APES: Articulated Part Extraction from Sprite Sheets –Supplementary Material–

Zhan Xu[1,2] Matthew Fisher[2] Yang Zhou[2] Deepali Aneja[2] Rushikesh Dudhat[1] Li Yi[3] Evangelos Kalogerakis[1]
[1]University of Massachusetts Amherst [2]Adobe Research [3]Tsinghua University

1. Applications

In our supplementary video https://youtu.be/YQtbRXFKNZE, we show two applications based on extracted articulated parts from APES: part-based animation and puppet creation via part swapping. We refer readers to the video for a demonstration of these applications.

For part-based animation, we create a set of control points or joints based on the extracted parts. We follow a simple heuristic to obtain them. First, we compute the centroid point of the whole character, and designate the part closest to the centroid as the "central" part (this coincides with the torso in Fig. 1). We create a joint at the center of this central part. The rest of the parts are designated as "limbs". We create joints ("pin joints") at the center of the intersection between the limbs and the central part (e.g., hips, shoulders in Fig. 1). Finally, we extract the medial axis of each limb, and create a joint at the medial axis point furthest to the pin joint for each limb (these points correspond to fingers and toes in Fig. 1). The resulting joints form a simple control structure (e.g., an animation skeleton) that can be loaded into standard animation software. We use the Adobe's Character Animator in the supplementary video. By manipulating the joint positions and angles, users can animate the characters based on the joints and parts extracted by APES.

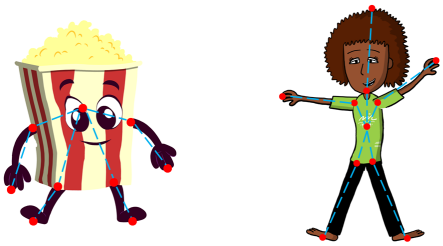

Figure 1. Control points (red dots) and skeletons created based on the extracted parts from APES.

2. Non-rigid Deformation for Reconstruction

As explained in Section 3.3, our part selection procedure uses a random rounding algorithm to solve the integer linear programming problem. The random rounding algorithm gives us multiple possible solutions. We pick the best solution in terms of reconstruction error after deforming each candidate set of parts to reconstruct the input poses. We describe here the deformation procedure.

Let $\mathbf{Q} = \{\mathbf{q}_c\}_{c=1}^{C}$ be a set of selected parts (where C is their total number), and the image $\mathbf{I}_t$ be the target pose to be reconstructed. A straight-forward method for reconstruction is fitting the optimal translation and rotation transformation for each part based on the predicted correspondences using the Procrustes orthogonal analysis used in ICP methods [1]. Such method is computationally efficient, but is sensitive to the noise in the predicted correspondences, and cannot capture non-rigid deformations that may exist in the input poses. Instead, we follow an as-rigid-as-possible deformation procedure (ARAP [4]). The ARAP takes as input a control mesh, and target positions for one or more of its vertices. To create a control mesh for each part $\mathbf{q}_c \in \mathbf{Q}$, we first uniformly sample a set of vertices $\mathcal{V}_c$ from its oriented bounding box. Then we create the Delaunay triangulation of the vertices. Each control mesh also incorporates the texture from the original appearance of its associated part. We treat the target positions $\{\mathcal{V}'_c\}_{c=1}^{C}$ of the mesh vertices as unknowns in an optimization problem that attempts to deform the control meshes of all parts such that their resulting appearance is as close as possible to the target pose $\mathbf{I}_t$, while at the same time the part deformations are as-rigid-as-possible. Specifically, we solve the following problem:

$$L(\mathcal{V}'_c) = \| \sum_{c=1}^{C} \Phi(\mathcal{V}'_c) - \mathbf{I}_t \|_2^2 + \lambda_r \sum_{c=1}^{C} \mathcal{R}(\mathcal{V}_c, \mathcal{V}'_c) \quad (1)$$

where λ_r is set to 0.05 in our experiments. The first term measures the reconstruction error between the deformed parts (caused by the shifted vertices) and the target pose. Φ renders each part with the shifted control vertices $\mathbf{V}'_c$ based on barycentric coordinates and bilinear interpolation [3],

which is differentiable. The second term is a regularization term that preserves the local shape during deformation. Specifically, it penalizes deviation from local rigid deformation for each control vertex neighborhood [4]:

$$\mathcal{R}(\mathcal{V}_c, \mathcal{V}'_c) = \sum_{\mathbf{v}_i \in \mathcal{V}_c} \sum_{\mathbf{v}_j \in \mathcal{N}(\mathbf{v}_i)} \|(\mathbf{v}'_j - \mathbf{v}'_i) - \mathbf{R}_i(\mathbf{v}_j - \mathbf{v}_i)\|^2 \tag{2}$$

where $\mathcal{N}(\mathbf{v}_i)$ is the neighborhood of the control vertex $\mathbf{v}_i$ and $\mathbf{R}_i$ is the best fit rotation of its neighborhood to the deformed configuration. The best fit rotations are computed via orthogonal Procrustes analysis. The optimization problem is solved iteratively. At each iteration, we alternative between solving for the deformed control vertices, and optimal rotations, as proposed in [4]. Fig.2 shows an example of our reconstruction approach.

(a) target pose (b) selected part (c) linear trans. (d) ARAP-based

Figure 2. Example of reconstruction via non-rigid deformation. By using the best-fit rigid transformation, the deformation (c) of the left arm cannot reconstruct well the target pose (a). Using the ARAP-based reconstruction (d), the arm is aligned better to the target appearance.

Non-rigid deformation for augmentation. As discussed in Section 5.1, we apply small, non-rigid deformations on each training body part to improve the pose diversity during training for the OkaySamurai dataset. To do so, for each part we uniformly sample control vertices in its oriented bounding box, and randomly shift the points by offsets sampled from Gaussian distribution $\mathcal{N}(0, \sigma^2)$ where σ is set as 2% of the maximum image dimension. We form the control meshes for parts, and deform them to reach the shifted control vertices using an ARAP deformation similar to Eq. 1.; here the reconstruction error measures difference between deformed and original control vertex positions.

3. Architecture Details

We provide here additional details of our network architecture.

Correspondence Module. Table 1 lists the layers used in the correspondence module along with the size of their output map. We call the architecture of our module as "Gated UNet" since the convolutional layers in the encoder implement Gated Convolution [6].

Table 1. **Correspondence module architecture (Gated UNet).** *Conv3x3* is convolutional layer with kernel size 3. *LN* is LeakyReLU with negative slope as 0.2. *BN* is BatchNorm. *Dot* between LR and Sigmoid is element-wise product. *Upsample* uses bilinear interpolation for upsampling. We note that there are skip connections between corresponding layers from encoder and decoder, following the original U-Net architecture.

	Layers	Output
Input	Concat(image, mask)	256×256×4
Encoder	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	256×256×32
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	256×256×32
	MaxPooling(2)	128×128×32
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	128×128×64
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	128×128×64
	MaxPooling(2)	64×64×64
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	64×64×128
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	64×64×128
	MaxPooling(2)	32×32×128
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	32×32×256
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	32×32×256
	MaxPooling(2)	16×16×256
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	16×16×256
	BN(LR(Conv3x3)·Sigmoid(Conv3x3))	16×16×256
Decoder	Upsample(2)	32×32×256
	BN(ReLU(Conv3x3))	32×32×128
	BN(ReLU(Conv3x3))	32×32×128
	Upsample(2)	64×64×128
	BN(ReLU(Conv3x3))	64×64×64
	BN(ReLU(Conv3x3))	64×64×64
	Upsample(2)	128×128×64
	BN(ReLU(Conv3x3))	128×128×32
	BN(ReLU(Conv3x3))	128×128×32
	Upsample(2)	256×256×32
	BN(ReLU(Conv3x3))	256×256×32
	BN(ReLU(Conv3x3))	256×256×32
	BN(ReLU(Conv3x3))	256×256×64

Table 2. **Clustering module architecture.** The symbol "*pred.R*" means predicted rotations, and "*pred.T*" means predicted translations. To update the voting map, we apply the predicted rotations to source pixels.

	Layers	Output
Input	Concat(voting map, mask)	256×256×5
Gated UNet	similar to Table 1 with intermediate channel numbers as 16, 32, 64, 128, 256	256×256×16
Average pooling per superpixel	N/A	K_s×16
MLP	16→64→2	pred.R: K_s×2
Updated input	Concat(updated voting map, mask)	256×256×5
Gated UNet	similar to Table 1 with intermediate channel numbers as 16, 32, 64, 128, 256	256×256×16
Average pooling per superpixel	N/A	K_s×16
MLP	16→64→2	pred.T: K_s×2

Clustering Module. Table 2 shows the architecture of the clustering module.

4. Implementation Details

The correspondence module is first trained alone. We set the learning rate to 10^{-3} and decrease it to 10^{-4} after 5 epochs. Then we train both the correspondence and clustering modules using the soft nearest neighbor in Equation 5 of the main text. We set the learning rate to 10^{-6} for the correspondence module and 10^{-4} for the segmentation module with a batch size of 8 for this stage. We use the Adam optimizer. The height H and width W of the images and voting maps are always 256. The number of clusters C_s during training is set to 12.

5. Correspondence Comparison

Fig. 3 shows a qualitative comparison example of predicted correspondences between our method, RAFT [5] and COTR [2]. To help RAFT and COTR better use the foreground masks, we map the predicted target positions to their nearest neighbors in the foreground. RAFT and COTR cannot produce correspondences reliably e.g., for hand and head regions, as shown below.

Figure 3. Visualization of the predicted correspondences from our method, RAFT and COTR. From left to right: (a) source image (b) target image (c) color-coded source pixels (d) ground-truth target pixels (e) prediction of our method (f) prediction of RAFT (g) prediction of COTR. Corresponding pixels have same color in the visualization of correspondence maps. Our method matches pixels more accurately compared to other methods.

6. More Qualitative Results

We show here additional qualitative results from the The CreativeFlow+ dataset. We note that this dataset does not include segmentations of articulated parts. Their provided segmentation maps are based on mesh components, which often do not match articulation. Thus, as an additional test set, we manually segmented 12 characters into rigid parts from their test split, and used them as reference to measure the performance of our trained model on them. We achieve an IoU of 67%, which is slightly lower than the IoU we achieved for the OkaySamurai dataset (71%). Two examples are shown in Fig. 4.

We include the results for the test sprite sheets in OkaySamurai dataset and SPRITE in our code repository.

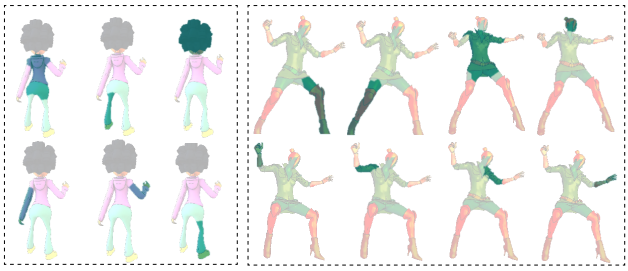

Figure 4. Part extraction results in the CreativeFlow+ dataset.

References

[1] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE TPAMI*, 14(2), 1992. 1

[2] Wei Jiang, Eduard Trulls, Jan Hosang, Andrea Tagliasacchi, and Kwang Moo Yi. Cotr: Correspondence transformer for matching across images. In *ICCV*, 2021. 3

[3] Erika Lu, Forrester Cole, Tali Dekel, Weidi Xie, Andrew Zisserman, David Salesin, William T Freeman, and Michael Rubinstein. Layered neural rendering for retiming people in video. In *SIGGRAPH Asia*, 2020. 1

[4] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *SGP*, volume 4, 2007. 1, 2

[5] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, 2020. 3

[6] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *ICCV*, 2019. 2