CMPT 365 Project Spring 2018

Due Date: Apr. 05, 2018 (Thurs.)

Objective: Apply course materal to a more in-depth project. The Project is in teams of 2. Each team gets a single score, and both members of the team receive identical marks.

For this project, the programming language and platform are open; the objective is to learn about the problem, not to make a lovely GUI (at least, not necessarily — do so, if it aids you). It's also open-ended in that there are many approaches to the problem that stem from the discussion below: e.g., the specification discusses video wipes, and we don't look at video dissolves or other types of gradual transitions.

1. Project

1.1. STI by Copying Pixels

We are interested in *finding and characterizing video transitions*, specifically cuts and wipes.

One approach to this problem is to construct a "spatio-temporal" image [STI ☺] = an image which contains video content for each frame along the ordinate axis, versus time along the abscissa. A simple version of such a construction consists of copying a column (or row), or weighted average of a few, directly into the STI.

E.g., suppose that we have a video containing a wipe, as in Fig. 1 below:

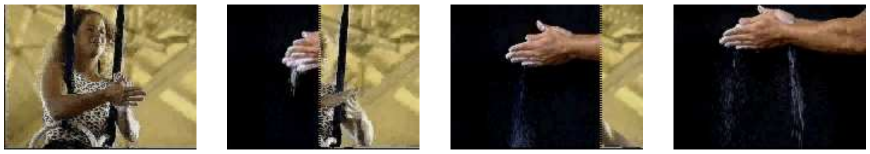
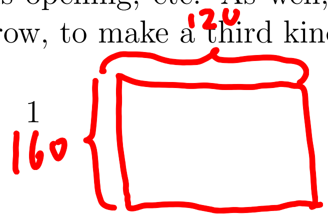


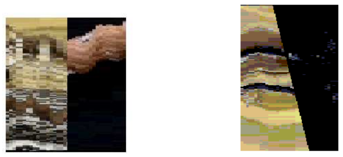
Figure 1: Frames from a video that includes a horizontal wipe.

Now suppose we copy the center column from each frame into a new, STI, image: the STI has the same number of rows r as does the original video frame, but a number of columns equal to the number of frames f in the video. The result is shown below in Fig. 2(a). If, instead, we copy over the center row from each frame, turned sideways, then the STI has the same number of rows as c, the number of columns in the original video, and a number of columns f equal to the number of frames. Such an STI made up of rows is shown in Fig. 2(b).

For example, this video happens to be 120×160 pixels per frame, and the clip has 100 frames. So the STI made from ~~the~~ center columns is of size 120×100. The STI made from the center rows is 160×100 pixels. Clearly, for the former a transition is evident at the time when the video wipe reaches the center column of the frame. Since the wipe is upright, the edge is also upright. For the latter STI, made from rows, the wipe shows up as a diagonal edge along the time direction.

In general, one can make a taxonomy of such pairs of STIs based on whether the wipe is a vertical wipe, a horizontal wipe, an iris opening, etc. As well, one can use a diagonal across the frame, rather than a column or row, to make a third kind of STI.



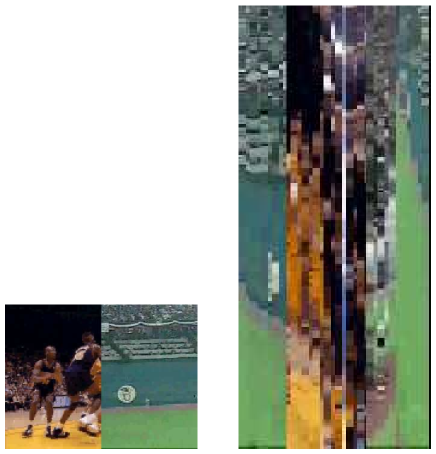


(a) (b)

Figure 2: (a): STI consisting of center column from each frame; (b): And from center row.

1.2. STI by Histogram Differences

One problem with the method outlined above is how to find that nice edge our eye sees so clearly in Fig. 2. For one thing, it's often the case that the edge is not so visible, especially in noisy videos such as from broadcast TV. Also, if we rely on the pixels themselves, small movements can muddy the STI. For example, Fig. 3(b) shows an abrupt cut allright, and then the flash of a flash-camera, but the diagonal edge indicating a wipe that follows is hard to see.



(a) (b)

Figure 3: (a): Frame during a wipe (broadcast video); (b): STI as in Fig. 2(b).

One approach that is taken to characterize images is to use a histogram of colour, rather than just the raw pixel data. Histograms are fairly insensitive to troublesome problems such as movement and occlusion (i.e., losing sight of an object over time).

It turns out that we also know that if we replace the colour, RGB, by the chromaticity (see Chapter 4)

$$\{r, g\} = \{R, G\}/(R + G + B)$$

then the image is much more characteristic of the surfaces being imaged, rather than of the light illuminating those surfaces. So, instead of colour, $\{R, G, B\}$, let's use chromaticity, $\{r, g\}$. [But watch out for black, i.e., $\{R, G, B\}$=$\{0, 0, 0\}$, pixels.]

The nice thing about chromaticity is that it's 2D. So our histogram is a 2D array, with r along one axis and g along another. The chromaticity is necessarily in the interval $[0, 1]$. But how many bins along each axis should we use? Applying (a cheap version of) a rule of thumb called Sturges's Rule, the number of bins $N = 1 + log_2(n)$, where n=size of data, so a rough idea is to use $n = number\_of\_rows$, so e.g. for frames of size 120×160 we would use $N = floor(1 + \log_2(120)) = 7$ bins along each of the r and g directions. That makes our histogram, H, a small, 7×7 array of integer values. If we normalize to make the sum of H equal unity, then we have an array of floats.

Now, suppose we wish to compare one column in a frame with the same column in the previous frame. Then we should compare the histogram H_t at time t with the histogram H_{t-1} for the previous frame. Here, we make a 2D histogram H_t just for the particular column we're working on, and the histogram H_{t-1} for a histogram for the same column, but for the previous frame. Let's get a measure of *histogram difference*. One measure would be the Euclidean distance between all the entries in the histogram: the square root of the sum-of-squares of the differences between entries. But a better value turns out to be the so-called "histogram intersection":

$$\mathcal{I} = \sum_i \sum_j min\left[H_t(i,j)\,,\, H_{t-1}(i,j)\right]$$

This formula assumes that one has first divided each histogram by its sum, so that each adds up to unity, e.g., $\sum_i \sum_j H_t(i,j) = 1$. How $\mathcal{I}$ works is that, for each array entry, you add up the smaller of the values at that array location, over the two histograms being compared.

Since we compared histograms for a single column, and obtained a scalar $\mathcal{I}$, each column in the whole frame can give us its own scalar, at the current time frame. So an STI made out of $\mathcal{I}$ values could have a number of rows equal to the number of columns in the video frame, and number of columns equal to the number of frames. E.g., in Fig. 4(a), we compare histograms for columns, so the STI is 160×100, for our example video. If we do the same for histograms of rows, then our STI is of size 120×100, as in Fig. 4(b).

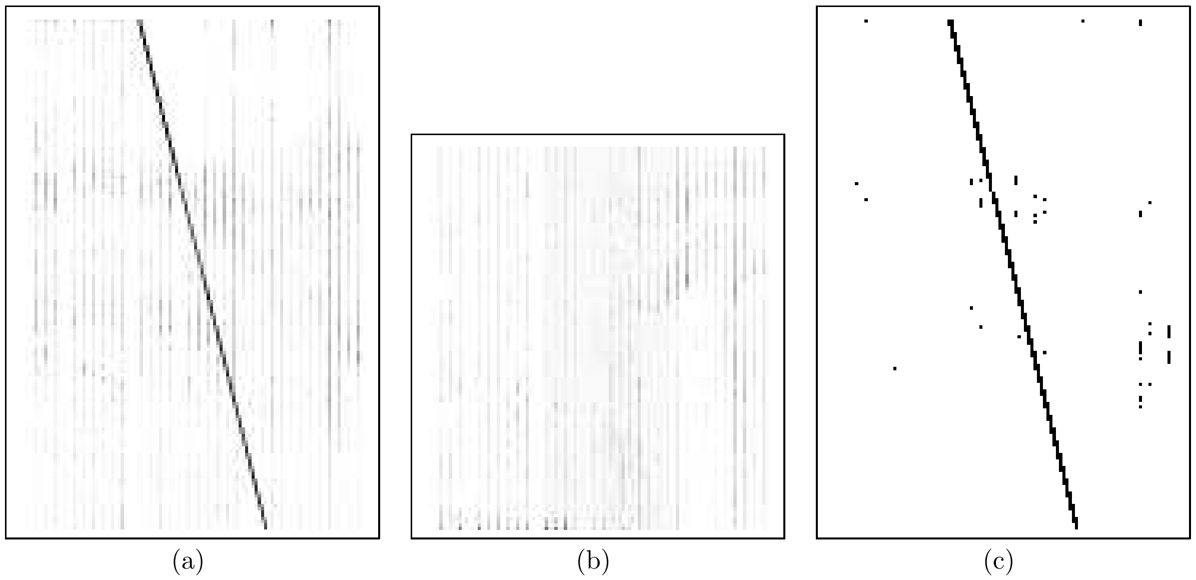


Figure 4: (a): STI made from histogram intersections of each column in each frame with the previous time instant; (b): And from rows. (c): A thresholded version of (a).

How the histogram intersection works is that, if one histogram is similar to the other, then $\mathcal{I} \simeq 1$. But if the histograms are different, then $\mathcal{I} \simeq 0$. So if we are comparing a video frame column to the same column at a previous instant, then we expect to have $\mathcal{I}$ about 1, and that's what we see except at a wipe, where the current column is from one video but the previous-time column is from another video.

The advantage of doing all this is that the output is much cleaner. If we wish to find a diagonal edge, then Fig. 4(a) provides a fairly clean sets of zero values down a straight diagonal, in a background of 1s. A "thresholded" version is shown in Fig. 4(c), where we display the boolean value ($\mathcal{I} > \tau$), with τ some high enough value (say, 0.7). Clearly, Fig. 4(c) provides a nice feature to search for in a video, if we wish to automatically find wipes.

2. Your job

Implement these ideas and test on videos. If you're really enthusiastic, try your code on video dissolves, as well. For faster processing, and also to remove some noise, the size of input frames can first be reduced to 32×32.

If you're truly over the top, then you'll likely be saying to yourself that histogram intersection is not the best one could do, in that to be compared, pixels have to be at roughly the same colour. But red is fairly like orange, so a reddish histogram bin should not give a zero intersection with an orange histogram bin.

Therefore, IBM has developed a system that uses a different definition of histogram-difference. Suppose we reshape each histogram into a long vector (e.g., a 49-element vector). Denote by $\boldsymbol{z}$ the *difference* between histograms: $\boldsymbol{z} = H_t - H_{t-1}$. Now the Euclidean distance between histograms is the square root of $\boldsymbol{z}^T \boldsymbol{z}$, where ^{T} means transpose.

But instead, let's interpose a matrix $\boldsymbol{A}$, defining a new squared distance D^2:

$$D^2 = \boldsymbol{z}^T \boldsymbol{A} \, \boldsymbol{z}$$

where $\boldsymbol{A}$ encapsulates nearness of colours. A form suggested is

$$A_{ij} = (1 - d_{ij}/d_{max})$$

with d_{ij} defined as a three-dimensional color difference and $d_{max} = max(d_{ij})$. E.g., we could use Euclidean distance between chromaticities $\{r, g\}$. Then $d_{max} = \sqrt{2}$.

3. What to submit

1. As a zipfile, submit code including documentation, along with a short (3-4 pages) written report summarizing what you have done for your project: What worked? What didn't? How would you like to extend your code? [And, what language did you use? Problems with libraries?]

   Not required, but as a different submission component it would be nice to include a video URL as well, if possible.

   Your project will be marked on effort and usefulness of the resulting product.

2. How hard was this project?
   State briefly how would you improve this project?
   Any ideas for a different project at the same difficulty level? Would you enjoy more difficulty? Less?