Animation Inbetweening Based on Machine Learning and Spline Curves

Haina Wang[a], Houxian Su[a], Zhizhi Wang[a]

[a]*Zhejiang University, No.38 ZheDa Road, Hangzhou, 310027, Zhejiang, China*

Abstract

In simple terms, animation inbetweening is a kind of video interpolation for low-frame-rate 2D animation. Although there are many mature video interpolation algorithms, they are all designed for videos with higher frame rate than 24fps, so they are not quite suitable for animation inbetweening works with large motion spans and frame rates usually lower than 12fps. In addition, these interpolation algorithms do not do a good job of preserving the outlines of the objects in the video, which is also required in actual animation inbetweening work. In this work, we propose a different interpolation method. Specifically, we use B-spline curves to fit the outlines of 2 input frames, and find the curves that have moved between the frames and then match them. After that, we predict the positions of the moving curves in the middle frame, and thus we obtain the middle frame composed of the unmoved curves and the predicted curves. Our algorithm can preserve the outlines of the middle frame during interpolation, and any number of mid-frames is supported.

Keywords: inbetweening, interpolation, spline, animation

1. Introduction

Animation inbetweening is a common process in 2D animation production. This process usually need to add some intermediate frames to the keyframes (frame rate is usually less than 12fps) drawn in the previous process, so that the frame rate reaches 12fps to 24fps, making the actions in the animation look smoother. In the CV field, video interpolation is a research field close to this work, but the difficulty of animation interpolation is that the objects' movement between adjacent frames changes greatly, the video frame rate is also very low, and the interpolation requires that the outline shape of the intermediate frame be drawn clearer.

In recent years, with the great development of neural networks and deep learning, video interpolation algorithm research has made a lot of progress. Generally speaking, there are two main methods for existing video interpolation algorithms, one is kernel-based and the other is flow-based. Generally speaking, kernel-based interpolation is to process the input front and back frames with convolutional neural networks and then merge them[1] [2]; while flow-based methods are based on optical flow, the main idea is to estimate the optical flows from adjacents frames to the intermediate frame, and then use the estimated optical flow to warp the adjacent frames to obtain the required intermediate frame. A classic algorithm for this method is super slomo[3].

As the research on video interpolation progresses, some new design algorithm processes also become more complex. For example, Bao et al. proposed DAIN[4], which used a network that analyzed depth information[5] when estimating optical flow, and used image semantic analysis to obtain features[6] for assistance in generating intermediate frames using optical flow information; Siyao et al. proposed AnimeInterp[7], which used Segment Guided Matching based optical flow estimation to improve the quality of interpolation outline. There are kernel-based methods that use transformer[8], and there are also algorithms that try to combine flow-based and kernel-based methods[9].

The above research progress has made video interpolation quality higher and higher, however the work object is for the videos with a frame rate of at least 24fps, and the outline of the generated intermediate frame is often blurred. Even if AnimeInterp[7] based on Segment Guided Matching mentioned above improves the quality of interpolation outline, there is still some blur. Some video interpolation works extract outlines[10] separately for frames, but only use outline as a reference for interpolation[11], and their outlines still have blur.

Based on the researches above, our work mainly focuses on ensuring the clarity of intermediate frame outlines during interpolation. We propose a model based on spline curves. In our work, we first use B-spline curves to fit the contour lines of the adjacent frames, find out the curves that have moved and match them, and then predict the position of moving curves in intermediate frames. In this way, we get an intermediate frame composed of unmoved curves and predicted curves. Because we generate an intermediate frame described by B-spline, we ensure the clarity of our generated intermediate frame outline.

2. Related Work

2.1. ATD-12K Dataset

Due to the fact that many of the drawings of 2D animation are commercial activities, obtaining data sets is also a big difficulty for animation interpolation research. Fortunately, Siyao et al. proposed a data set ATD-12K in their AnimeInterp work[7].

ATD-12K is a large-scale animation triplet dataset, which comprises 12,000 triplets with rich annotations.

2.2. Conversion of Sketches

ATD-12K is taken from finished animations, so it's not feasible to treat it as datasets directly. Nevertheless, Edgar Simo-Serra et al.[12] has present a novel technique to simplify sketch drawings based on learning a series of convolution operators. Within the help of their work, ATD-12K can be transferred into hand-drawn like images with contours well-preserved. Furthermore, our output can also be optimized with it.

2.3. Bspline Fittings

To fit the contours of the work into Bsplines, we've refer to Wang et al.'s [13] work and Celnikera et al.'s [14] works. Specifically, we rewrite the cpp code provided by Wang with python after replacing some cpp library calls into our works.

3. Our Algorithm

3.1. Algorithm Overview

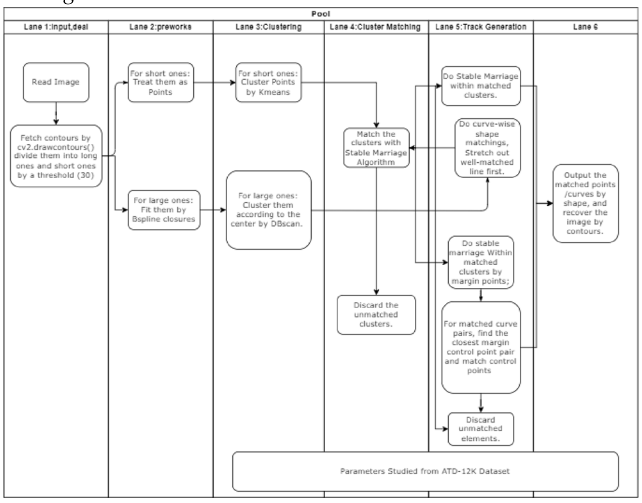
Pool
Lane 1:input,deal
Lane 2:preworks
Lane 3:Clustering
Lane 4:Cluster Matching
Lane 5:Track Generation
Lane 6
Read Image
Fetch contours by cv2.drawcontours() divide them into long ones and short ones by a threshold (30)
For short ones: Treat them as Points
For large ones: Fit them by Bspline closures
For short ones: Cluster Points by Kmeans
For large ones: Cluster them according to the center by DBscan.
Match the clusters with Stable Marriage Algorithm
Discard the unmatched clusters.
Do Stable Marriage within matched clusters.
Do curve-wise shape matchings, Stretch out well-matched line first.
Do stable marriage Within matched clusters by margin points;
For matched curve pairs, find the closest margin control point pair and match control points
Discard unmatched elements.
Output the matched points /curves by shape, and recover the image by contours.
Parameters Studied from ATD-12K Dataset

figure 1: Our algorithm's framework.

In our algorithm, we first use local threshold, stylization and global threshold in opencv to binarize the adjacent frames of the input into black and white. If it is for training, we will also do the same for the intermediate frame as ground truth. Then we use B-spline curves to fit the contour lines of the adjacent frames of interpolation, and then we find out the curves that have moved in adjacent frames. For the moved curves, we cluster them, and after clustering we match the moved curves from the front frame with the moved curves of the back frame. After that, we use the quadratic motion equation about time t which is trained to estimate the motion trajectory of the control points of the moved B-spline curves and the position of the control points in the intermediate frame. Finally, we restore the B-spline curve of the intermediate frame outlines from the predicted control points, which is the moved curves in our generated intermediate frame. Superimpose them with the unmoved curves of the adjacent frames to obtain an intermediate frame.

We leave many parameters within each step of the algorithm, which are to be trained by simple regressions on our dataset, therefore to get a machined-learning-assisted method.

3.2. Pre-processing and B-spline Fitting

We use local threshold, stylization and global threshold in opencv to binarize the adjacent front and back frames of the input into black and white. We use findcontours in opencv to extract all the outlines of the adjacent frames, denoted as $A\_contours$, $C\_contours$. Divide the outlines into those with $length > 30$ and those with $length < 30$, and get $A\_contours\_long$, $C\_contours\_long$, $A\_contours\_short$, $C\_contours\_short$. If this set of data is used for training, we will also read in the intermediate frame and perform the same operation.

Then we use B-spline curves to seperately fit the contour lines in $A\_contours$ and $C\_contours$. We limit the number of control points of B-spline to some n, where n is modifiable. This is designed to meet the fitting requirements for various images. If this tuple of data is used for training, we will also read in the intermediate frame and perform the same operation.

3.3. Movement Finding

For $A\_contours\_long$ and $C\_contours\_long$, we need to find out the set of curves that are really moving. We currently use 5 coefficients k_1 to k_5.

The fitting result is a set of closed third order B-spline curve with n control points third-order B-spline sub-curves obtained by fitting. For each sub-curve obtained by fitting, we take k_3 points at equal intervals. If k_1 points are consecutively not on the other frame, or a total of k_2 points are not on the other frame, this sub-curve needs to move; if the entire curve has k_4 consecutive points that need to move, or a total of k_5 segments need to move, the entire curve needs to move.

By simple trainings, the coefficients related to motion finding are set as follows:

$$k_1 = 8,\ k_2 = 17,\ k_3 = 25,\ k_4 = 9,\ k_5 = 10$$

3.4. Clustering

For the long curves $A\_contours\_long\_move$ and $C\_contours\_long\_move$ that need to move, we use biclustering; for the short curves $A\_contours\_short$ and $C\_contours\_short$, we use single clustering.

For biclustering, we firstly cluster the centroids of the curves using the algorithm DBscan, and limit the number of classes from $n = 5$ to $n = 12$ by tuning the parameters; then cluster all the points on the curves using the algorithm Agg. if there are not more than $2/3$ of the points on the same sub-curve of curve belong to the same class, it is regarded as the $n + 1$ class. Such kind of curves are usually born of global contours.

And for single clustering, we directly use K-means.

3.5. Matching

For the matching between A and C, we use diffrent algorithm for long curves and short curves.

To match the long curves, we first use structural method to try to match similar curves piecewisely. More specifically, we score the curve pairs in 3 dimensions:

1) shape. we choose a random point M first, and for n control points $P_{1..n}$, calculate n cross product $\vec{MP}_1 \times \vec{MP}_2, \vec{MP}_2 \times \vec{MP}_3, ..., \vec{MP}_n \times \vec{MP}_1$. Then we get another n-dimensional vector. Similar contours will have similar cross products, which can be judge by the first norm of the difference; Besides, the result may have many dimensions exactly the same and a few dimensions significantly differ(which is the most cases when a large character has a small moving part), so a few worst dimensions is required.

2) control point polygon's shape. We calculate the length of the sides of the control point polygon's to generate the vector, and use similar method to score it.

The weight and the detailing parameters are decided by regressions on the training data. We apply this to the large contours, stretch them out once a pair of curves among the two frames matches perfectly(decide by some parameter). And for those curves with no well matches, we leave them to the next step.

In the next step, we use the algorithm network flow to construct a surjection to the front frame (allowing a sub-curve of the back frame curve to match multiple front frame sub-curves), and the coefficient is the distance between the midpoints of the four sides of the rectangle enclosing the outline. For the matched curves, start from the control point corresponding to the nearest midpoint and match according to the direction (for example, if the lower boundary of the two curves is closest, then the lowest and leftmost points of the two curves match, and then match one by one counterclockwise).

To match the short curves, we directly treat them as separate points and use the Gale-shapley algorithm.

Mind that, in both the matchings of long and short curves with clusters, we use some functions to judge the quality of matchings and the risk. The matchings with little chance to be accurate (e.g. too far away or too prominent within its belonging cluster) will be discarded. The Gale-shapley algorithm can guarantee that there exist no pair of matchings which may get better after switches, so discarded part will not be too large. In practice, a poor matching may trigger notorious noise curves globally, which is much worse than do nothing at all.

3.6. Movement Prediction

If the point pair (x_1, y_1) and (x_2, y_2) are matched, it means there's a motion trajectory about time t pass the points $(x_1, y_1, 0)$ and $(x_2, y_2, 1)$.

We use the trained equations $x(t) = at^2 + (x_2 - x_1 - a)t + x_1$ and $y(t) = at^2 + (y_2 - y_1 - a)t + y_1$ to predict the $(x_{t_{1/2}}, y_{t_{1/2}})$ in $(x_{t_{1/2}}, y_{t_{1/2}}, t_{1/2})$.

3.7. Intermediate Frame Generating

We restore the B-spline curve of the intermediate frame outlines from the predicted control points, which is the moved curves in our generated intermediate frame. Superimpose them with the unmoved curves of the adjacent frames A and C to obtain an intermediate frame.

3.8. Pre Training

For most of the part, we just use simple simulated annealings on the dataset; And for some The main loss functions is the SSIM values; For example, the loss function of motion-detect part is the SSIM value between the mid-frame and the front/back frame after removing the moving parts.

4. Experimental Results

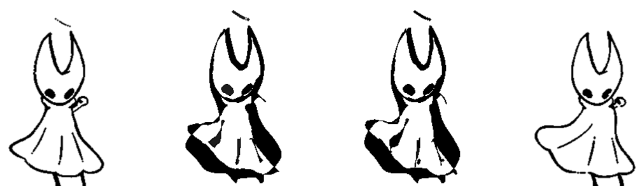

figure 2: A inbetweening sample of hornet(from game HOLLOW KNIGHT)

When generating the sample, the action-detecting part treat all the large contours to be moving. However, our current matching method match the long contours and roughly drawn their movements. However, the limitation of Bsplines and cv2.findcontours affect the result significantly, which will be discussed in the following section.

5. Discussions

Two limitations prevent our result from being better:

1) The method of finding contours. To divide the entire image into contours is difficult, and thinning algorithm costs much time, So we currently tried the cv2.findcontours(), leave the expecting to fill the colors by some later implementations afterwords. However, This results in the program's robustness be affected dramastically. A few pixels of noise may cause the contour result of the builtin function to be much different and later result in poor matching results; And during the moving of the contours, the rough contours may intersect or separate, which it's really bad after coloring given that most of the lines in the real cases are thin lines.

2) The B-splines. As the core part of our method, we've encountered these issues relevant with B-splines:

a) Fitting problems. It's obvious that B-splines can be self-intersected for some control points, and sadly this may happen during the fitting. To avoid this, we have to set *epoch* of fitting to a relative low value. However, inaccurate splines may result in intersections between 2 contours especially when they are born from the same thin line before, so here comes the paradox.

b) Side-affects of aligning the contours. The advantage of using the spline curve is that in addition to ensuring the integrity of the contour line, it also unifies the vectorized length of the curve for easy matching. However, it turns out that the contours still varies: A few long contours requires hundreds of control points to fit fairly, while most contours have length less then 100. Therefore, to simply let all the contours have the same number of control points may not be a wise idea. Some effective ways to get and match contours is required.

Acknowledgements

Thanks to Professor Gao'ang Wang and Senior Guan'hong Wang for their guidance, resources and company for the pasting year. The weekly seminars and meetings they held benefit a lot to us, inspire us from multiple aspects.

References

[1] LIU Z, YEH R A, TANG X, et al. Video frame synthesis using deep voxel flow[C] // Proceedings of the IEEE international conference on computer vision. 2017 : 4463 – 4471.

[2] LEE H, KIM T, CHUNG T-Y, et al. Adacof: Adaptive collaboration of flows for video frame interpolation[C] // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020 : 5316 – 5325.

[3] JIANG H, SUN D, JAMPANI V, et al. Super slomo: High quality estimation of multiple intermediate frames for video interpolation[C] // Proceedings of the IEEE conference on computer vision and pattern recognition. 2018 : 9000 – 9008.

[4] BAO W, LAI W-S, MA C, et al. Depth-aware video frame interpolation[C] // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019 : 3703 – 3712.

[5] CHEN W, FU Z, YANG D, et al. Single-image depth perception in the wild[J]. Advances in neural information processing systems, 2016, 29.

[6] NIKLAUS S, LIU F. Context-aware synthesis for video frame interpolation[C] // Proceedings of the IEEE conference on computer vision and pattern recognition. 2018 : 1701 – 1710.

[7] SIYAO L, ZHAO S, YU W, et al. Deep animation video interpolation in the wild[C] // Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021 : 6587 – 6595.

[8] SHI Z, XU X, LIU X, et al. Video frame interpolation transformer[C] // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022 : 17482 – 17491.

[9] BAO W, LAI W-S, ZHANG X, et al. Memc-net: Motion estimation and motion compensation driven neural network for video interpolation and enhancement[J]. IEEE transactions on pattern analysis and machine intelligence, 2019, 43(3) : 933 – 948.

[10] SIMO-SERRA E, IIZUKA S, ISHIKAWA H. Mastering sketching: adversarial augmentation for structured prediction[J]. ACM Transactions on Graphics (TOG), 2018, 37(1) : 1 – 13.

[11] LI X, ZHANG B, LIAO J, et al. Deep sketch-guided cartoon video inbetweening[J]. IEEE Transactions on Visualization and Computer Graphics, 2021, 28(8) : 2938 – 2952.

[12] SIMO-SERRA E, IIZUKA S, SASAKI K, et al. Learning to simplify: fully convolutional networks for rough sketch cleanup[J]. ACM Transactions on Graphics (TOG), 2016, 35(4) : 1 – 11.

[13] WANG W, POTTMANN H, LIU Y. Fitting B-spline curves to point clouds by curvature-based squared distance minimization[J]. ACM Transactions on Graphics (ToG), 2006, 25(2) : 214 – 238.

[14] CELNIKER G, WELCH W. Linear constraints for deformable non-uniform b-spline surfaces[C] // Proceedings of the 1992 Symposium on Interactive 3D graphics. 1992 : 165 – 170.