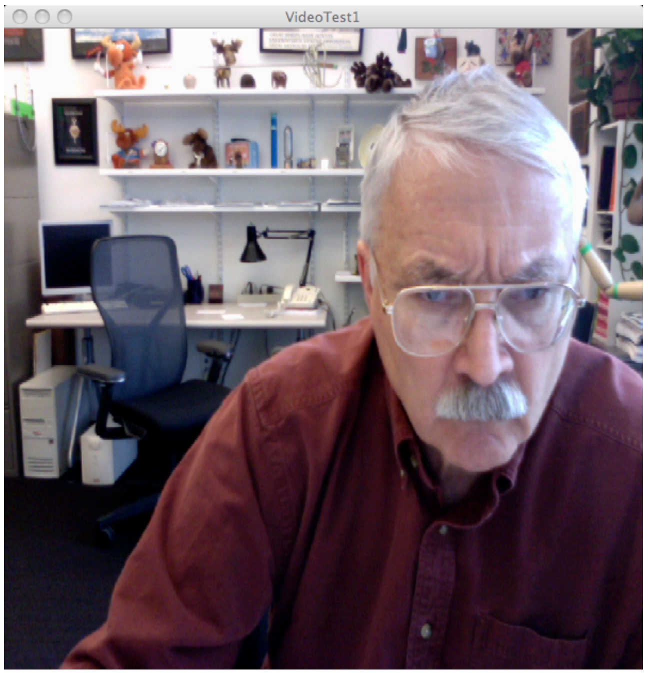
- **import processing.video.*;**
  The video library is part of the download when you install Processing on your computer. This just tells Processing that you are going to use parts of the video library.
- **Capture video;**
  This creates a reference to the **Capture** class that works with the camera.
- **video = new Capture(this, width, height, 15);**
  This assigns the reference **video** to communicate with the camera. The arguments are:
  this - this program
  width - the width of the video image
  height - the height of the video image
  15 - the number of frames of videoper second.

-
  ```
  if( video.available( )  )
  {
     video.read( );
  }
  ```
  The **if** *uses the function* **available( )**. *This function returns true if there is video data ready to use.*
  Within the **if**, *the* **read()** *function is called to copy the video data from the camera into the computer's memory.*

At this point we can go to the code used in the previous set of notes. We can call **video.loadPixels( )**, *search the data and alter any values that we want to. We can then call* **video.updatePixels( )** *to alter the computer's memory and we can call* **image(video, . . .)** *to display the altered video image.*

The functions **red( )**, **green( )**, *and* **blue( )** *can be used to get the values for these three colors from any element of the* **pixels** *array. The values returned are* **floats**.

Here is one un-altered frame from the camera on Jim's office computer.

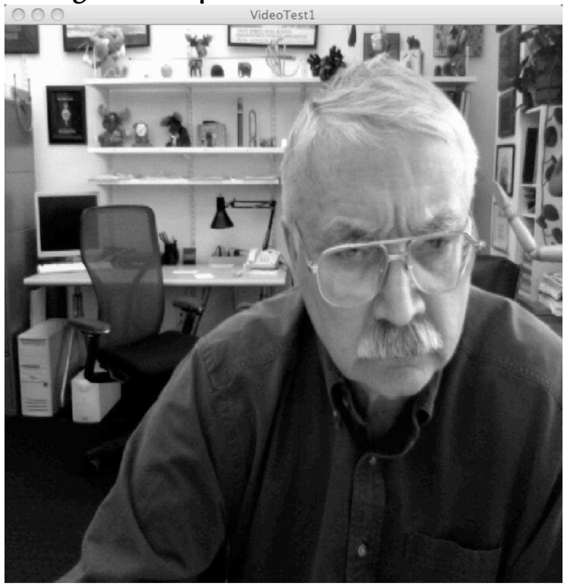


Here is a piece of code that alters the image of Jim in his office from color to gray:

```
import processing.video.*;

Capture video;

int x, y, dX, dY;

void setup()
{
   size( 600, 600 );
   video = new Capture
    ( this, width, height, 30 );
}
```

```
void draw( )
{
  if( video.available( )  )
  {
     video.read( );
  }
  loadPixels( );
  for( int i = 0;
        i < pixels.length; i++ )
  {
     float r = red(video.pixels[i] );
     float g = green(video.pixels[i] );
     float b = blue( video.pixels[i] );
     float sum = r + g + b;
     float average = sum/3;
     video.pixels[i] = color( average ) ;
  }
  updatePixels( );
  image( video, 0, 0 );
}
```

And here is the image this produces:



This code does the following:

1. check to see if there is any video to read
2. copy the information in memory to the **pixels** array in the object referenced by **video**
3. traverses the **pixels** array from the zeroth element to the last element.
   - For each element, the **red**, **green**, and **blue** values are extracted,
   - summed, and
   - averaged.
   - Then the average value is used to assign the **pixels** element a gray value instead of color.
4. Transfers the data in the altered **pixels** array back to the computer's memory
5. draws the **video** frame's image in the window with the altered data

In Shiffman's examples, he uses a set of nested loops to traverse the array:

```
 for( int x = 0; x < video.width; x++)
 {
    for( int y = 0; y < video.height; y++)
    {
      loc = x + y*video.width;
      fgColor = video.pixels[loc];
      bgColor = backgroundImage.pixels[loc];

      float r1 = red(fgColor);
      float g1 = green(fgColor);
      float b1 = blue(fgColor);
      float r2 = red(bgColor);
      float g2 = green(bgColor);
      float b2 = blue(bgColor);

      diff = dist( r1, g1, b1, r2, g2, b2 );

      if ( diff > threshold )
      {
       pixels[loc] = fgColor;
      }
      else
      {
        pixels[loc] = color( 0, 255, 0 );
      }
    } // end inner for loop
 } // end outer for loop
```

His code traverses the entire array in order of the pixels as they appear on screen and not in a linear order .