Video – revised for Processing 2.0

Note: In working on this, Processing seems to crash frequently when Jim changed from file to file and tried different ideas. If you are going to work with video be sure to save your work often and before every run.

[control][alt][delete] brings up the Task Manager on Windows machines
[option][command (the apple key)][esc] brings up a similar window that allows you for do a "force quit" on an application.

You may need to use these.

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

The demo code all share this common set of code:

1. You must import the video library from the Sketch menu in the IDE. When you do this, the IDE will add this line of code to your program:

```
   import processing.video.*;
```

Do not type this yourself – if you do, it will not work!

2. You need a **`Capture`** object declared as a global variable.

```
    Capture video;
```

The demos use the object name **`video`**. The provided demos (more later) use the name context and John and Jim often use the name, **`cam`**. Choose your own but make it obvious that it is an object reference to the **`Capture`** class.

3. This is where we begin to find differences between the "older" notes and demos and Java 2.0. You need an array of Strings for the possible camera names. The single camera on most laptop portables can provide different formats of video signals and we need to tell Processing which one to use. The dimensions of the window are very important. Use **`size(640/480)`** or a multiple like s**`ize(1280, 960)`**. Otherwise you may not get video but you may get a lot of dire warnings in the Console window. The code on the next page gets the names of those signal formats as Strings and uses the first one as the source:

```
void setup( )
{
 size( 640, 480 ); // These values are are important.
                   // See the notes!
 String [ ] camreras = camptüre.list( );
 if ( cameras.length == 0)
 {
   println("There are no cameras available.");
   exit( );
 }
 // execution does not get here if there are no cameras
 video = new Capture( this, camera[0] );
```

You can use:

```
 video = new Capture( this, width, height );
```

or you can use depending on which one works for what you are trying to do.

```
 video.start( );
  . . . // what ever else you need to do
```

4. Next we move to the **`draw( )`** function:

```
 void draw( )
 {
   if (video.available( ) )
   {
     video.read( );
      . . . and the rest of your code

   }  // end of the if
 }  // end of draw( )
```

Just showing the video image:

Just displaying video from the camera is very straightforward. Once the connection is made between the camera and the program, the video image can be shown like a jpeg like a jpg:

```
    image(video, 0, 0);
```

or

```
    image(video, 0, 0, width, height);
```

Demo VideoTest1 and 2 show this type of use.

Shiffman presents a number of examples in Chapter 16 of the book. His first examples work with a camera attached to your computer. If you make these modifications, most of his code "should" work.

Altering the video image or using the video data:

If you want to work with the pixel values of the video and display altered images, the strategy seems to be different (or Jim misunderstood the original code…)

One way of viewing the strategy is the following:

First, in the setup( ) function we get access to the pixels array of the graphics window Processing uses by calling loadPixels( ).

```
loadPixels( );
```

With this loadPixels( ) call we can alter the value of any pixel that is being displayed in the graphics window.

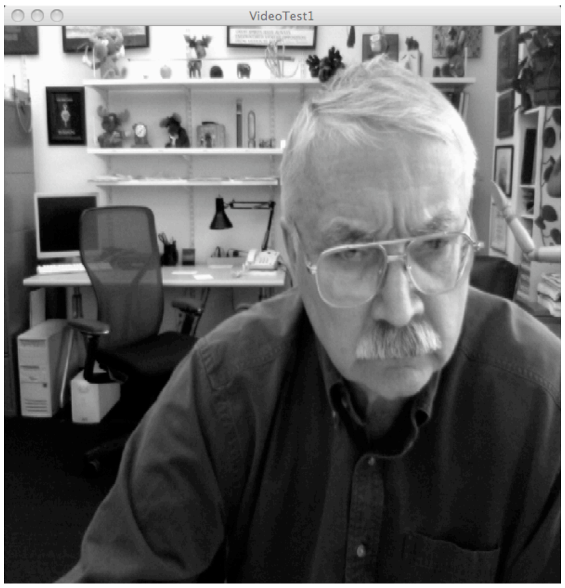
Then:

1. Traverse video's pixel array visiting the elements we need to visit
2. Extract data from each element
3. Compute the data we need
4. Alter the pixel array of the graphics window with the computed data
5. Save any values we need for the next iteration or the next frame.

If we are going to alter the video image, we DO NOT SHOW the original video image. We alter the corresponding element of the pixel array of the graphics window and display that when we are done.

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

`void draw( )`	
`{`	
`if ( video.available( ) )`	Check to see if there is a video signal available this frame.
`{`	
`video.read( );`	Read it.
`video.loadPixels( );`	Load the video's pixels array for access.
`for(int i = 0;` `i < video.pixels.length;` `i++)`	Traverse the video's pixel array pixel by pixel.
`{`	
`color videoColor =` `video.pixels[i];`	Get the color for the pixel.
`float r =` `red(videoColor);`	Extract the red value.
`float g =` `green(videoColor);`	Extract the green value.
`float b =` `blue(videoColor);`	Extract the blue value.
`float average = (r+g+b)/3;`	Average them to produce a gray value.
`pixels[i] =` `color(average);`	Transfer the color value of the graphics window's Corresponding element in its pixel array .
`}`	
`updatePixels( );`	Transfer the updated pixels array of the grahics window back to Processing for display.
`}`	



In Shiffman's examples (which no longer work for Processing 2.0), 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
```

This is different from the code written by Jim. Either works fine. Jim's might be a bit easier to follow.