Mesh-Based Inverse Kinematics

Robert W. Sumner          Matthias Zwicker          Craig Gotsman          Jovan Popović

CSE291 Presentation by Chih Liang

What is MeshIK?
Given a number of example meshes (or poses), MeshIK computes a new mesh with the fixed vertices as contraints.

Feature Vector
- Represent each example using feature vector
- Feature vector consists of deformation gradients
- Deformation gradients describe deformation of each triangle
- Transformation of a triangle relative to a reference triangle

Deformation Gradient
Affine Transformation for j-th triangle:
\[ \Phi_j(p) = T_j p + t_j \]
Deformation Gradient is Jacobi matrix:
\[ D_{pT_j(p)} = T_j \]
3x3 matrix \( T_j \) contains rotation, scaling, and skewing components

Solving for Transformation
\( T_j \) is not unique for a triangle, a fourth vertex is added:
\[ v_4 = v_1 \pm \frac{(v_2 - v_1) \times (v_3 - v_1)}{\sqrt{(v_2 - v_1) \times (v_2 - v_1)}} \]
From the affine transformation of all 4 vertices,
\[ \begin{align*}
    v_1 - T_1 t_j &= v_1 - T_2 t_j \\
    v_1 - T_3 t_j &= v_1 - T_4 t_j
\end{align*} \]
Subtract each from last equation & solve for \( T_j \)
\[ T_j = (v_1 - v_4, v_2 - v_4, v_3 - v_4) [v_1 - v_4, v_2 - v_4, v_3 - v_4]^{-1} \]
where bar means reference

Feature vector of a deformed mesh
\[ f = G x \]
\[ x = (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) \]
That means, \( G = 9m \times 3n \) sparse matrix

\[ M = \# \text{ of triangles} \]
\[ N = \# \text{ of vertices} \]
Linear operator G example

\[ \begin{bmatrix}
  a_{10} & a_{11} & a_{12} \\
  a_{20} & a_{21} & a_{22} \\
  a_{30} & a_{31} & a_{32}
\end{bmatrix}
\]

For \( j \)-th triangle, let

\[ \begin{bmatrix}
  v_x \ v_y \ v_z \ v_x \ v_y \ v_z \\
  v_x \ v_y \ v_z \ v_x \ v_y \ v_z \\
  v_x \ v_y \ v_z \ v_x \ v_y \ v_z
\end{bmatrix}
\]

Finding a mesh given a feature vector

\[ \mathbf{x} = \arg \min_{\mathbf{x}} \| \mathbf{Gx} - (\mathbf{f} + \mathbf{c}) \| \]

- \( \mathbf{G} \) is void of columns that multiply the fixed vertices
- \( \mathbf{c} \) is the result of this multiplication
- \( \mathbf{x} \) is void of the fixed vertices

Feature space

- Combine feature vectors of examples to generate new shapes
- \( \mathbf{w} = \) weights
- \( \mathbf{f}_1, \ldots, \mathbf{f}_n = \) feature vectors of examples

Linear vs Nonlinear feature space

Linear feature space

\[ \mathbf{M}(\mathbf{w}, \mathbf{f}_1, \ldots, \mathbf{f}_n) = \mathbf{Mw} \]

doesn’t handle rotations correctly

Quaternions, a nonlinear interpolation, can handle only 2 rotations. 

Nonlinear Feature Space

- Defines the space of desirable deformations.
- Polar decomposition and matrix exponential map.
**Polar Decomposition**

Decompose the deformation gradient $T_{ij}$ for the $j$-th triangle in the $i$-th pose:

$$T_{ij} = R_{ij} S_{ij}$$

- $R$ is the rotation component
- $S$ is the scale/shear component

**Exponential Map**

Nonlinear blend for deformation gradient of the $j$-th triangle for all example poses:

$$T_j = \exp\left\{ \sum_{i=1}^l w_i \log(R_{ij}) \right\} \cdot \sum_{i=1}^l w_i S_{ij}$$

- Rotations: matrix exponential and log functions
- Scale/Shear: linear combinations

**Nonlinear Least-Squares**

$$x^*, w^* = \arg\min_{x, w} \|Gx - (M(w) + c)\|$$

Solve simultaneously for $x$ and $w$

- $c$ = user constraints
- $M(w)$ = feature vector from nonlinear blend
- $Gx$ = feature vector of deformed mesh

**Gauss-Newton Example**

$$x^* = \arg\min_x \|x^2\|$$

$$x_1 = \arg\min_\delta \| (x_1 + \delta)^2 \|$$

$$x_1 = \arg\min_\delta \| x_1 + 2x_1 \delta \|$$

$$\implies x_1 = \frac{x_1}{2}$$

**Gauss-Newton Algorithm**

$$x^*, w^* = \arg\min_{x, w} \|Gx - (M(w) + c)\|$$

- Linearize nonlinear function $w$/
- Result

$$x_{k+1} = x_k - \arg\min_{x} \|Gx - (M(w_k + \delta x_k) + c)\|$$

$$= \arg\min_{x} \|Gx - D_x M(w_k + \delta x_k) - (M(w_k) + c)\|$$

$m$ is the number of triangles.
Each triangle has $T$, which has 9 entries specified by row and column indices.
Gauss-Newton Iteration

\[ \delta_k x_{k+1} = \arg \min_{\delta x} \| Gx - D_k M(w_k) \delta - (M(w_k) + c) \| \]

\[ \Rightarrow A^T A \begin{bmatrix} x \\ \delta \end{bmatrix} = A^T (M(w_k) + c) \]

- Each iteration, update \[ w_{k+1} = w_k + \delta_k \]
- Until \[ \| \delta_k \| < \sqrt{\epsilon}/\| w_k \| \]

Cholesky Factorization

- Make \[ A^T A = U^T U \]
- \[ U = \text{upper-triangular matrix} \]
- Cholesky way to solve for \[ U^T U X = B \]
  1. Solve for \[ Y \] in \[ U^T Y = B \]
  2. Solve for \[ X \] in \[ UX = Y \]
- \[ \text{[triangular matrix]}^{-1} = \text{[triangular matrix]} \]
  (can compute easily)

Pros & Cons of MeshIK

- A small of work from user to displace vertices
- Mesh deformations are meaningful
- Can specify weights independently to blend the poses
- Must provide example meshes
- Slow due to convergence
- Requires same connectivity structure for all meshes

Demo Video