IEICE TRANS. INF. & SYST., VOL.E101–D, NO.2 FEBRUARY 2018

LETTER

Painterly Morphing Effects for Mobile Smart Devices

SungIk CHO[†], ***Nonmember and*** **JungHyun HAN**[†a)], ***Member***

SUMMARY This paper proposes a painterly morphing algorithm for mobile smart devices, where each frame in the morphing sequence looks like an oil-painted picture with brush strokes. It can be presented, for example, during the transition between the main screen and a specific application screen. For this, a novel dissimilarity function and acceleration data structures are developed. The experimental results show that the algorithm produces visually stunning effects at an interactive time.

key words: *mobile smart devices, user interface, interactive morphing, painterly effect*

1. Introduction

With the growing use of mobile devices such as smartphones and smartpads, there has been an increasing demand on *interactive special effects* that can be executed on such mobile devices. A good candidate would be *image morphing* [1]. It can be presented, for example, during the transition from a specific application such as photo viewer back to the main screen. We propose a *painterly morphing* method, where each frame in the morphing sequence looks like an oil-painted picture with brush strokes.

2. Fast Morphing Based on Patch Similarities

Based upon the framework proposed by *regenerative morphing* [2] and *image melding* [3], our morphing algorithm takes as input two source images, S^1 and S^2, initializes a sequence of in-between target images, T^ks, by *cross-dissolving* S^1 and S^2, and then updates T^ks such that S^1, T^ks, and S^2 form a painterly morphing sequence.

2.1 Dissimilarity Measure

In our algorithm, similarity or equivalently dissimilarity between images plays a key role. The function frequently used for evaluating dissimilarity is the *sum of squared differences* (SSD). Given two images, P and Q, of the same size, $SSD(P, Q)$ is defined as follows:

$$SSD(P,Q) = \sum_{i=1}^{m}(p_i - q_i)^2 = mE(P^2) + mE(Q^2) - 2mE(P)E(Q) - 2\sum_{i=1}^{m}[(E(P) - p_i)(E(Q) - q_i)] \quad (1)$$

where m is the number of pixels in an image, p_i and q_i denote the i-th pixels in P and Q, respectively, $E(P)$ denotes the mean of the lightness values of all pixels in P, $E(P^2)$ is the mean of the squared lightness values in P, and $E(Q)$ and $E(Q^2)$ are defined in the same manner. Equation (1) is derived using the theorem, $E(P^2) = E(P)^2 + \sigma^2(P)$, where $\sigma^2(P)$ is the *variance* of P.

The last line of Eq. (1) reflects the *covariance* of two images. We delete the covariance term to generate a painterly morphing effect. The result is then divided by m to define our dissimilarity function:

$$D(P,Q) = E(P^2) + E(Q^2) - 2E(P)E(Q) \quad (2)$$

2.2 Morphing with Similar Patch Tables

Given an image, consider its square sub-area composed of $n \times n$ pixels, which we call a *patch*. If the resolution of an image is $M \times N$ and that of a patch is $n \times n$, we have $(M - n + 1) \times (N - n + 1)$ patches in total. For each of such patches, the "mean" and "mean of squares" are computed and stored in what we call a *patch table*. Each of S^1, S^2, and T^ks has its own patch table, as shown in Fig. 1. The patch

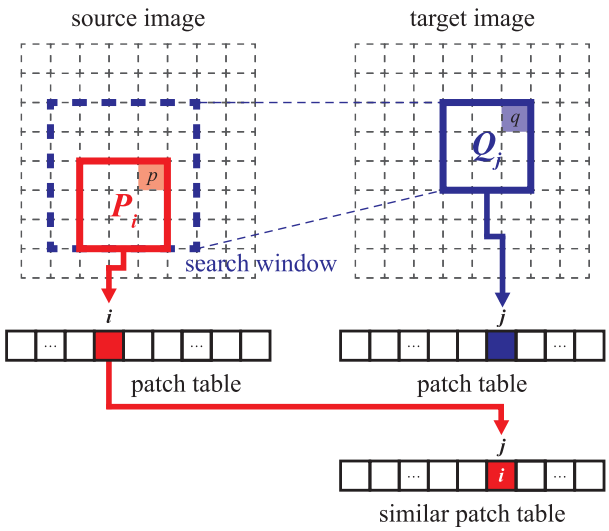

Fig. 1 The similar patch table is filled using the patch tables of the source and target images and is used later to update the target image.

Manuscript received September 12, 2017.
Manuscript revised October 24, 2017.
Manuscript publicized November 6, 2017.
[†]The authors are with the Korea University, Korea.
a) E-mail: jhan@korea.ac.kr
DOI: 10.1587/transinf.2017EDL8200

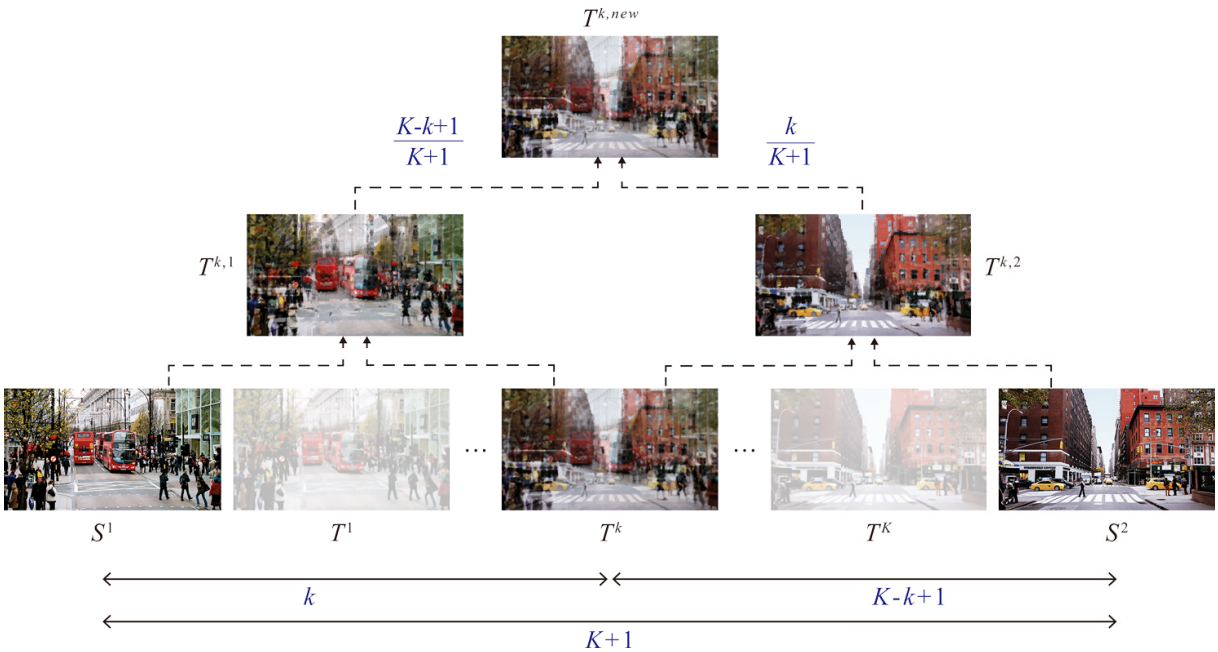

Fig. 2 Target image update. For an initial target image, T^k, two temporary images, $T^{k,1}$ and $T^{k,2}$, are computed using S^1 and S^2, respectively. Then, $T^{k,new}$ is obtained by linearly interpolating $T^{k,1}$ and $T^{k,2}$. The distance between S^1 and T^k and that between T^k and S^2 are used to determine the weights for interpolation. Finally, T^k is replaced by $T^{k,new}$.

Fig. 3 Iteration and blending (Another source image S^2 is omitted just for saving the space).

table for T^k is created using its current content, which was obtained by cross-dissolving S^1 and S^2.

For each patch of a target image T^k, S^1 is searched for the *most similar patch* using Eq. (2). It is recorded in the so-called *similar patch table* (SPT), which has the same dimension as the patch table. In order to achieve a smooth transition across the morphing sequence, searching is restricted to a sub-area in S^1, which is called the *search window*. S^2 is also searched in the same manner. Then, a target image has two SPTs, one for S^1 (SPT^1) and the other for S^2 (SPT^2).

A target image T^k is updated by computing new colors for all of its pixels. In updating a pixel q of T^k, $n \times n$ patches sharing q (each denoted by Q_j) are involved. See Fig. 1 for an example of Q_j. For each Q_j, the most similar patch (P_i) is retrieved from SPT^1, and P_i's pixel p which has the same within-patch location as q is extracted. Then, the mean of $n \times n$ colors is assigned to q. When all pixels in T^k are updated, a temporary image $T^{k,1}$ is generated. The same process is repeated using SPT^2 to generate another temporary image $T^{k,2}$.

Algorithm 1: Pseudo code for morphing

```
Make the patch tables of S^1 and S^2
foreach target image {T^k}_1^K do
    Make the patch table of T^k
end
foreach target image {T^k}_1^K do
    Make SPT^1 using the patch tables of T^k & S^1
    Make SPT^2 using the patch tables of T^k & S^2
    T^{k,1} ← Color accumulation using SPT^1
    T^{k,2} ← Color accumulation using SPT^2
    T^{k,new} = ((K-k+1)/(K+1)) T^{k,1} + (k/(K+1)) T^{k,2}
end
```

$T^{k,1}$ and $T^{k,2}$ are linearly interpolated to determine $T^{k,new}$ using the relative position of T^k between S^1 and S^2, as illustrated in Fig. 2. $T^{k,new}$ replaces T^k. **Algorithm 1** shows the pseudo code for the morphing process.

2.3 Blending for Painterly Stylization

Suppose that, in Fig. 2, T^k is replaced by the updated target image $T^{k,new}$ and then **Algorithm 1** is executed again on the sequence of the "updated" target images. Then, the resulting target images would look like being drawn using a "larger" brush. This kind of iteration can be continued. The more iterations, the larger brush effect, as shown in Fig. 3, where T_i^k denotes the target image updated at the i-th iteration. Observe how the image changes as i increases. T_2^k looks as if it were drawn by a brush larger than the one used for T_1^k. A similar observation can be made between T_3^k and T_2^k.

In general, an oil-painted picture is created by starting

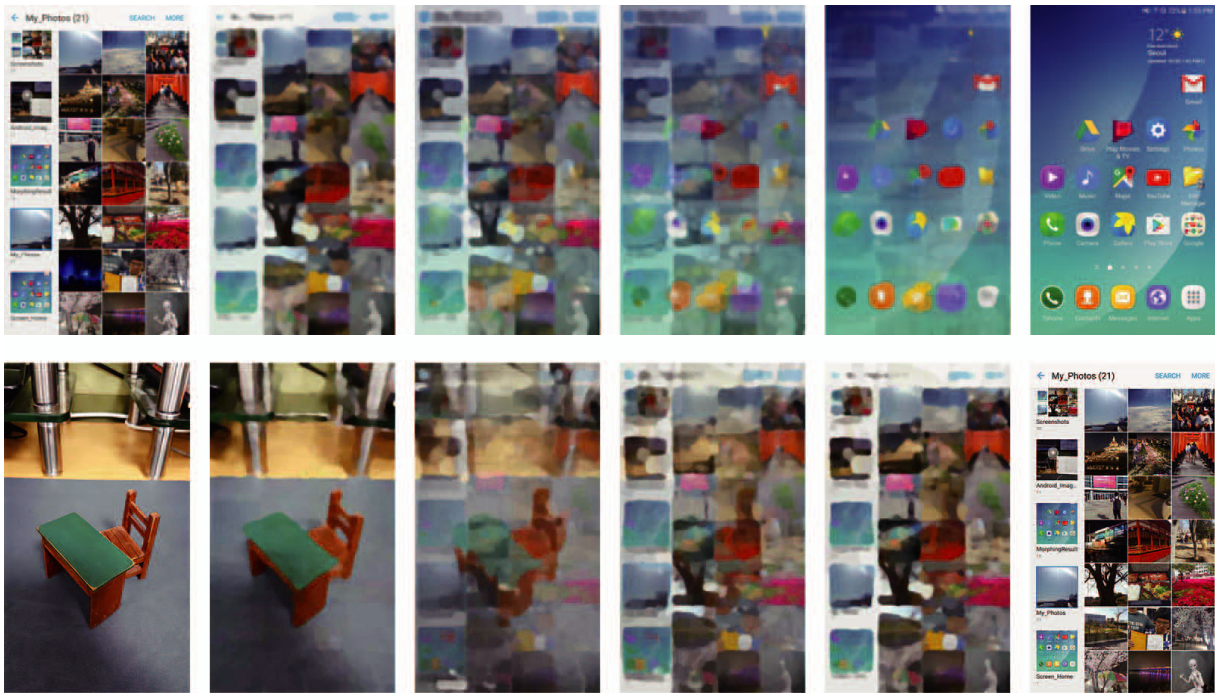

Fig. 4 Morphing sequences generated in a smartphone.

with a large brush and then by adding details progressively with smaller brushes [4]. This painting technique can be simulated if we save the iteratively-updated target images and blend them. For example, in Fig. 3, T_2^k would first be blended with T_3^k, and then T_1^k would be blended with the result. The first step of blending T_2^k with T_3^k is implemented by taking T_2^k as the target and T_3^k as the source, i.e., by updating T_2^k using its similarity to T_3^k.

Once T_2^k is updated by T_3^k, it is used for updating T_1^k through the same procedure presented above. The right most image in Fig. 3 shows the final result of blending. Observe how it differs from T_i^k.

3. Experimental Results

The proposed algorithm has been implemented on Samsung Galaxy Note 5. The program is written in C++ using the Java Native Interface (JNI) with OpenMP and OpenCV. The source images, S^1 and S^2, are originally in QHD (Quad High Definition) with the resolution of 2560×1440. They are down-sampled to 320×180 resolution, and the target images are created with the same resolution. A patch is composed of 3×3 pixels, and the number of target images is set to 10 in a morphing sequence. Three iterations were made for blending the target images. Figure 4 shows morphing examples.

Figure 5 compares our method with simple linear blending and regenerative morphing [2]. In general, image morphing is defined as a combination of cross-dissolve and warping, but linear blending presented in Fig. 5 (a) shows the cross-dissolve effect only. On the other hand, regenerative morphing produces a comparable result (Fig. 5 (b)) but the differences with our method should be clarified. First, it does not generate the painterly morphing effect. Second, it is not implemented in interactive time. The regenerative morphing method was implemented on Matlab with 20 target images, each with 400×400 resolution, and was reported to take "several tens of minutes" in a mid-range PC [2]. In contrast, the entire time consumed by our method is 1.433s on average in a mid-range mobile device.

4. Conclusion

This paper proposed to create a painterly morphing sequence on mobile smart devices, where each frame in the sequence looks like an oil-painted picture with brush strokes. For this, a novel dissimilarity measure function is devised and the ingredients of the function are computed and stored in patch tables. Then, similar patch tables are created using them, which enable interactive similarity-based image morphing. The experimental results showed that the algorithm produces novel styles of morphing with visually stunning effects on mobile devices at an interactive time.

The user interface in smartTVs is important [5], and it may benefit from our painterly morphing, e.g., the special effects can be presented during the transition between the main screen and an arbitrary screen. Unfortunately, the computing power of the state-of-the-art smartTV is far weaker than that of smartphone or smartpad. A more effective solution to get closer to the interactive-time performance in smartTV would be to restrict the target image update to the region of interest (ROI). Our future work will be done along this direction.

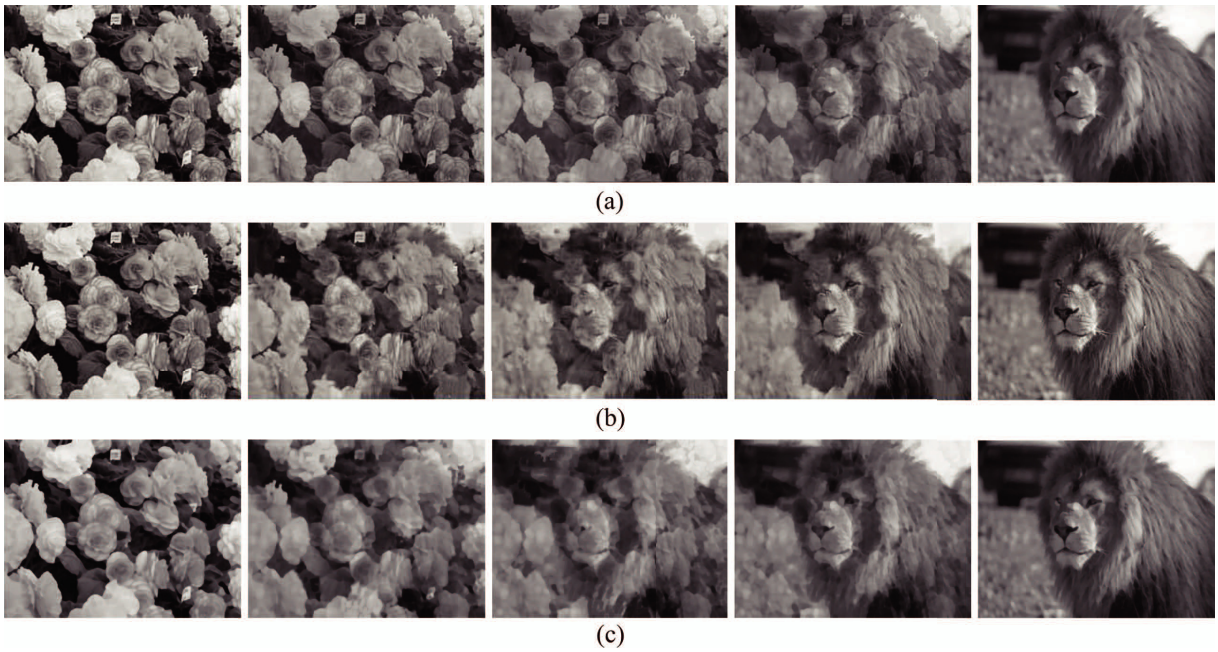

Fig. 5 Comparison with other methods. (a) Linear blending. (b) Regenerative morphing†. (c) The proposed method.

Acknowledgments

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (NRF-2017M3C4A7066316).

References

[1] M. Haseyama, M. Takizawa, and T. Yamamoto, "Video frame interpolation by image morphing including fully automatic correspondence setting," IEICE Trans. Inf. & Syst., vol.92, no.10, pp.2163–2166, Oct. 2009.

[2] E. Shechtman, A. Rav-Acha, M. Irani, and S. Seitz, "Regenerative morphing," 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.615–622, 2010.

[3] S. Darabi, E. Shechtman, C. Barnes, D.B. Goldman, and P. Sen, "Image melding: Combining inconsistent images using patch-based synthesis.," ACM Trans. Graph., vol.31, no.4, Article No. 82, 2012.

[4] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," Proc. 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '98, pp.453–460, 1998.

[5] T. Kim, S. Choi, and H. Bahn, "A personalized interface for supporting multi-users in smart TVs," IEEE Trans. Consum. Electron., vol.62, no.3, pp.310–315, 2016.

†© 2010 IEEE. Reprinted, with permission, from E. Shechtman, A. Rav-Acha, M. Irani and S. Seitz, "Regenerative morphing," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, 2010, pp.615–622.