2008－CG－130 (15)
2008／2／19

アニメーションの彩色における分類ベースの領域マッチング

Pablo Garcia Trigo† Henry Johan‡ 今給黎 隆† 西田 友是†
東京大学† Nanyang Technological University‡

手描きアニメーションの彩色作業を補助するための分類ベースのマッチング手法を提案する．入力された各フレームを領域分割し，各領域の形状と位相情報の特徴を調べ，フレーム間の各領域をマッチングする．マッチングの失敗には，利用者による対話的な修正で対応する．提案法の特徴は，領域のマッチング結果の関連をチェーンと呼ばれるグループに分類し、アニメーションに沿って領域の特徴の変化を調べることにある．前のフレームまでのマッチング結果を利用することで，マッチング精度を向上させ，後のフレームになるほど失敗するマッチングの量を減らすことができる．結果として，正確な彩色を行うための作業を短縮することができる．

Classification-based region matching for 2D animation coloring

Pablo Garcia Trigo† Henry Johan‡ Takashi Imagire† Tomoyuki Nishita†
The University of Tokyo† Nanyang Technological University‡

We propose a classification-based method for assisting the coloring process of 2D (hand-drawn) animated cartoons. It segments input frames extracting the shape and topological features of each region, and then matches them between frames, allowing the user to fix coloring mistakes interactively. Its main contribution is that the matching algorithm classifies the matched regions into groups called "chains" and uses them for seeing how the regions' features vary along the animation. Thanks to that, the matching accuracy is improved and the matching mistakes are reduced, thus reducing the total effort needed until having a correctly colored cartoon.

1. Introduction

Traditional 2D animated cartoons are created by drawing and coloring each frame manually. Compared to other ways of making animations, it is a very flexible means, but it is also a very time-consuming process.

The traditional 2D animation pipeline consists of the following steps [1]:

1. The main artists draw the concept storyboard.
2. The senior artists draw the key frames (lines only).
3. Secondary artists draw the inbetweenings (the frames between the key frames).
4. Other artists color all the drawn frames.
5. Background, music and dialogs are added.

Usually steps 3 and 4 are the most time-consuming, even when using commercial software like Retas or Animo, (approximately to 60% of the time of the whole process [2]) and previous research has concentrated on automating them. For automating the coloring step the usual approach has been the following:

> Input: All the key frames and inbetweenings drawn after steps 2 and 3 of the animation pipeline. All of them consisting of lines only.
>
> Computer-assisted coloring process:
>
> 1. Segment each input frame into its closed regions.
> 2. Analyze each region in each frame and extract its shape and topological features.
> 3. Matching: For all frames, identify what regions in one frame correspond to what regions in the other frames using the features found in the previous step and paint them with the same color.
> 4. Check if there are regions wrongly colored and correct them manually.

This paper introduces a classification-based interactive method for assisting the coloring process. Each time a region is matched to another, they are classified as belonging to the same group of regions. We call these groups "chains" of regions. During the matching, we use not only the features of the region currently being matched but also the features of the previous regions in the chain. Having the chains, we know how regions' features vary during the animation and that helps us reduce the number of wrongly matched regions in the matching step (third step of the computer-assisted coloring process). Consequently, more regions are automatically correctly colored and we reduce the workload.

2. Related work

Previous methods have concentrated in automating completely the coloring process with no user intervention at all. Methods like [3] specialize in the colorization of old grayscale cartoons. The levels of gray serve as a hint for the colorization, but nowadays animation pipeline produces frames in black and white only. Thus, this solution is not applicable.

Having only lines and the possible lack of coherence between frames can make the matching very difficult. For addressing that, methods like [2] [4] and [5] increase the matching accuracy by using master frames as a reference, building a hierarchy of regions or inserting skeletons.

While the above approaches can color successfully certain kinds of animations, they may achieve bad results in others. That is specially true in [4] [5] and [6] if the topology of the regions suffers many changes. This leaves the artist with the task of correcting many wrongly colored regions manually in the end. This is due to the fact that coloring mistakes in one frame usually propagate to the rest of the frames. Even more, those mistakes prevent other regions from being correctly colored, thus creating more mistakes.

This paper proposes a different approach. Instead of aiming for the complete automation of the coloring process, it recognizes that matching mistakes are very difficult to avoid in all cases and introduces user interactivity for fixing matching mistakes as soon as they appear. On top of this, it builds chains of regions along the animation. This helps to know how regions change and increases the accuracy of the matching.

We work directly with raster images (bitmaps) as in [7] and [8]. Taking the bitmaps and vectorizing them is done in [9] and [10], but we decided that it was not necessary for our method. Also, we do not apply any line thinning algorithm as in [2]. Line thickness can be a distinctive characteristic in some animation styles and thinning them could affect the dominant points or even the neighborhood relations.

3. Proposed method

3.1. General overview

Our coloring algorithm goes as follows:

1. Scan the hand-drawn frames.
2. Apply filters to reduce noise and holes.
3. Segment each input frame into its closed regions.
4. For each region, extract its shape and topological features (Section 3.2).
5. Interactive matching (Section 3.5).
   During the interactive matching, group the regions into chains to improve the matching accuracy (Sections 3.3 and 3.4).

3.2. Region features

For each region we extract the following features:

1. Shape features:
   a. Area.
   b. Dominant Points.
2. Topological features:
   a. Position.
   b. Neighbor regions.

1.a) The area is the number of pixels that make up that region. The background is usually the region with the biggest area, although not necessarily.

1.b) Dominant Points are points in a boundary that have a high curvature. Additionally, they may have other features. There is not a single definition and they are also called Character Points. In [11] the k-cosine is used for finding them. We use a simple method that has proved good enough for our samples: To classify whether a pixel p that is in the border of a region is a dominant point or not, we count inside a small square centered at p the amount of pixels of the region that fall in the square. If they are 75% or more of the area of the square, that pixel is considered to have a high curvature and is classified as Dominant Point (see Figure 1). The same applies if they are 25% or less. This can be tuned for each specific animation.

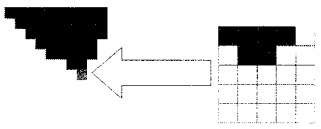

Figure 1: Example of a Dominant Point. The arrow points at the tip of a stroke. The tip is a Dominant Point. It is calculated with a square of 5x5 pixels. 6 out of 24 pixels are in black (the stroke drawn by the artist) and 18 are in white (the region to be colored).

2.a) The position of a given region r is reported as its centroid, where the centroid is the arithmetic mean of all pixels of r (see Figure 2). Centroids may not necessarily fall in the middle of a region, or not even inside.

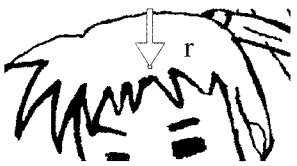

Figure 2: Centroid of a region r. For example, if r is the hair region, its centroid is at the tip of the arrow.

2.b) For each region we register its neighbor regions. They can be found expanding the borders of the regions outwards up to a certain distance. This can be tuned according to the thickness of the lines.

3.3. The matching algorithm

We can think of the matching process as a process of making chains of regions. It starts with a given region *r1*, let us call it source region, and tries to find its corresponding region *r2* in the next frame (see Figure 3). If found, *r1* and *r2* are linked, *r2* becomes the new source region and the process is repeated. When it cannot find its corresponding region in the next frame the chain ends.

Thus, we call "to match a given region *r*" the process of finding all the regions that correspond to *r* in the following frames, and the result is a chain of regions, started by *r* (see Figure 4).

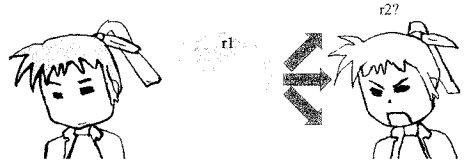

Figure 3: r1 is the hair region in the first frame at the left. We want to find its corresponding region r2 in the second frame (at the right). For finding it, we will compare r1 to all regions in the second frame.

Figure 4: The chain of regions corresponding to the hair. They go from the first frame (the leftmost picture) to the last frame (the rightmost).

We have implemented a forward matching algorithm. We start in the first frame and we try to match all its regions. When done, we look at the next frame, and do the same with all the regions that are still unmatched. We continue until we reach the last frame. Inside each frame, we start matching from the biggest region to the smallest. Empirically, we found this order to give good results.

3.4. The comparison function

How do we find the correspondence of one region in the following frames? This is, how do we know that region *r1* in the frame *f* is the region *r2* in the frame (*f* +1)? We use a comparison function *comp*. It accepts two regions as input and returns a score indicating how probable is that those two regions correspond to each other. A score is always a number between 0 and 1, 0 in the case of total dissimilarity and 1 when the features are identical.

For a given region *r1* in a frame (see Figure 3), we run *comp* against *r1* and all the regions in the next frame that have not been matched yet. Then, we pick up the region with the highest score as *r1*'s correspondence. Sometimes we cannot find a correspondence because all the regions in the next frame have already been matched.

comp is made of four sub functions that compare each of the features in Section 3.2 and return a score.

Then, the score of each sub function is multiplied by a weight *w* that increases or reduces the importance of that feature. Finally, all the weighted scores are summed to make up the final score that *comp* will return.

$$\begin{aligned} comp(r1, r2) = {} & \\ & comp_{area}(r1, r2) * w_{area} + \\ & comp_{position}(r1, r2) * w_{position} + \\ & comp_{DominantPoints}(r1, r2) * w_{DominantPoints} + \\ & comp_{neighbors}(r1, r2) * w_{neighbors} \ . \end{aligned}$$

$comp_{area}$ compares the area of two regions subtracting their areas and $comp_{position}$ compares the position calculating the distance between their centroids. For comparing the Dominant Points of two regions *r1* and *r2*, $comp_{DominantPoints}$ calculates the average of all the Dominant Points of each region and then calculates their difference. Similarly, for comparing the neighbors, $comp_{neighbors}$ calculates the average of their centroids and returns their difference. The string encoding used in [8] is more accurate, although ours is faster.

$$\begin{aligned} comp_{area}(r1, r2) &= |r1_{area} - r2_{area}| \ , \\ comp_{position}(r1, r2) &= distance(r1_{centroid}, r2_{centroid}) \ , \\ comp_{DominantPoints} &= |Average(r1_{AllDominantPoints}) - \\ & \qquad Average(r2_{AllDominantPoints})| \ , \\ comp_{neighbors} &= |Average(all\ neighbors\ of\ r1_{centroid}) - \\ & \qquad Average(all\ neighbors\ of\ r2_{centroid})| \ . \end{aligned}$$

Regarding the weights, they are recalculated according to the regions in the chain to reflect the variation of the features along it: for each region in the chain, we compare it with its immediate predecessor. We compare the four features separately and store the minimum and maximum scores of each feature in the chain. By doing so, we know how much does a feature vary. Features that vary a lot are not reliable and we assign them a low weight. Conversely, features that do not change almost are given a high weight. Empirically, we found that the following values worked well for our examples: A weight of 4 if the feature varied less than 10%, a weight of 3 if it varied less than 30% and a weight of 2 if it varied less than 50%. A weight of 1 otherwise.

Taking into account the previous regions in the chain enhances the accuracy, but it is not possible to make *comp* work correctly for all cases and eventually it will not return the highest score for the corresponding region, becoming a wrong match.

We can always fine-tune *comp*'s weight calculations to the particular animation, but in this paper, we have opted to make the matching process interactive and ask the user to supervise the matchings as they are done. Although this means more work for the user, it means that matching mistakes are corrected as early as they appear and, in the end, it is less work than fixing all the mistakes of all the frames after an automatic matching with mistakes.

3.5. Interactive matching

The user interface consists of a set of panels that display the frames to be matched. We can have as many as we want, and, if the user has enough monitors, he/she should be able to see all the frames of the animation at once. (Although that could be impractical if the animation is too long).

Usually, we will use a smaller working set of frames, e.g. the first five frames, and do the matching in these five frames only. Once all regions in the working set have been matched, it proceeds to the next five ones, and so on until the last frame. As long as matches are found, chains grow along. The smallest working set would be two frames: A frame and its following frame. Our example animation has five frames, so we can see them all at once while performing the matching (see Figure 5).

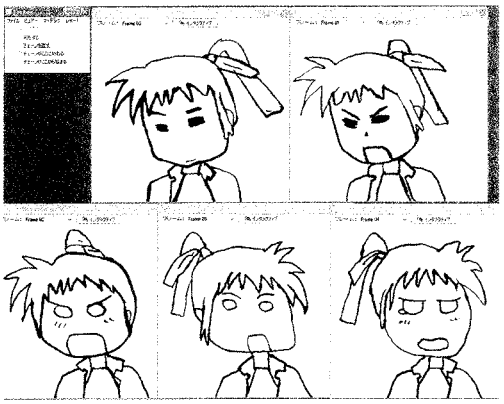

Figure 5: The user interface showing five frames.

Initially, the user has only to click the "Interactive matching" button on top of the panels (see Figure 6) or, more conveniently, just press the space bar in the keyboard. At each press, the system executes one step of the algorithm described in Section 3.3: it picks up the first non-matched region of the working set, matches it and shows the result on screen. The matched regions appear colored with the same color. A random color is used for distinguishing each chain. Figures 7, 8 and 9 show the first two steps of the matching process.

For fixing matching mistakes the toolbox (the upper left dialog in the user interface, Figure 6) allows us to fix a chain, indicate that a chain ends and indicate that a chain starts.

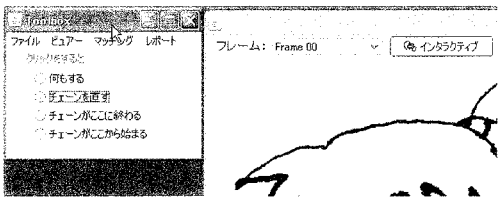

Figure 6: Close-up of the user interface

"Fix the chain" asks the user to click in two regions and links them. It is the most used function. Note that it is possible to link regions that are in non-contiguous frames. This is specially useful for cases where a region becomes non-visible during some frames but later reappears.

When this function is used in regions *r1* and *r2*, the part of the chain that started at *r1* (let us call it "tail") is deleted (all those regions become non-matched), *r1* and *r2* are linked and *r2* is matched. Thus, the tail starting at *r1* is effectively recalculated.

"Chain ends at a region *r*" removes the tail from *r*. Useful for when a region becomes hidden completely in the rest of the animation. Only needs one click in the region r.

"Chain starts at a region *r*" launches the matching process at *r* as a new chain. Any tail starting at *r* or any link from another chain to *r* is deleted first. Only needs one click in region r.

4. Results and Discussion

We implemented our system in Java. We ran the tests on an Intel Core2 Quad CPU Q6700 @ 2.66 GHz, with 2 GB of RAM. The current implementation is single-threaded and does all its calculations on CPU. Segmenting and extracting the region features of one frame takes around 300 milliseconds. After the segmentation and the extraction of features, all the interaction is in real-time. The animation consists of five frames, each of 507x446 pixels. Each frame has respectively: 15, 16, 17, 18 and 18 regions. The outputs are in Figures 7, 8, 9, 10 and 11. We counted the number of mouse clicks on the panels (for indicating a region) and on the toolbox (choosing an option). The results are summarized in Table 1.

"Without chains" means that the weights were not recalculated according to the regions in the chains. They were set from the beginning to 1, meaning that each feature had the same weight. "With chains" means doing the comparison as in Section 3.4.

Table 1: Comparison of the number of clicks and wrongly colored regions using the different algorithms.

Algorithm	Number Of Clicks	Wrongly Colored Regions
Interactive without chains	38 clicks	0 regions
Interactive with chains	34 clicks	0 regions
Non-interactive without chains	0 clicks	43 regions
Non-interactive with chains	0 clicks	40 regions

The interactive algorithms had the user click in the right regions for fixing them, and thus achieved a correct matching. The Non-interactive ones required zero effort from the user, but had matching mistakes.

The final step until obtaining the completely colored animation would be to color the first frame and the first region of those chains that do not start at the first frame. Color will automatically propagate to all the matched regions. The five frames used as a test contain some difficult parts: the pony tail, the ear and the collar. That is because they change their shape, position and/or disappear.

As seen in Table 1, our non-interactive algorithm colored incorrectly 40 regions. Fixing that would require more than the 34 clicks of the interactive case. Both in the interactive and non-interactive cases the use of chains improved the overall accuracy.

It can happen that a correct matching without chains becomes incorrect when using them. This shows that chains are not perfect and also need fine-tuning for the animation in question. But their use, on the whole, seems beneficiary and is fast in its current implementation.

Our method shows how, for difficult cases, an approach that takes into account the previously matched frames and involves the user can be more efficient that a strict frame-by-frame non-interactive approach.

5. Conclusions and Future Work

As seen in the results, fixing the coloring mistakes as soon as they appear has allowed us to stop their propagation and the generation of other mistakes, while keeping the user intervention low.

Also, classifying the regions into chains and seeing how regions change along them has allowed us to increase the matching accuracy, thus reducing the total number of mistakes and reducing the total effort needed until having a correctly colored cartoon.

As for future work, we want to further improve the comparison function and take even more advantage of the chains to improve the accuracy.

One interesting case to include is to consider that there is occlusion in a region when the user links two regions in non-consecutive frames.

Another possible direction for improving the matching accuracy is to look for regions that break up into several ones or regions that merge into a single one, and treat them apart, since these are sources of matching mistakes.

References

[1] Catmull, E. The problems of computer-assisted animation. In Proceedings of the 5th Annual Conference on Computer Graphics and interactive Techniques, SIGGRAPH '78: 348-353, 1978.

[2] J. Qiu, H. Seah, F. Tian, Z. Wu, and Q. Chen. Feature- and region-based auto painting for 2D animation. Visual Computer, 21:928-944, 2005.

[3] Sýkora, D., Buriánek, J., and Žára, J. Unsupervised colorization of black-and-white cartoons. In Proceedings of the 3rd international Symposium on Non-Photorealistic Animation and Rendering, NPAR '04: 121-127, 2004.

[4] Qiu, J., Soon Seah, H., Tian, F., Chen, Q., and Wu, Z. Enhanced auto coloring with hierarchical region matching. Image, Colour and Illumination in Animation. Computer animation & virtual worlds. 16, 3-4: 463-473, 2005.

[5] Qiu, J., Seah, H. S., and Tian, F. Auto coloring with character registration. In Proceedings of the 2006 international Conference on Game Research and Development. ACM International Conference Proceeding Series, vol. 223: 25-32, 2006.

[6] Jie Qiu, Hock Soon Seah, Feng Tian, Quan Chen, Zhongke Wu, and Konstantin Melikhov, "Auto Coloring with Enhanced Character Registration," International Journal of Computer Games Technology, vol. 2008: Article ID: 135398, 2008.

[7] Fekete, J., Bizouarn, É., Cournarie, É., Galas, T., and Taillefer, F. TicTacToon: a paperless system for professional 2D animation. In Proceedings of the 22nd Annual Conference on Computer Graphics and interactive Techniques, SIGGRAPH '95: 79-90, 1995.

[8] J. Madeira, A. Stork, and M. Gross. An approach to computer-supported cartooning. Visual Computer, 12:1-17, 1996.

[9] C. Chang and S. Lee. Automatic cel painting in computer-assisted cartoon production using similarity recognition. The Journal of Visualization and Computer Animation, 8(3):165-185, 1997.

[10] H. Seah and F. Tian. Computer-assisted coloring by matching line drawings. Visual Computer, 16(5): 289-304, 2000.

[11] Jie Qiu; Hock Soon Seah; Feng Tian; Quan Chen; Melikhov, K., "Computer-assisted auto coloring by region matching," Computer Graphics and Applications. Proceedings. 11th Pacific Conference: 175-184, 2003.

Figure 7: The matching step by step: The frames before being matched.

Figure 8:The matching step by step: step1. The hair is matched.

Figure 9: The matching step by step: step 2. The face is matched.

Figure 10: The matching step by step: The frames after being completely matched.

Figure 11: The frames after being matched automatically without user guidance. Errors in the first frames are repeated in the coming frames and prevent correct matches, thus generating more mistakes.