1 Introduction

In a practical mocap pipeline, it is frequently necessary for artists to manually adjust the results of the automatic retargeting. This is due to various reasons, including errors, late changes in artistic requirements, and the fact that subtle aspects of the original performance may be lost in the retargeting process. However, facial animation editing takes a lot of time. Once automatic retargeting is done, every frame in the shot becomes a keyframe, making temporally coherent editing challenging. In addition, the blendshape weights resulting from retargeting can be quite different from what artists usually use during their manual keyframing [Seol et al. 2011].

We present an intuitive animation editing method that helps artists to efficiently modify the retargeting results in a mocap pipeline. Based on Radial Basis Functions (RBF) network, our tune pose editing allows artists to intuitively control and revise the retargeting results to produce the desired final animation.

2 Tune pose editing

The facial mocap retargeting system consists of two main parts: solving and mapping. Similar to many previous approaches, the solving part computes the weight values \( w = \{w_i, i = 1...N\} \) of the actor’s FACS expressions at each frame as

\[
\text{min}_w \{||s - Bw||^2 + f(w)\}
\]

where \( s \) is the position of input face markers at a target frame, \( B \) is a FACS basis matrix, and \( f(w) \) is a function serving as a prior to shape the desired solution.

The tune pose editing becomes available in the mapping phase. In this phase, we map the resulting FACS weight \( w \) to the blendshape weights \( b = \{b_j, j = 1...M\} \) of the target character. Whereas previous approaches have used the simple setup of \( w = b \), we formulate the mapping as a scattered data interpolation based on RBF.

\[
b = \sum_{k=1}^{N} x_k \Phi(\|w_k - w\|) + g(w),
\]

where \( x_k \) is the weight of the basis function. The linear kernel \( \Phi(r) = r \) is used with an additional linear polynomial \( g(w) \), as this gives the desired results without any parameter tuning and has proven robust. Initial sample data consist of \( N \) sample pairs of \( w \) and \( b \) for the actor’s FACS expressions.

Applying this mapping to \( w \) at each frame computed in the solving stage gives the blendshape weights \( b \) of the character face. When artists want to edit any expression in the shot, they are allowed to simply go to the target frame and modify the facial expression as they want by changing \( b \) (adding a tune pose). Then our system automatically re-trains the mapping to include \( w \) and the modified \( b \) at the frame just edited. Applying the tuned mapping to \( w \) gives a refined animation on the character face.

Depending on the number or semantics of the FACS expressions, you may need to train multiple RBF mappings for separate facial regions. Dividing facial regions improves the effect of additional tune poses, and avoids the combinatorial explosion of required tune poses to make a global tune effect within a shot. In our cases, either three or five facial regions proved effective.

3 Discussion

One practical issue to consider is the trade-off between the exact recreation of artist’s modification and the stability of the RBF function. Artists sometimes add inconsistent tune poses to previously existing data, and this leads to a kind of overfitting. We control the trade-off using a regularization parameter inside the kernel function. A small amount of regularization \( \lambda = 0.05 \) made a good balance in our case. A possible extension would be developing a tool that warns artists when they introduce a certain amount of overfitting.

Although our tune pose editing provides powerful editing, artists may need manual keyframe edits to get the fine details of the final quality. We ask artists to move on to keyframe editing when tune pose gives diminishing returns, as abundant sample points by artists tend to degrade the quality of the mapping. In the manual keyframing stage, graph editing tools have proven to be very helpful. These include graph smoothing filters, spacetime editing, or salient points sampling [Seol et al. 2011].

References