EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

Supplementary Material for Interactive Motion Mapping for Real-time Character Control

Helge Rhodin[1], James Tompkin[1,2], Kwang In Kim[1,3], Kiran Varanasi[1,4], Hans-Peter Seidel[1], Christian Theobalt[1]

[1]Max-Planck-Institute for Informatics, [2]Intel Visual Computing Institute, [3]Lancaster University, [4]Technicolor Research

1. Shape Space Transformation

In this section, we explain the utilized shape space (main paper Section 4) and its reconstruction step (main paper Section 6) in more detail. The main paper contains a condensed explanation as we only apply well known techniques from soft body deformation methods [SCOL04] to extend established deformation gradient based methods [SP04]. Our shape space represents the shape of connected components by per-face transformations, which are more suitable to model rotations than absolute vertex positions. In addition, we model the global position of each connected component by vertex positions as this information is not contained in the face transformations.

1.1. Mesh Representation

For each face f of the input mesh we compute the affine transformation A_f in relation to the rest pose $\mathbf{x}_0$. Rotation R_f and shear S_f are extracted by polar decomposition on A_f. We compute the polar decomposition iteratively according to Higham [Hig86]:

$$\begin{aligned}
&\text{Initialize: } U_0 = A\\
&\text{Repeat: } U_{k+1} = \frac{U_k + U_k^{-T}}{2}\\
&R_f = U_{k+1}\\
&\text{If } det(R_f) < 0 \text{ then } R_f = -R_f \qquad \text{// prevent reflection}\\
&S_f = \frac{R_f^T A_f + A_f^T R_f}{2} \qquad (1)
\end{aligned}$$

In our experiments three iterations sufficed. Rotations are processed in axis-angle form the symmetric shear matrix is linearized to a vector of 6 elements.

In addition to rotation and shear, we also store the original vertex positions v_i for each vertex i. Therefore, each pose vector is a concatenation of $3F$ rotation, $6F$ shear, and $3V$ point parameters for a character with F faces and V vertices.

1.2. Reconstruction

The combined shape space is overcomplete. The face transformations can be completely characterized by the explicit vertex positions. As we do not enforce constraints between both representations during mapping, then they can contradict. A globally-consistent shape is reconstructed by considering A_f and v_i as soft constraints with weight h on $\mathbf{v}$. We choose h by hand such that the contributions of face rotations are approximately one order of magnitude larger than the point features. Thereby, the face transformations dominate the shape reconstruction while the vertex positions determine the global position of connected components.

An efficient solution is possible by using a Laplacian coordinate representation [SCOL04]. The Laplacian L is constructed once from the reference pose vertex positions $\mathbf{v}_1$ and the mesh face edges with cotangent weights. The Laplacian gives the differential coordinates δ by:

$$L\mathbf{v} = \delta \qquad (2)$$

Given a shape space pose representation $\mathbf{y}^{\text{shape}} := (\mathbf{A}, \mathbf{v})$, where $\mathbf{A} = (A_1, \cdots, A_F)$ and $\mathbf{v} = (v_1, .., v_V)$, we optimize for the mesh vertex positions $\mathbf{v}^*$ that simultaneously fit the reference pose updated by rotations $\mathbf{A}$ and vertex positions $\mathbf{v}$ in the least squares sense, as in soft body deformation methods:

$$\mathbf{v}^* = \arg\min_{\upsilon} \|\mathbf{v} - \upsilon\|^2 + h\|L\mathbf{A}(\mathbf{v}_1) - L\upsilon\|^2, \qquad (3)$$

where $\mathbf{A}(\mathbf{v}_1)$ is the reference mesh updated by per face transformations $\mathbf{A}$. Since Equation 3 is quadratic in υ and L is sparse, $\mathbf{v}^*$ can be inferred efficiently by solving a sparse linear system of equations.

Explicit vertex constraints can easily be added by setting selected v_i to the goal location and increasing their weight similar to [SCOL04]. This strategy could provide a good way to prevent ground penetration and foot skating.

2. Latent Volume—a Pose Prior

We represent the plausible ranges of variations of character poses $\mathbf{y}^{\text{pose}}$ by their *latent volume* (see main paper Section 5.1). Our approach is to exploit unlabeled data for inferring the admissible variations. This corresponds to finding the support of probability distribution $P_{\mathbf{y}^{\text{pose}}}$.

In general, finding the support of a probability distribution is a non-trivial problem. However, by assuming that $P_{\mathbf{y}^{\text{pose}}}$ is approximated by a Gaussian distribution, we find an efficient estimate of its support by first decorrelating the data and then finding the support of the resulting empirical distribution within each data dimension. That is, the latent volume defines the support of $P_{\mathbf{y}^{\text{pose}}}$ and is specified as the minimum interval I_c encompassing all decorrelated data points for each dimension c. In the context of the regression problem, the latent volume may be interpreted as an approximate prior distribution $P_{\mathbf{y}^{\text{pose}}}$ on the target character pose.

3. Pseudocode

To aid reproductions and extensions of the presented method, we provide pseudocode for the core methods. Where suitable, we use the notation introduce in the main paper. These listings begin overleaf.

The confidence score is computed by the method **computeConfidence**. It requires the pose $\mathbf{x}_t^{\text{pose}}$ at which the score should be computed and the list of already selected source labels $\mathcal{L}_x$. The output is the scalar confidence value.

function COMPUTECONFIDENCE($\mathbf{x}_t^{\text{pose}}, \mathcal{L}_x$)
  /* compute prediction variance */
  $X = (\mathcal{L}_x[0], \mathcal{L}_x[1], \ldots)$// data matrix
  $A = \sigma^{-2} X X^\top + I$
  return $\mathbf{x}_t^{\text{pose}\top} A^{-1} \mathbf{x}_t^{\text{pose}}$
end function

The performance based key frame selection is implemented as method **performanceCorrespond**. It assumes a list of target labels $\mathcal{L}_y$. It outputs corresponding source labels $\mathcal{L}_x$. The user interacts with the system through the *InputDevice* and *RemoteControl* and gets feedback trough the display of the confidence bar and the current pose.

function PERFORMANCECORRESPOND($\mathcal{L}_y$)
  while size($\mathcal{L}_x$) < size($\mathcal{L}_y$) **do**
    $\mathbf{x}_t^{\text{pose}}$ = InputDevice.getCurrentPose()
    /* display current pose */
    display($\mathbf{x}_t^{\text{pose}}$)
    /* display confidence score bar */
    display(computeConfidence($\mathbf{x}_t^{\text{pose}}$, $\mathcal{L}_x$))
    if RemoteControl.buttonPressed() **then**
      $\mathcal{L}_x$.add($\mathbf{x}_t^{\text{pose}}$)
    end if
  end while
  return $\mathcal{L}_x$
end function

The automatic key frame proposal is defined in method **autoCorrespond**. It requires the source reference motion $\mathcal{U}_x := (\mathbf{x}_1^{\text{pose}}, \cdots, \mathbf{x}_M^{\text{pose}})$ and the target labels $\mathcal{L}_y$ as input. It returns the correspondences selected by the user out of a small pool of automatically proposed candidates.

function AUTOCORRESPOND($\mathcal{U}_x, \mathcal{L}_y$)
  for all $\mathbf{y}^{\text{pose}} \in \mathcal{L}_y$ **do**
    display($\mathbf{y}^{\text{pose}}$)
    /* compute confidence score */
    for all $\mathbf{x}^{\text{pose}} \in \mathcal{U}_x$ **do**
      conf[$\mathbf{x}^{\text{pose}}$] = computeConfidence($\mathbf{x}^{\text{pose}}, \mathcal{L}_x$)
    end for
    for all $\mathbf{x}^{\text{pose}} \in \mathcal{U}_x$ **do**
      if conf.isLocalMaxima(conf[$\mathbf{x}^{\text{pose}}$]) **then**
        proposals.add($\mathbf{x}^{\text{pose}}$)
      end if
    end for
    /* Let user choose from top k proposals */
    $\mathcal{L}_x$.add(userSelection(proposals))
  end for
  return $\mathcal{L}_x$
end function

The target motion $\mathcal{U}_y$ is preprocessed offline by method **learnOffline**. As input it requires the mesh edges *meshEdges* and the $\mathcal{U}_y$ matrix, where each column contains the vertex positions of one frame and the global world translations $\mathcal{U}_y^{\text{trans}}$. The function computes a PCA basis V_y, mean $\bar{\mathbf{y}}^{\text{shape}}$, latent volume intervals $I_{\text{min}}, I_{\text{max}}$, and the global translation regression matrix W_T.

function LEARNOFFLINE($\mathcal{U}_y$, $\mathcal{U}_y^{\text{trans}}$, meshEdges)
/* Shape space computation*/
shapeSpace.init($\mathcal{U}_y$[0], meshEdges)
$\mathcal{U}_y^{\text{shape}}$ = shapeSpace.project($\mathcal{U}_y$)
/* Compute D-dimensional target PCA basis V_y*/
$\bar{\mathbf{y}}^{\text{shape}}$ = mean($\mathcal{U}_y^{\text{shape}}$)
V_y = PCA($\mathcal{U}_y^{\text{shape}} - \bar{\mathbf{y}}^{\text{shape}}$) //element wise
$\tilde{\mathcal{U}}_y^{\text{shape}} = V_y^\top \cdot \mathcal{U}_y^{\text{shape}}$
/* derive bounds $I_{\text{min}}, I_{\text{max}}$ for each PCA coefficient*/
I_{min} = min($\tilde{\mathcal{U}}_y^{\text{shape}}$) //element wise
I_{max} = max($\tilde{\mathcal{U}}_y^{\text{shape}}$) //element wise
/* compute global translation map*/
$\mathrm{X} = \text{derivatives}(\tilde{\mathcal{U}}_y^{\text{shape}})$ // finite differences approximation
$\mathrm{Y} = \mathcal{U}_y^{\text{trans}}$
$A = \sigma_T^{-2} X X^\top + I$
$W_T = \sigma_T^{-2} Y X^\top A^{-\top}$
return $V_y, I_{\text{min}}, I_{\text{max}}, M_T$
end function

After correspondence definition **learnOnline** prepares the source character and learns the mapping from source to target. As input it assumes source and target labels $\mathcal{L}_x$, $\mathcal{L}_y$, and source reference motion $\mathcal{U}_x$ in matrix form as input and accesses $\bar{\mathbf{y}}^{\text{shape}}$ and V_y from the preceding offline training. It outputs the linear regression matrix W.

function LEARNONLINE($\mathcal{L}_x$, $\mathcal{L}_y$, $\mathcal{U}_x$)
/* Build data matrices from key frames*/
$\bar{\mathbf{x}}^{\text{pose}}$ = mean($\mathcal{U}_x$)
$\mathrm{X} = (\mathcal{L}_x - \bar{\mathbf{x}}^{\text{pose}})$
$\mathrm{Y} = V_y^\top \cdot (\text{shapeSpace.project}(\mathcal{L}_y) - \bar{\mathbf{y}}^{\text{shape}})$
/* Learn mapping*/
$A = \sigma^{-2} X X^\top + I$
$W = \sigma^{-2} Y X^\top A^{-\top}$
return W
end function

Real-time synthesis is obtained by **predictTarget**. It takes the current live input pose χ_t^{pose} and computes the corresponding target pose γ_t^{pose} based on the previously computed PCA basis and W and target translation γ_t^{trans} from W_T.

function PREDICTTARGET(χ^{pose})
/* Mean center source pose */
$\tilde{\chi_t}^{\text{pose}} = V_x^\top \cdot (\chi_t^{\text{pose}} - \bar{\mathbf{x}}^{\text{pose}})$
/* apply linear map and bound by latent volume*/
$\tilde{\gamma_t}^{\text{shape}} = W \cdot \tilde{\chi}_t^{\text{pose}}$
$\tilde{\gamma_t}^{\text{shape}} = \max(I_{\text{min}}, \tilde{\gamma}_t^{\text{shape}})$ //element wise
$\tilde{\gamma_t}^{\text{shape}} = \min(I_{\text{max}}, \tilde{\gamma}_t^{\text{shape}})$ //element wise
/* reconstruct target pose*/
$\gamma_t^{\text{shape}} = V_y \cdot \tilde{\gamma}_t^{\text{shape}} + \bar{\mathbf{y}}^{\text{shape}}$
γ_t^{pose} = shapeSpace.reconst(γ_t^{shape})
/* infer global translation*/
$\gamma_t^{\text{trans}} = W_T \cdot |\tilde{\gamma}_t^{\text{shape}}|$ //element wise absolute value
return $\gamma_t^{\text{pose}}, \gamma_t^{\text{trans}}$
end function

The shape space is divided into methods **ShapeSpace::init**, **ShapeSpace::project** and **ShapeSpace::reconstruct**. **ShapeSpace::init** requires the reference pose $\mathbf{y}_1^{\text{pose}}$ and the mesh connectivity *meshEdges* as input. It builds the augmented laplacian matrix L' and computes its Cholesky decomposition LC.

function SHAPESPACE::INIT($\mathbf{y}_1^{\text{pose}}$, meshEdges)
  /* compute laplacian matrix*/
  L = buildLaplacianMatrix($\mathbf{y}_1^{\text{pose}}$, meshEdges)
  /* append identity matrix for vertex constraints */
  L' = $\binom{L}{I}$
  /* prepare solver */
  LC = sparseCholeskyDecomposition(L')
end function

For a given target pose $\mathbf{y}_t^{\text{pose}}$ method **ShapeSpace::project** extracts face rotations R_f and shear S_f for each face f. It returns the feature vector $\mathbf{y}_t^{\text{shape}} \in 3V + 9F$ that contains rotation, shear, and vertex features.

function SHAPESPACE::PROJECT($\mathbf{y}_t^{\text{pose}}$)
  $\mathbf{y}_t^{\text{shape}} = \text{vec}(\mathbf{y}_t^{\text{pose}})$ // initialize feature vector with linearized vertex positions
  for all f $\in \mathbf{y}_1^{\text{pose}}$ **do** // loop over all faces
    /* compute affine transform */
    A_f = affineTransform($\mathbf{y}_t^{\text{pose}}$.face(f), $\mathbf{y}_0^{\text{pose}}$.face(f))
    /* polar decomposition */
    [R,S] = polarDecomposition(A)
    /* append roations and shear to feature vector */
    $\mathbf{y}_t^{\text{shape}} = \begin{pmatrix} \mathbf{y}_t^{\text{shape}} \\ axisAngle(R) \\ \text{vec}(S) \end{pmatrix}$
  end for
  return $\mathbf{y}_t^{\text{shape}}$
end function

ShapeSpace::project inverts the projection step. It takes the shape representation γ_t^{shape} as input and solves for the mesh γ_t^{pose} that best match the face and vertex constraints by solving a linear system efficiently with the offline computed Cholesky decomposition LC.

function SHAPESPACE::RECONSTRUCT(γ_t^{shape})
  /* extract position, rotation, and shear form vector */
  $[v_1, \cdots, v_N, R_1, \cdots R_F, S_1, \cdots S_F]$ = extract(γ_t^{shape})
  /* differential of transformed faces */
  diff = $L\,R(S(\mathbf{y}_1^{\text{pose}}))$
  /* integrate vertex constraints */
  diff' = $\begin{pmatrix} \text{diff} \\ v_1 \\ \vdots \\ v_N \end{pmatrix} \in 2V \times 3$
  /* solve linear system */
  $\mathbf{v}_t$ = solve($L'\mathbf{v}_t$ =diff') using LC
  $\gamma_t^{\text{pose}} = \text{vec}(\mathbf{v}_t)$
  return γ_t^{pose}
end function

References

[Hig86] HIGHAM N.: Computing the polar decomposition-with applications. *SIAM Journal on Scientific and Statistical Computing* (1986). 1

[SCOL04] SORKINE O., COHEN-OR D., LIPMAN Y.: Laplacian surface editing. *Proc. SGP* (2004), 175–184. 1

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM TOG (Proc. SIGGRAPH) 23*, 3 (2004), 399–405. 1