2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)

Efficient Video Data Structure and Compression Scheme for Fabric Wicking Phenomenon Studies

Chau-Wai Wong
Electrical and Computer Engineering
North Carolina State University, Raleigh, USA
chauwai.wong@ncsu.edu

Abstract

The physiological comfortableness of clothes is heavily attributed to the wicking phenomenon of fabrics. We have been working with textile scientists to automate a newly designed video-based test method that targets at analyzing the yarn-level wicking behaviors of fabrics. To access yarn-level wicking information, we need an efficient video data structure that allows the retrieval of the color information at each pixel along the time. In this paper, we propose using the volume of blocks as the basic storage unit rather than using the frame as most off-the-shelf video codecs do. The proposed data structure allows quick retrieval of time series data and can achieve a balance between the time overhead and memory overhead. We also propose a compression scheme specially designed for the proposed data structure. Experimental results show that the proposed data structure and compression scheme can store the video information in manageable file size while providing visual quality at a customized level.

1. Introduction

The physiological comfortableness of clothes is heavily attributed to the wicking phenomenon of fabrics. Two dominant factors of the wicking phenomenon are how fast and in what pattern the sweat or liquid transfers within fabrics. Traditionally, textile scientists rely on such wicking test methods as the vertical wicking test [1] and the gravimetric absorbency test [2] to assess the wicking performance of fabrics. However, these tests were not designed to reveal the yarn-level wicking behaviors and therefore are not adequate for assessing the performance of some specially designed fabrics made up of yarns of different wicking properties. Figure 1 shows four key frames of a video recorded for studying the wicking phenomenon for a blended fabric made up of hydrophobic and hydrophilic yarns. We call these videos wicking-performance videos.

Our colleagues from the textile college have designed a new wicking test method that targets at analyzing the yarn-level wicking behaviors. Apart from the innovation of the new method offered to the textile community, our interdisciplinary team agrees that it is beneficial to automate the detection of yarn-level wicking behaviors. Specifically, we have developed a preliminary wetting event timestamp detection algorithm [3] that works at the level of the individual time series of each pixel location of the wicking-performance video. By treating neighboring pixels independently, our detection algorithm avoids making mistakes that are commonly seen in algorithms exploiting spatial correlation for decision making. Our algorithm can precisely infer the yarn-level wicking behaviors even if neighboring pixels belong to different yarns.

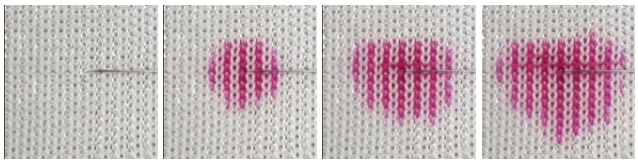

Figure 1: Four frames of a wicking-performance video for a fabric made up of yarns of different wicking properties.

To access the fabric video data at a specific pixel location along the time for the subsequent wetting event timestamp detection, one needs to decode every frame of the video and discard the color information at all locations other than the location of interest. If the video is of a resolution 1920×1080, the overhead in time is 2 million ($= 1920 \times 1080$) to 1. If memory usage is not a restriction, loading a 3-minute video will require 31 GBytes ($= 1920 \times 1080$ pixels/frame $\times$ 30 frames/sec $\times$ 180 secs $\times$ 3 bytes per pixel) memory, which is obviously nonscalable.

In this paper, we propose an efficient data structure for videos acquired for studying the wicking phenomenon of fabrics. In comparison to videos encoded by off-the-shelf video codecs that treat the frame as the basic storage unit, we propose using the volume of blocks of size N-by-N as the basic storage unit, where N is much smaller than the dimension of video frames. Using the proposed method, we can achieve a balance between the time overhead and memory overhead when converting videos in an off-the-shelf

978-1-7281-1198-8/19/$31.00 ©2019 IEEE
DOI 10.1109/MIPR.2019.00084

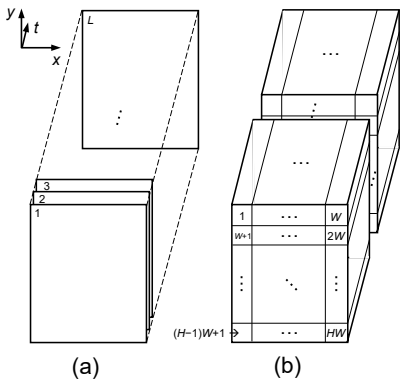

Figure 2: Comparison of (a) data structure of videos encoded by off-the-shelf codecs, and (b) proposed data structure. The basic storage unit for (a) is the frame and for (b) is the volume of blocks. Labeled numbers are indices of basic storage units.

format into the data structure designed for wicking performance studies. We also propose a compression scheme specially designed for the proposed data structure so that the video data can be stored in manageable file size while the visual quality can be maintained at a customized level.

2. Proposed Data Structure for Wicking Videos

2.1. Pros and Cons of Alternative Designs

Videos compressed by off-the-shelf codecs cannot be efficiently used for time series data extraction. Modern video codecs use the frame as the basic storage unit as shown in Figure 2(a). In order to extract the color information at a particular pixel location throughout the whole duration of the video, almost the whole video needs to be decoded. A negligible saving can be achieved by decoding only the specific block that contains the pixel of interest for those B frames that are not referenced by other frames. Below, we examine the pros and cons of three strategies for time series data extraction.

The first approach is to keep only the RGB values at the required location and discard the color information of other locations. As the frames are sequentially decoded following the picture order count (POC) within each group of pictures (GOP), the whole time series of RGB values will be available and stored to the hard disk once all GOPs are decoded. If the video is of a resolution 1920×1080, the overhead in time is about 2 million to 1, which is highly nondesirable.

The second approach improves the first one by not discarding information for other locations but appending the color data for each location to the file indexed by its spatial location. Using this approach, two million files are to be created on a hard disk and each of them will be appended the newest RGB color information once for each frame. This approach can create harsh working conditions for the hard disk by populating the file system with two million entries of the filenames and their respective pointers. It requires I/O operations in reading and writing for at least two million multiplies the number of frames of the video. To make the matter worse, as the hard disk alternating among different pixels, the data for a particular pixel will be stored in an interleaved way, which makes the retrieval of the time series highly inefficient.

Table 1: Complexity for different approaches for dumping time series for every pixel location. Complexity for decoder excluded.

Approach	Peak memory usage ($\times 3$ bytes)	Pixel loading overhead	I/O for output
#1	WH	$WH^{\triangledown}$	Low
#2	WH	1	High$^{\triangledown}$
#3	$WHL^{\triangledown}$	1	Low
Proposed	$WH\ell$	1 to N^2	Low

$^{\triangledown}$ indicates that the approach is extremely inefficient in this aspect.

The third approach tries to improve the previous two by first loading the whole video in memory and then saving a time series for every pixel location. However, storing a 3-min decoded video will require 31 GBytes memory. This is not only nonscalable in time, but also far from any recommended software engineering practice to consume 31 GBytes without absolute necessity.

We summarize the aforementioned approaches in three aspects in Table 1 where the frame size is W-by-H and the total number of frames is L. When the three approaches are compared side-by-side, it is revealed that each approach excels in two aspects but can be extremely inefficient in the remaining aspect. Our aim is to design a simple-to-implement data structure that is relatively efficient in all aspects.

2.2. Proposed Efficient Video Data Structure

We propose an efficient data structure by combining the advantages of the three aforementioned approaches. We start with Approach 3 by holding in memory only a short segment of the video to avoid memory overuse. In case each segment is of $\ell = 120$ frames long, the peak memory usage is only 712 MBytes ($= 1920 \times 1080$ pixels/frame $\times$ 120 frames $\times$ 3 bytes per pixel). We then apply the idea of Approach 2 by not discarding any information in the memory: we save the pixels the first time they are decoded. We also try to be considerate about the file system's capability by not creating a separate file for each pixel. Instead, we cut the video frame into square blocks of size N-by-N, where N can be 32, 64, etc., and append a time series of collocated blocks into a file indexed by the block location relative to the frame. Once all data of the current segment have been saved to the hard disk, we repeat the above procedure for the next segment. At the completion of the iteration, there are only $(1920/N) \times (1080/N)$ data files managed by the file system. In this proposed method, we consider the time series/volume of blocks as the basic storage unit. Figure 2(b) illustrates how the proposed data structure is different from that of an off-the-shelf encoded video shown in Figure 2(a).

When a time series at a specific location is inquired, one should load the volume of blocks that covers the requested

location. By going through all the blocks along the time, the time series of the requested location can be reconstructed. Since we store neighboring pixels together as a block, the overhead in loading time is N^2 when loading the time series of only one location. Such overhead may be reduced by simultaneously inquiring all time series located within the same block.

To design the data structure within each file of the volume of blocks, we choose to store it in a block-by-block way instead of in a series-by-series way. This data structure can facilitate further data compression because we observe that the redundancy along the time is stronger than that spatially. Consider a typical fabric video acquired by a still camera. The scene i) contains almost no motion except due to minor camera shaking, ii) has intensity changes only at the frame level due to the contrast adjustment; and iii) has some acquisition noise. This means that all frames are roughly aligned and visually similar, and therefore have strong redundancy in time. On the contrary, if one picks a pair of two pixel locations from a block, it is possible that they correspond to yarns at different wicking stages, and therefore, has very different patterns in intensity. Arranging data block-by-block within each file could make data compression more efficient.

3. Data Structure Compression

Once the wicking video is organized into the proposed data structure, i.e., the whole volume of full-size frames are broken into shorter volumes of blocks, each short volume can be further compressed to save the storage. Although one can take an off-the-shelf video encoder to compress the volumes of blocks, it may be overkill as the fabric video has almost still scene and the information needs to be specially encoded are the frame-level intensity change and various types of noises.

We propose a simple regression-based encoding method that makes use of off-the-shelf lossy image compression tool, e.g., JPEG, and lossless image/text compression tools, e.g., PNG, LZMA, Deflate to address the need for volume compression. The block diagram of the proposed encoder is shown in Figure 3. Specifically, we exploit the redundancy over a short period of time as discussed in Section 2.2: the scene of wicking video is almost still except the color may be scaled and biased at the frame level. We propose to use the averaged block of the current volume, $\mathbf{x}_{\text{ave}} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{y}_k$, as the reference to predict each block $\mathbf{y}_k \in \mathbb{R}^{N\times N}$, where K is the number of blocks/frames in this volume. Note that in order to avoid the mismatch between the encoder and decoder, any reference to be used for prediction should pass through a lossy reconstruction channel. In our example, we use JPEG to reduce the file size of the reference block as much as possible. The JPEG binary output data $\mathbf{b}_{\text{ave}}$ and reconstructed averaged block $\mathbf{x}_{\text{ave-recon}}$

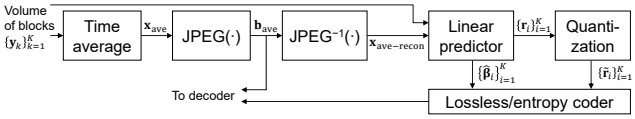

Figure 3: The block diagram of the compression scheme for the proposed video data structure.

are shown as follows:

$$\mathbf{b}_{\text{ave}} = \text{JPEG}(\mathbf{x}_{\text{ave}}),\ \ \mathbf{x}_{\text{ave-recon}} = \text{JPEG}^{-1}(\mathbf{b}_{\text{ave}}), \qquad (1)$$

where JPEG$(\cdot)$ and JPEG$^{-1}(\cdot)$ are JPEG encoders and decoders, respectively. The relationship between the ith block $\mathbf{y}_i$ and reconstructed averaged block $\mathbf{x}_{\text{ave-recon}}$ are modeled as follows:

$$\mathbf{y}_i = \beta_{i1}\mathbf{x}_{\text{ave-recon}} + \beta_{i0} + \mathbf{e}_i,\ \ i = 1, \cdots, K, \qquad (2)$$

where β_{i1} and β_{i0} are unknown modeling parameters that corresponds to scaling and bias, and $\mathbf{e}_i$ is a noise vector with $\mathbb{E}[\mathbf{e}_i] = \mathbf{0}$. The parameters can be estimated using least-squares, namely, $\{(\hat{\beta}_{i0}, \hat{\beta}_{i1})^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}_i\}_{i=1}^K$, where $\mathbf{X} = (\mathbb{1}, \mathbf{x}_{\text{ave-recon}})$ and $\mathbb{1}$ is an all-one vector. Note the computational advantage of the regression approach: for each volume, $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ needs to be calculated only once. Hence, the residue of the ith block, $\mathbf{r}_i$, after the block prediction is

$$\mathbf{r}_i = \mathbf{y}_i - \hat{\beta}_{i1}\mathbf{x}_{\text{ave-recon}} - \hat{\beta}_{i0} = \big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T - \mathbf{I}\big)\mathbf{y}_i. \quad (3)$$

To summarize, for lossless encoding, the following information should be sent to the decoder: i) the JPEG binary output data, $\mathbf{b}_{\text{ave}}$, ii) the block prediction coefficients, $\{(\hat{\beta}_{i0}, \hat{\beta}_{i1})^T\}_{i=1}^K$, and iii) the block residues $\{\mathbf{r}_i\}_{i=1}^K$. Note that the coefficients and residues should be sent in losslessly compressed formats.

One can choose to further reduce the file size using lossy compression at the cost of the introducing reconstruction error. Given that the block residues dominate the encoded bitstream in size, doing lossy coding on them should provide the most reduction in file size. We apply a uniform quantizer with quantization step size q to each element in the residue. The codeword for entropy coding/lossless compression $\tilde{\mathbf{r}}_i$ and the reconstructed residue $\hat{\mathbf{r}}_i$ are defined as

$$\tilde{\mathbf{r}}_i = \text{round}(\mathbf{r}_i/q),\ \text{and}\ \ \hat{\mathbf{r}}_i = q\tilde{\mathbf{r}}_i. \qquad (4)$$

4. Experimental Results and Discussions

We use for experiments four wicking-performance videos of the length 315, 127, 142, and 138 seconds, respectively. The videos were taken by iPhone X fixed on a tripod and encoded using H.265/HEVC. Their original resolution was 1920×1088 and they were downsampled by a factor of 4 to avoid oversampling and to boost the signal strength. Figure 1 shows four key frames of the 127-sec video in which a needle was injecting pink liquid to one

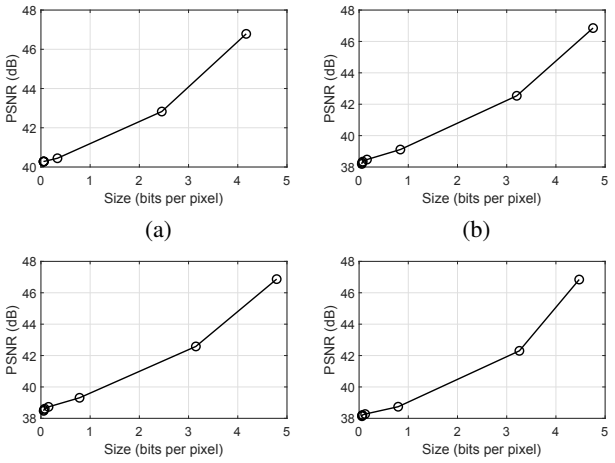

Figure 4: Rate-distortion curves for videos (a) #38, (b) #39, (c) #40, and (d) #41. The quantization step size q used from left to right are $100, 50, 35, 25, 15, 7,$ and 4. The clustered points at the bottom left corners imply that most residues are smaller than 25.

yarn, and the liquid subsequently transferred within yarns and between yarns. It is visible from all four videos that the injection needle may slightly vibrate. Fabrics after absorbing the liquid may also have some nonrigid distortion in shape. The camera may have minor shake, and the focus of the camera may blur from time to time. At the beginning and toward the end of the video, there can exist significant camera motions.

We developed the proposed video data structure and compression algorithm in Matlab. We used Matlab's built-in JPEG compression algorithm for encoding the reference blocks. We used 7z's LZMA implementation to losslessly encode the quantized residues and prediction coefficients. We set the length of each volume of blocks to $\ell = 240$. We measure the performance of the rate-distortion behavior of the proposed scheme using the peak signal-to-noise ratio (PSNR) and the filesize in bits per pixel using quantization step size $q \in \{4, 7, 15, 25, 35, 50, 100\}$ that has the same unit as the pixel intensity ranging from 0 to 255. The larger the q is, the more distortion the reconstructed video will be. Figure 4 reveals that when the quantizer is finer, i.e., q is small, PSNR can easily exceed 40 dB. When the quantizer is coarser, i.e., q is large, PSNR drops until it reaches a "lower bound," as is shown by the clustered rate-distortion operational points at the bottom left corners of the plots. This implies that most residues are smaller than $q = 25$, and the quality of the reconstructed video is mainly controlled by the quality of reference blocks and the predictor coefficients. It is revealed by the plots that a 24 bits per pixel video may be represented by much less than 1 bit per pixel while maintaining the PSNR above 38 dB.

We are also interested in the size of the compressed files at different lossy coding quality. Figure 5 shows for video #39 the relative and absolute size of compressed files as the

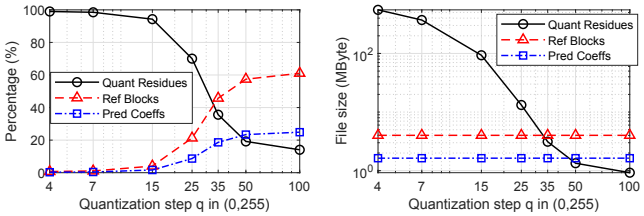

Figure 5: (a) Relative and (b) absolute size as a function of the quantization step q for video #39. Both absolute and percentage size of quantized residues reduce as quantizer becomes coarser.

quantization step size q varies. It is revealed that both absolute and percentage size of quantized residues reduce as quantizer becomes coarser. The filesize at PSNR = 46 dB is no more than 600 MBytes, and the filesize at PSNR = 39 dB is no more than 100 MBytes, which is a significant saving than storing time series directly. The crossing pattern in Figure 5(a) implies that future work should focus on improving the compression ratio of the quantized residues if high PSNR videos are preferred by the textile scientists. However, if low PSNR videos are adequate for wicking phenomenon studies, future work should improve the compression ratio of reference blocks and prediction coefficients.

Note that achieving a high compression ratio is not the only design goal for this compression scheme; development time, maintenance costs, royalty rate are all considerations when automating the wicking phenomenon studies.

5. Conclusion

In this paper, we proposed an efficient data structure for videos acquired for studying the wicking phenomenon for fabrics. We proposed using the volume of blocks as the basic storage unit instead of using the frame as most off-the-shelf codecs do. The proposed data structure allows quick retrieval of a time series and can achieve a balance between the time and memory overhead. We also proposed a compression scheme that was designed for the proposed data structure. Experimental results showed that the overall system can encode the video in manageable file size while providing satisfactory visual quality at different levels.

Acknowledgment: We thank Ms. Hey-sang Kim, Dr. Stephen Michielsen, and Dr. Emiel DenHartog for providing the background knowledge and the sample videos.

References

[1] *Vertical Wicking of Textiles*, AATCC Std. TM197, 2013.

[2] B. Miller and I. Tyomkin, "Spontaneous transplanar uptake of liquids by fabrics," *Textile Research Journal*, vol. 54, no. 11, pp. 706–712, Nov. 1984.

[3] X. Liu and C.-W. Wong, "Video-based wetting detection for blended fabrics," in *IEEE International Conference on Image Processing (ICIP'19)*, Brighton, UK, May 2019, submitted.