Animation & Simulation

He Wang (王鹤)

Interpolation-Based Animation

- Deforming an embedding space
  - Polyline deformation
  - Global deformation
  - FFD
  - Composite FFD
  - Mean Value Coordinates
  - Harmonic Coordinates
  - Green Coordinates
  - Animating FFD

Interpolation-Based Animation

- 3D shape interpolation
  - Surface-based
    - Vertex-matching
    - Vertex-based interpolation
    - Difficult to handle holes (topology)
  - Volume-based
    - Blend volumes
    - Less sensitive to holes but more expensive since requiring volume representation

Interpolation-Based Animation

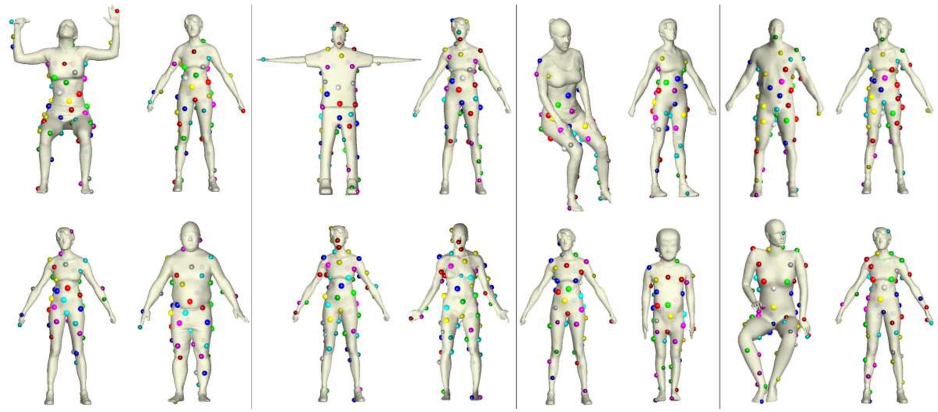
- 3D shape interpolation
    - Key concept: Topology
        - Connectivity of a surface
            - Simply, the number holes, (not in general)
            - A doughnut = a teacup, a beach ball = a blanket
            - Two objects are ***homeomorphic**,* if there is a *continuous*, *one-to-one*, *invertible* mapping between points on the surfaces of two objects
            - *Genus* refers to how many holes (a beach ball has 0, a teacup has 1)
        - In Computer graphics, vertex/edge/face connectivity of a polyhedron
            - Invariant to the position and orientation of the object

Interpolation-Based Animation

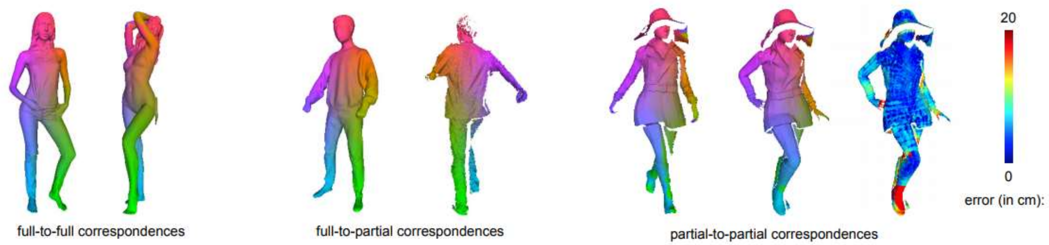
- 3D shape interpolation
  - Correspondence problem
  - Interpolation problem

Interpolation-Based Animation

- 3D shape interpolation
  - Matching topology
    - The same topology (vertex-edge), interpolate individual vertices



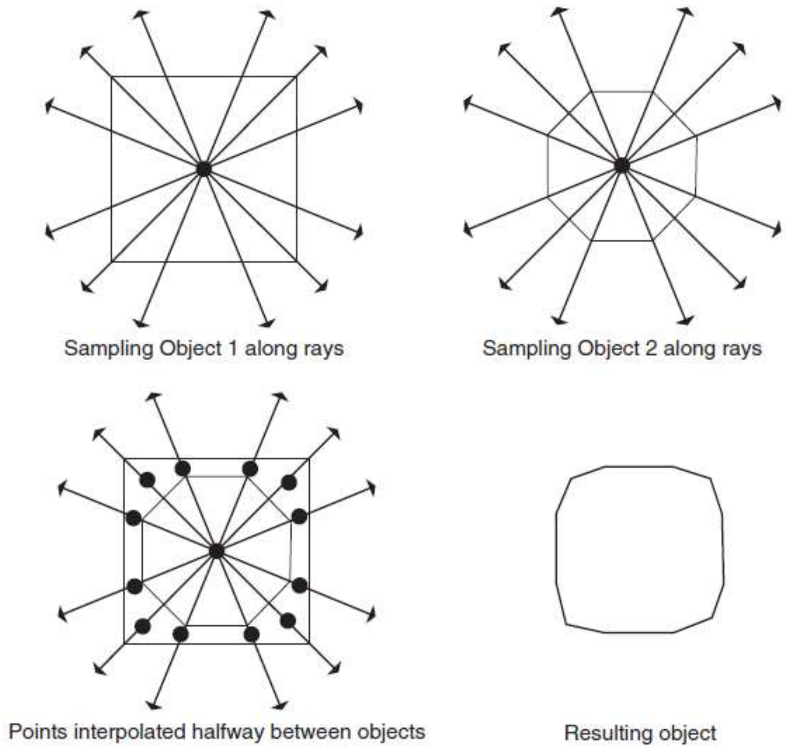
Lipman et al. Mobius Voting For Surface Correspondence, 2016



Wei et al. Dense Human Body Correspondences Using Convolutional Networks, 2016

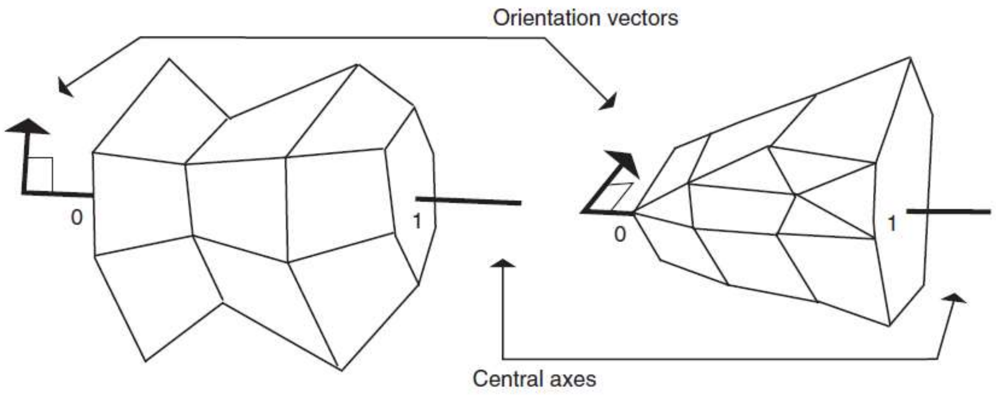
Interpolation-Based Animation

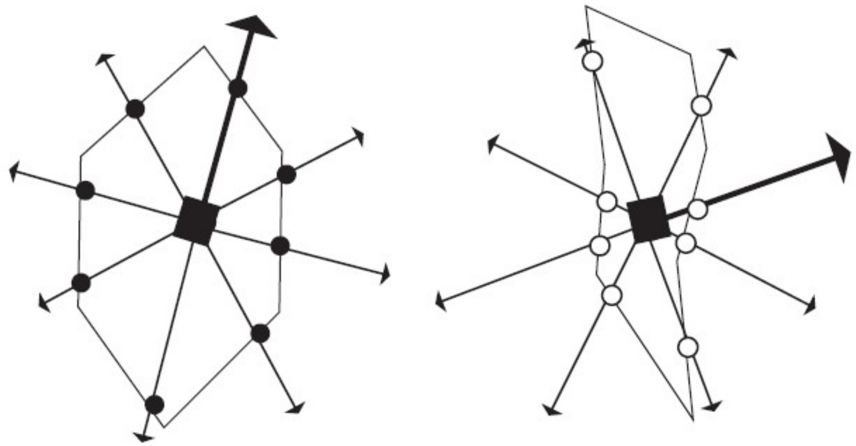
- 3D shape interpolation
  - Matching topology
  - Star-shaped polyhedral
    - Defining the same 'coordinate system' on objects
    - Points on the same axis have correspondence
    - Interpolate on the axes



Interpolation-Based Animation

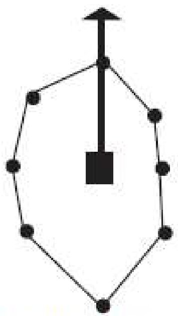
- 3D shape interpolation
  - Matching topology
  - Star-shaped polyhedral
  - Axial slices



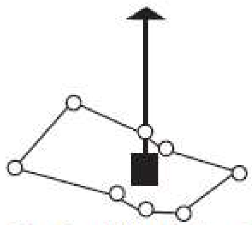


Interpolation-Based Animation

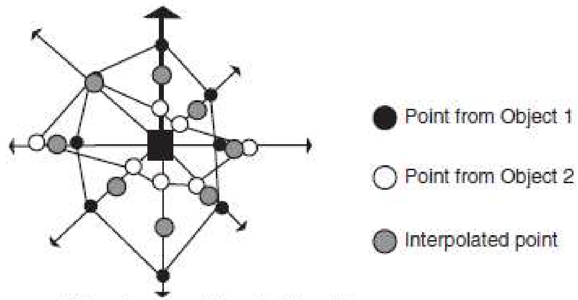
- 3D shape interpolation
  - Matching topology
  - Star-shaped polyhedral
  - Axial slices



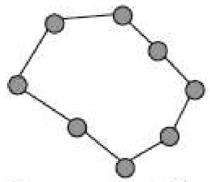
Slice from Object 1 showing
reconstructed polygon



Slice from Object 2 showing
reconstructed polygon



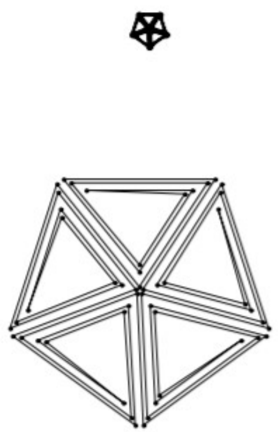
Superimposed slices showing interpolated points



Slice reconstructed from
interpolated points

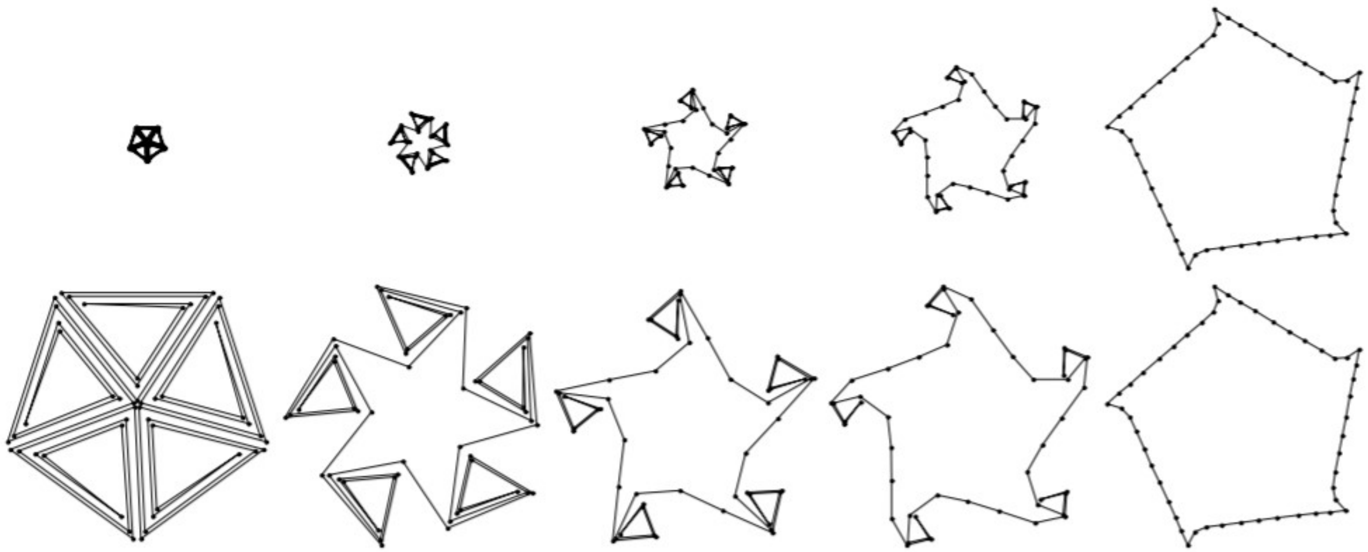
Interpolation-Based Animation

- 3D shape interpolation
  - Advanced 2D interpolation (Connelly et al. 2002)



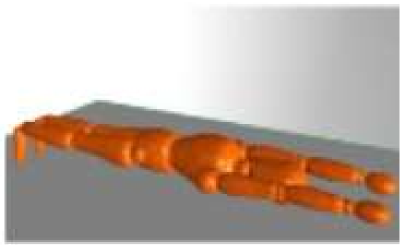
Interpolation-Based Animation

- 3D shape interpolation
  - Advanced 2D interpolation (Connelly et al. 2002)



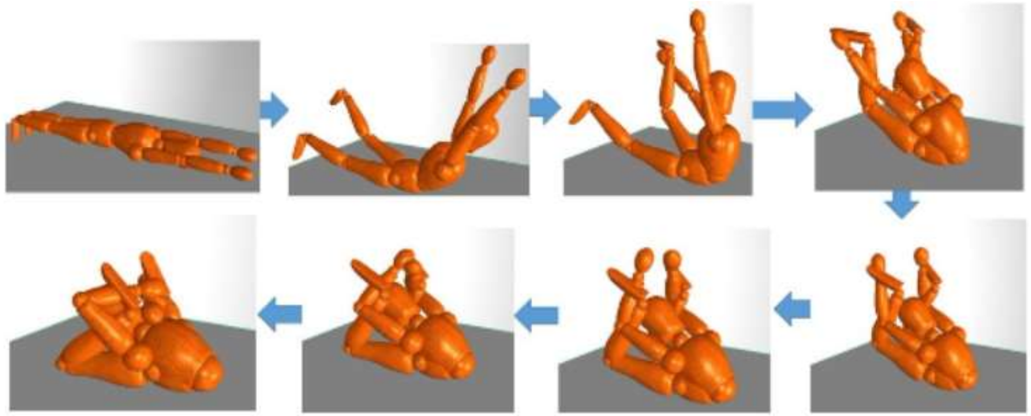
Interpolation-Based Animation

- 3D shape interpolation
  - Advanced 3D interpolation (Wang et al., 2015, An Energy-Driven Motion Planning Method for Two Distant Postures)



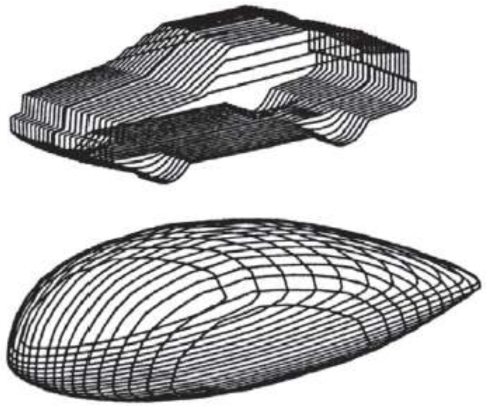
Interpolation-Based Animation

- 3D shape interpolation
  - Advanced 3D interpolation (Wang et al., 2015, An Energy-Driven Motion Planning Method for Two Distant Postures)

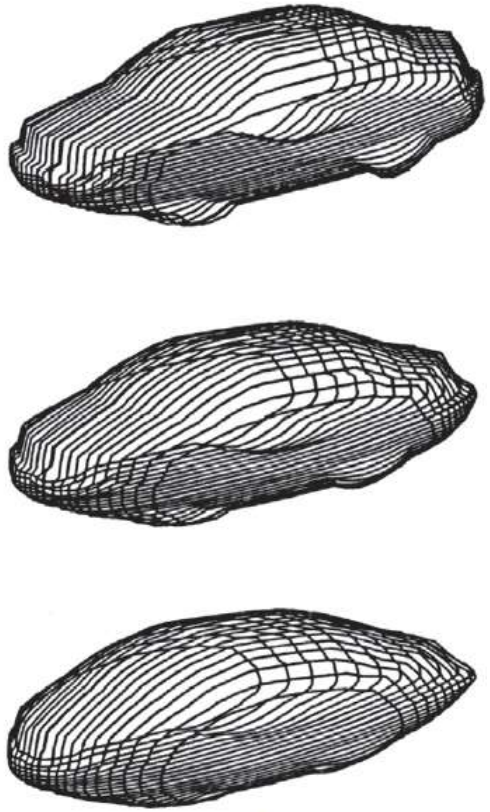


Interpolation-Based Animation

- 3D shape interpolation
  - Matching topology
  - Star-shaped polyhedral
  - Axial slices



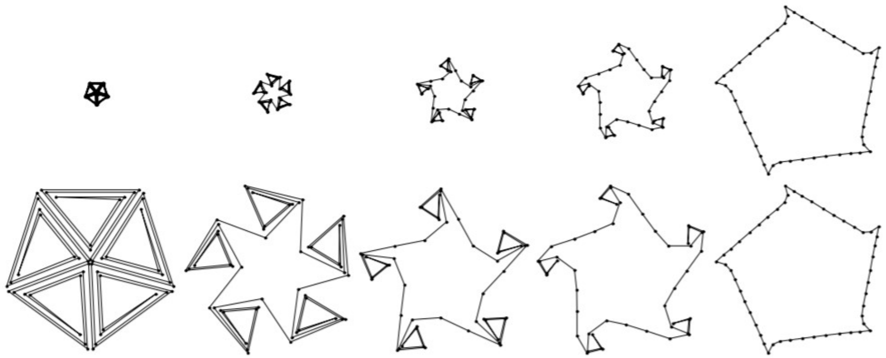
Original shapes sliced into contours



Interpolated shapes

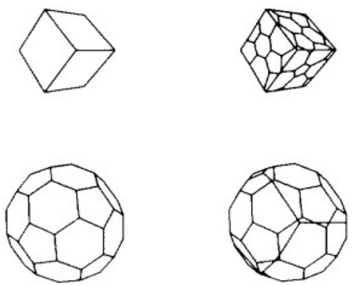
Interpolation-Based Animation

- 3D shape interpolation
  - Matching topology
  - Star-shaped polyhedral
  - Axial slices
  - Map to sphere
    - Map objects onto a unit sphere (with no overlapping!)



Interpolation-Based Animation

- 3D shape interpolation
  - Matching topology
  - Star-shaped polyhedral
  - Axial slices
  - Map to sphere
    - Map objects onto a unit sphere (with no overlapping!)
    - Genus 0 objects are easier (no holes)
    - (Siggraph 1992, Shape Transformation for Polyhedral Objects)

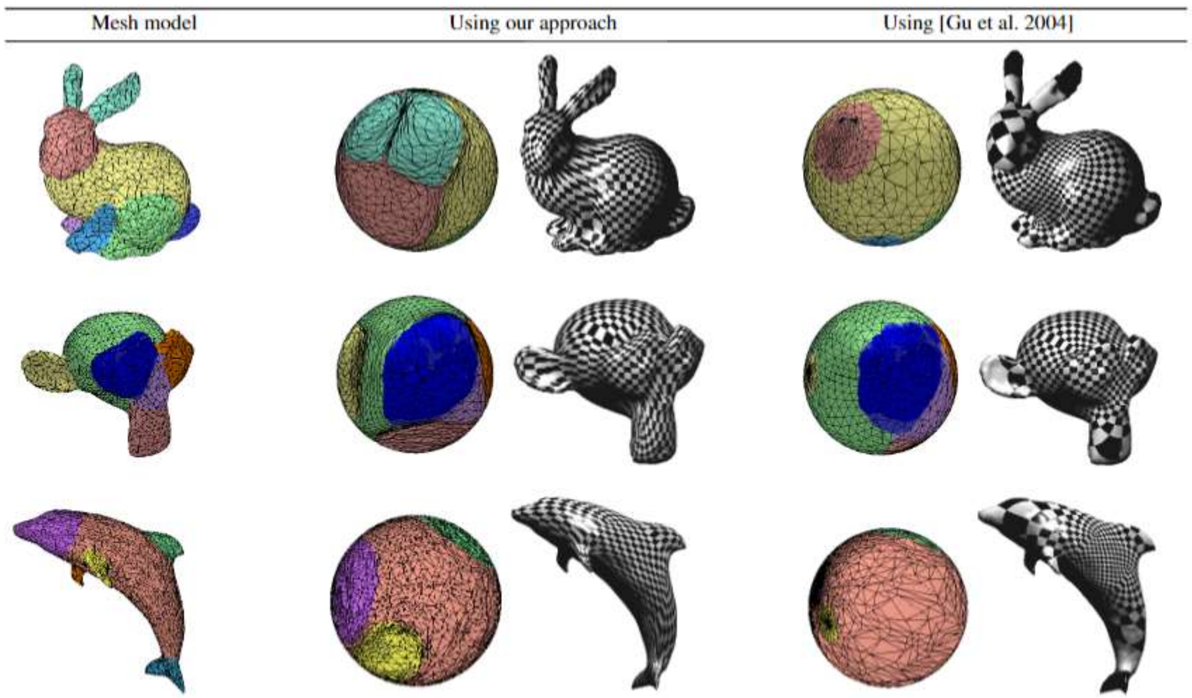


Interpolation-Based Animation

- 3D shape interpolation
  - Matching topology
  - Star-shaped polyhedral
  - Axial slices
  - Map to sphere
    - Map objects onto a unit sphere (with no overlapping!)
    - Genus 0 objects are easier (no holes)
    - (Siggraph 1992, Shape Transformation for Polyhedral Objects)
    - (Siggraph 2014, Wang et al. Harmonic Parameterization by Electrostatics)

Interpolation-Based Animation

- (Siggraph 2014, Wang et al. Harmonic Parameterization by Electrostatics)

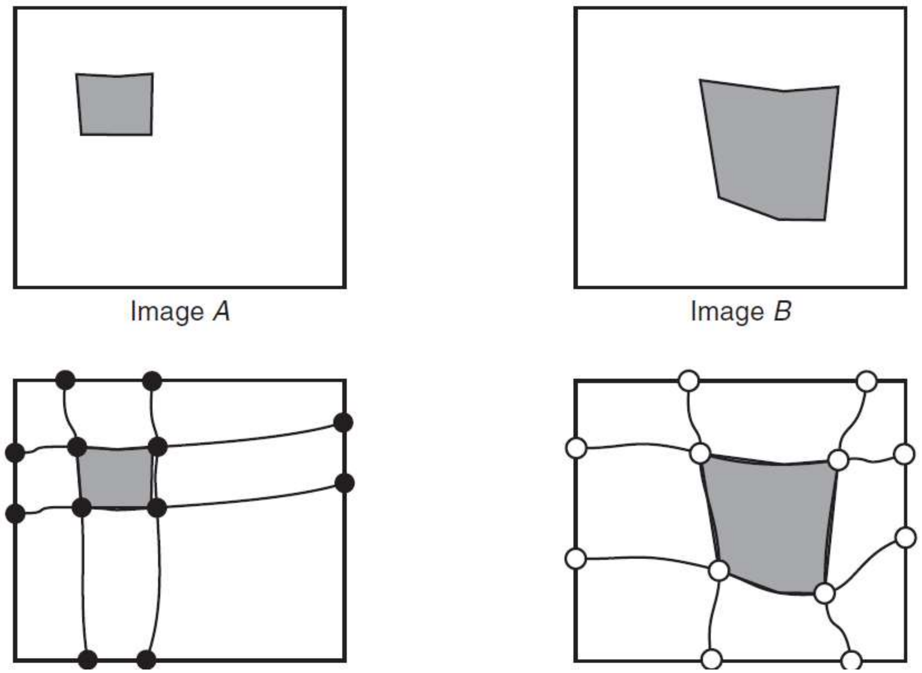


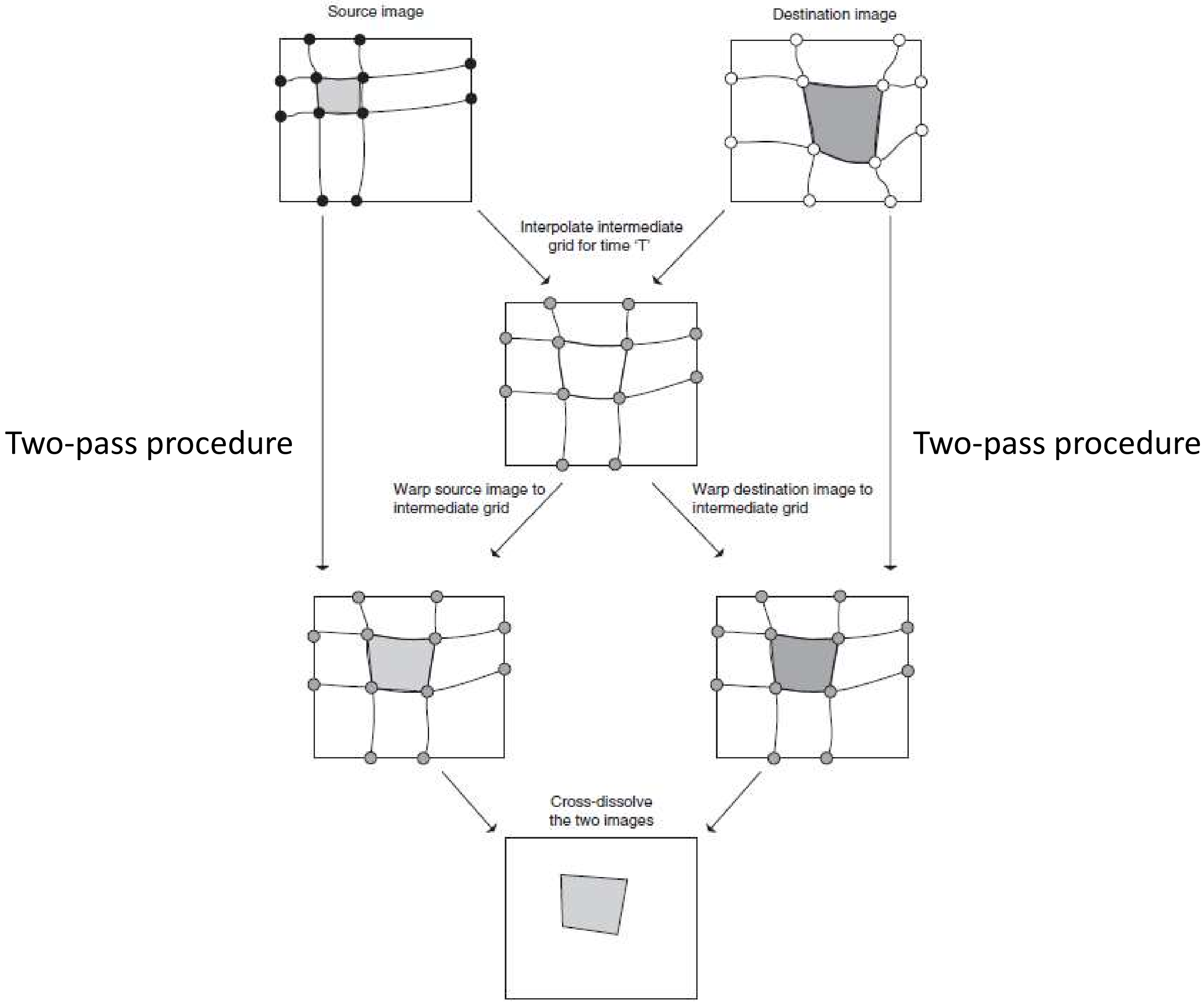
Interpolation-Based Animation

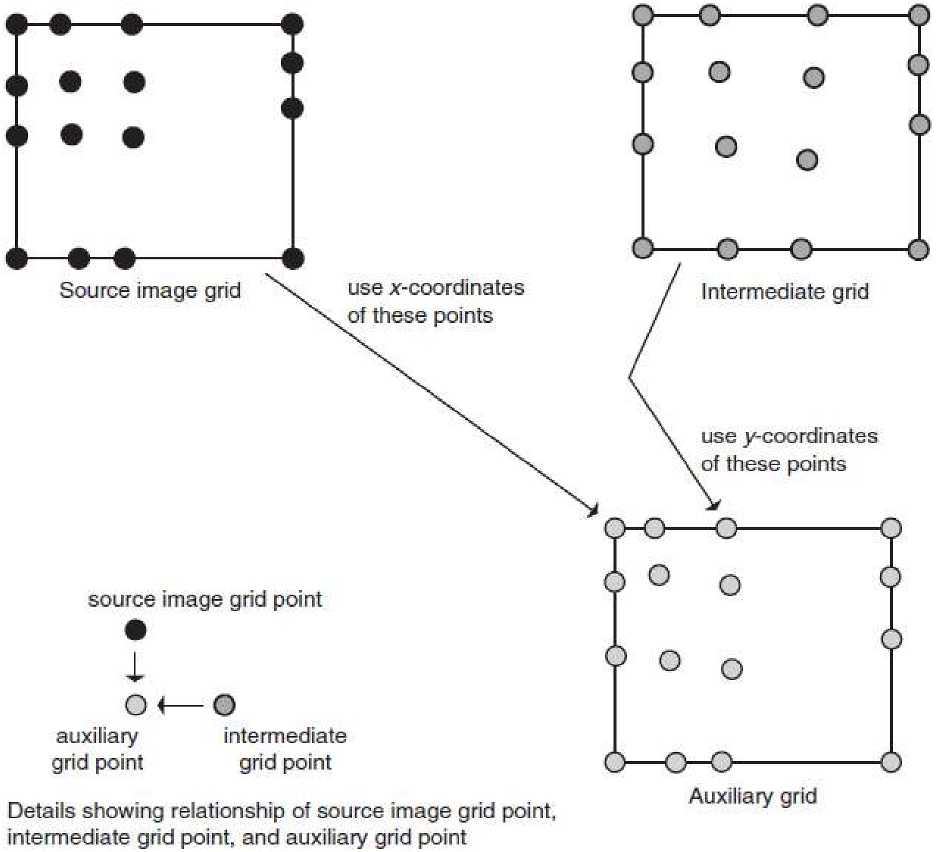
- 2D morphing
  - Morphing one image to another
    - User defined correspondence
    - Controlled transformation

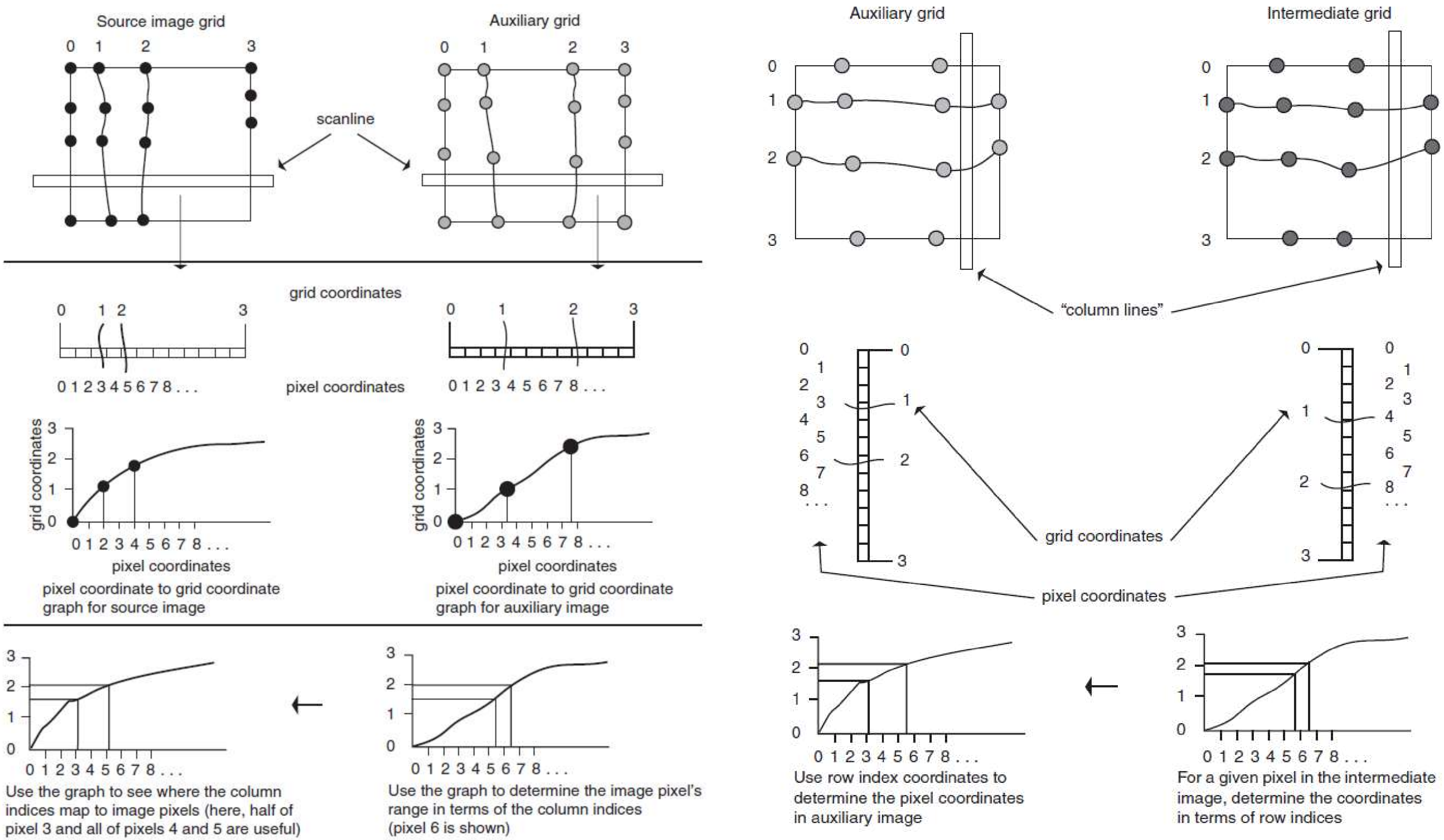
Interpolation-Based Animation

- 2D morphing
  - Coordinate grid



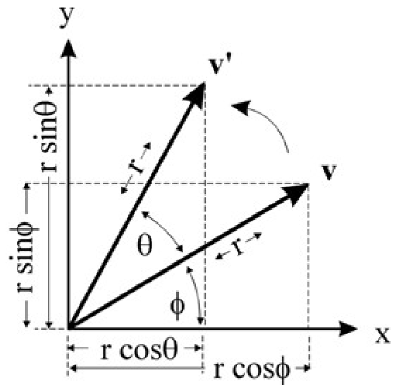

Source image
Destination image
Interpolate intermediate grid for time 'T'
Two-pass procedure
Two-pass procedure
Warp source image to intermediate grid
Warp destination image to intermediate grid
Cross-dissolve the two images

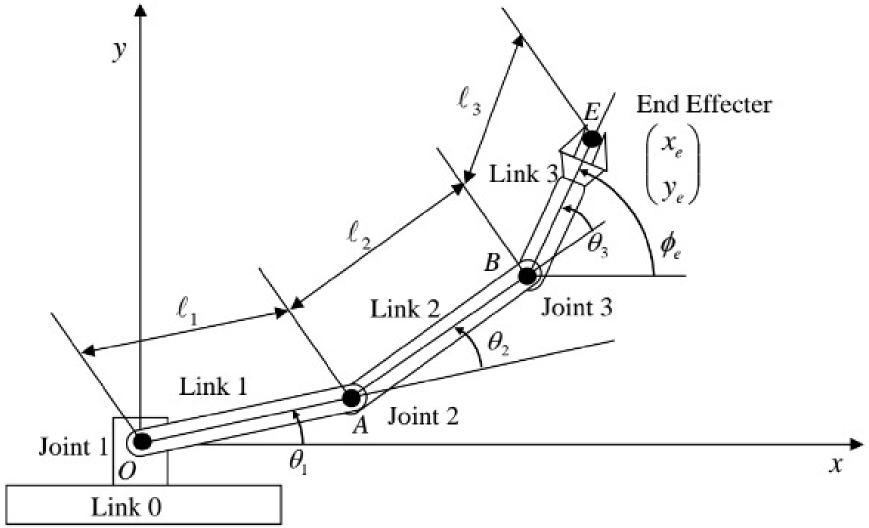

Source image grid
use x-coordinates
of these points
Intermediate grid
use y-coordinates
of these points
source image grid point
auxiliary
grid point
intermediate
grid point
Auxiliary grid
Details showing relationship of source image grid point,
intermediate grid point, and auxiliary grid point


Source image grid
Auxiliary grid
scanline
grid coordinates
pixel coordinates
grid coordinates
pixel coordinates
pixel coordinate to grid coordinate graph for source image
pixel coordinate to grid coordinate graph for auxiliary image
Use the graph to see where the column indices map to image pixels (here, half of pixel 3 and all of pixels 4 and 5 are useful)
Use the graph to determine the image pixel's range in terms of the column indices (pixel 6 is shown)
Auxiliary grid
Intermediate grid
"column lines"
grid coordinates
pixel coordinates
Use row index coordinates to determine the pixel coordinates in auxiliary image
For a given pixel in the intermediate image, determine the coordinates in terms of row indices

Kinematic Linkages

- Forward Kinematics





Clockwise

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Counter-clockwise

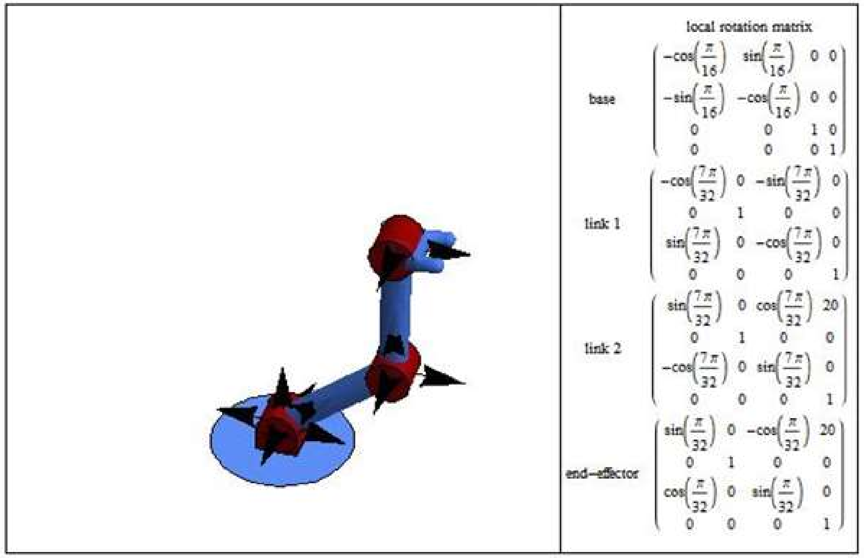
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Kinematic Linkages

- Forward Kinematics



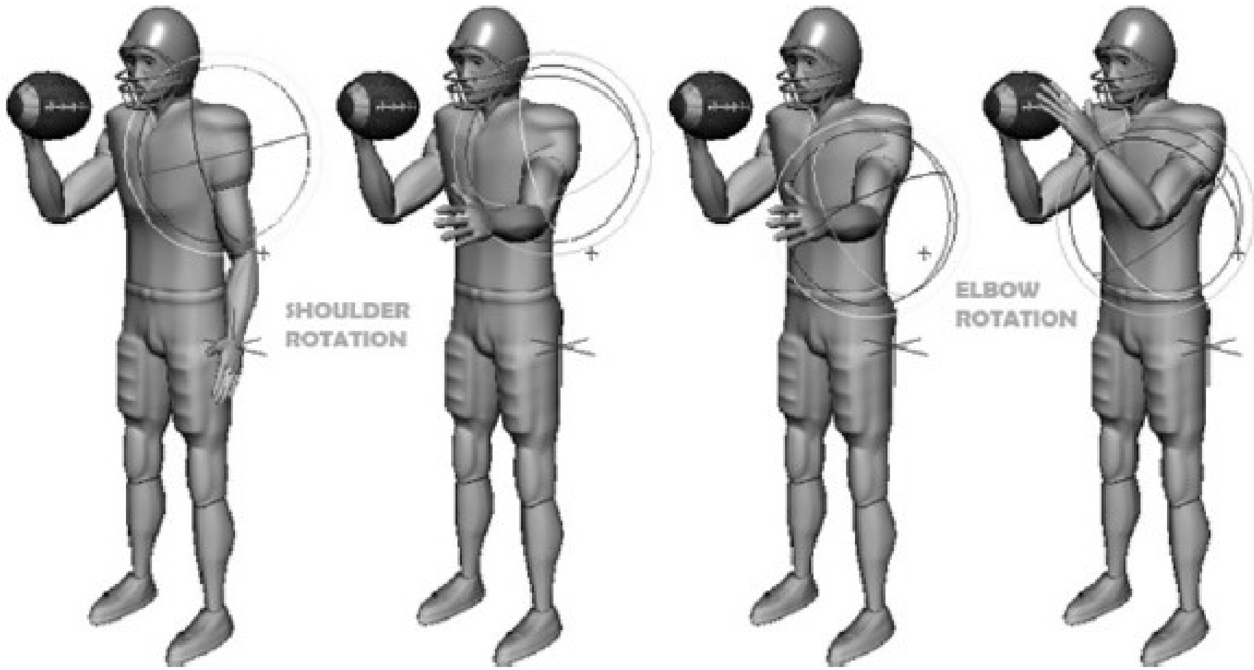
$$\mathbf{R}_X(\theta)=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}_Y(\theta)=\begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}_Z(\theta)=\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

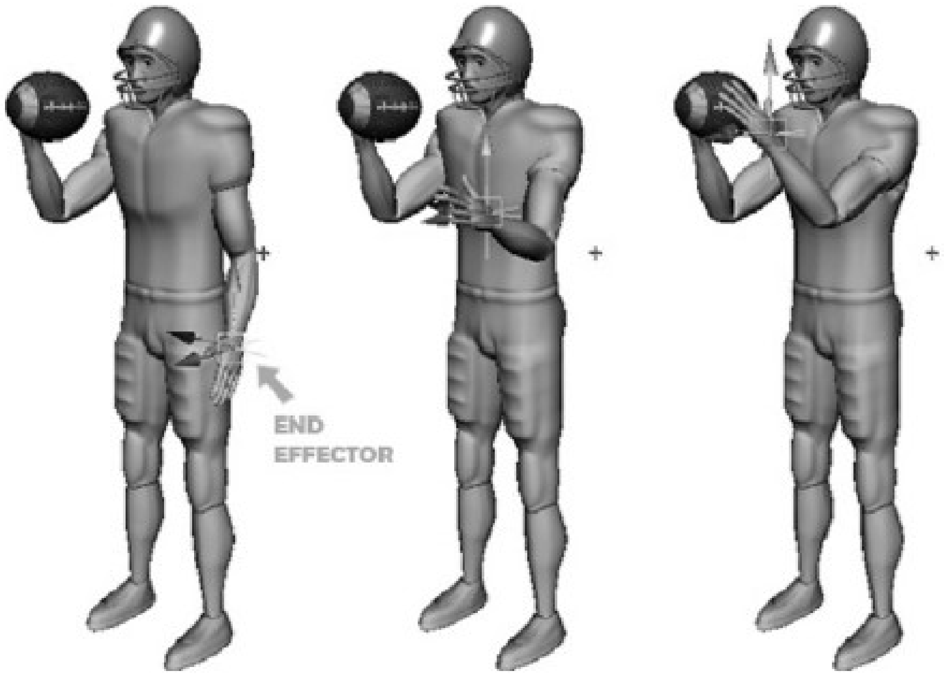
Kinematic Linkages

- Forward Kinematics



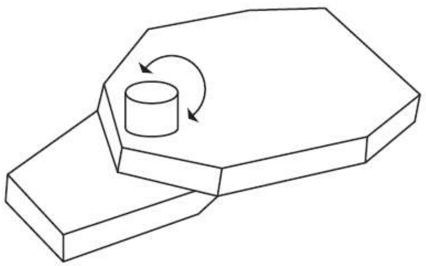
Kinematic Linkages

- Inverse Kinematics

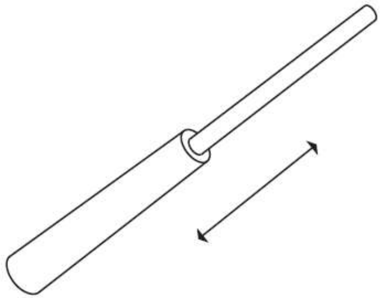


Kinematic Linkages

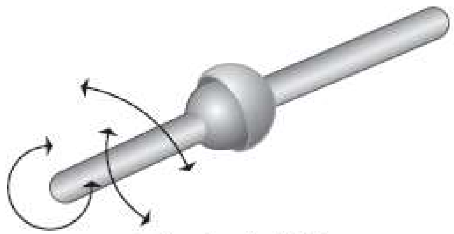
- Hierarchical Modelling
    - Joint types



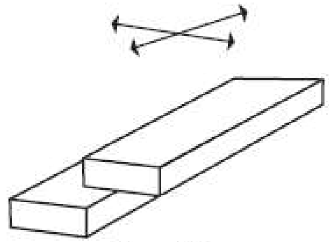
Revolute joint



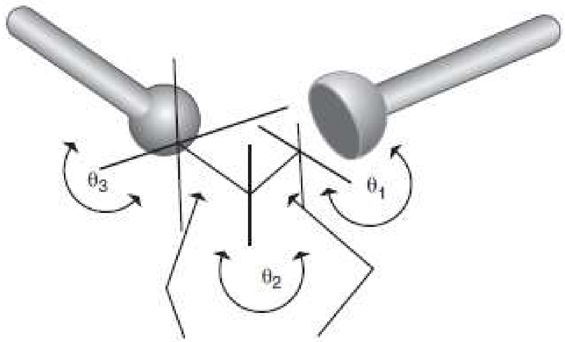
Prismatic joint



Ball-and-socket joint

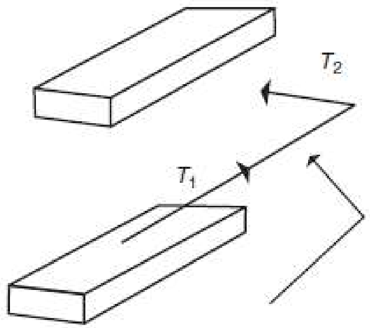


Planar joint



zero-length linkages

Ball-and-socket joint modeled as 3 one-degree joints with zero-length links

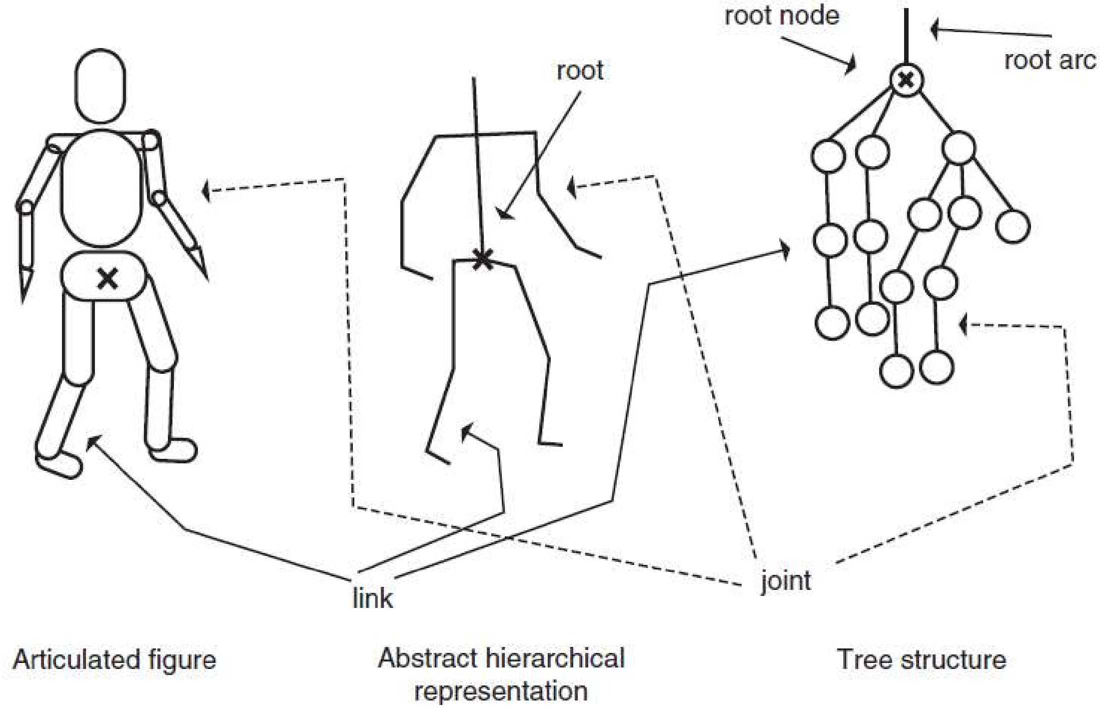


zero-length linkage

Planar joint modeled as 2 one-degree prismatic joints with zero-length links

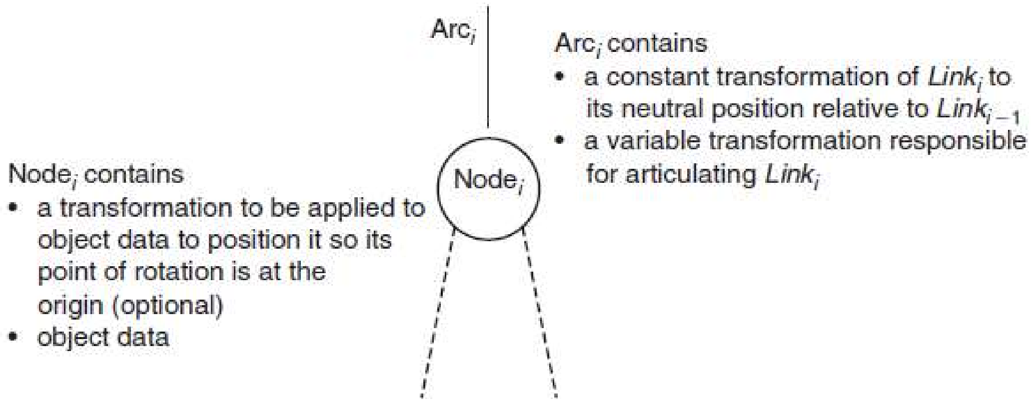
Kinematic Linkages

- Hierarchical Modelling
  - Data structure (tree)



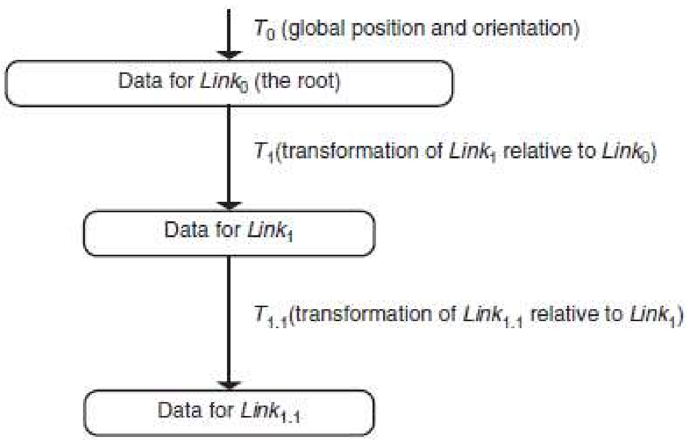
Kinematic Linkages

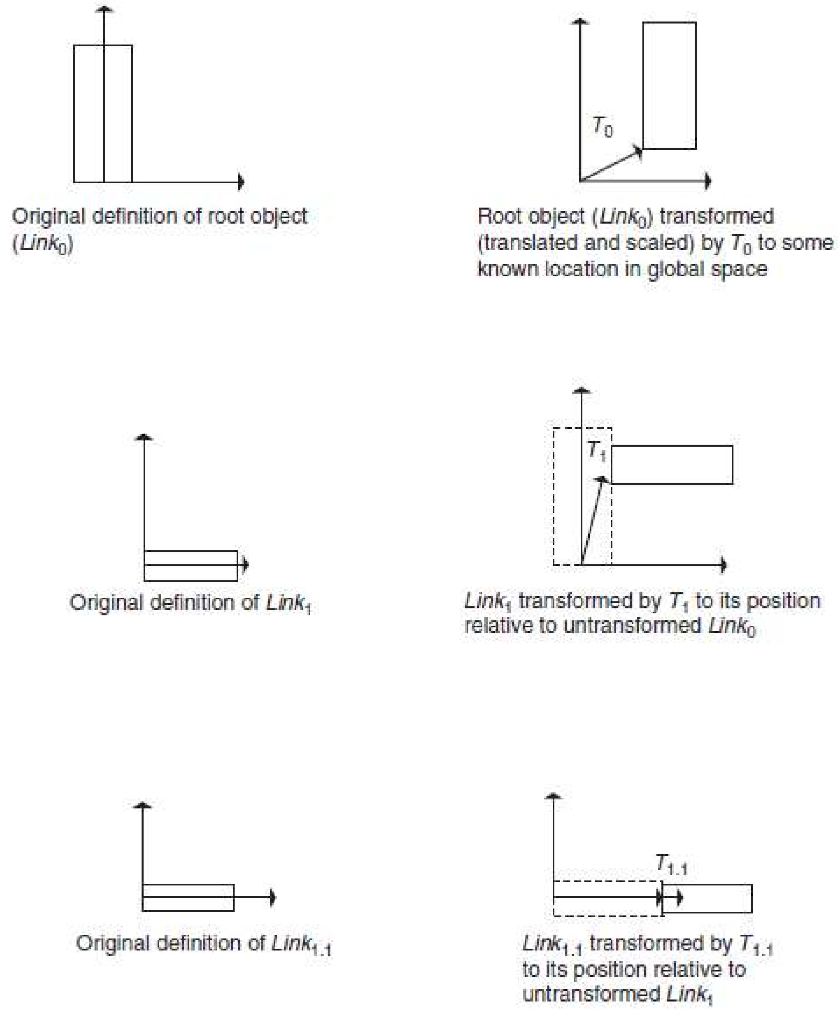
- Hierarchical Modelling
  - Data structure (tree)



Kinematic Linkages

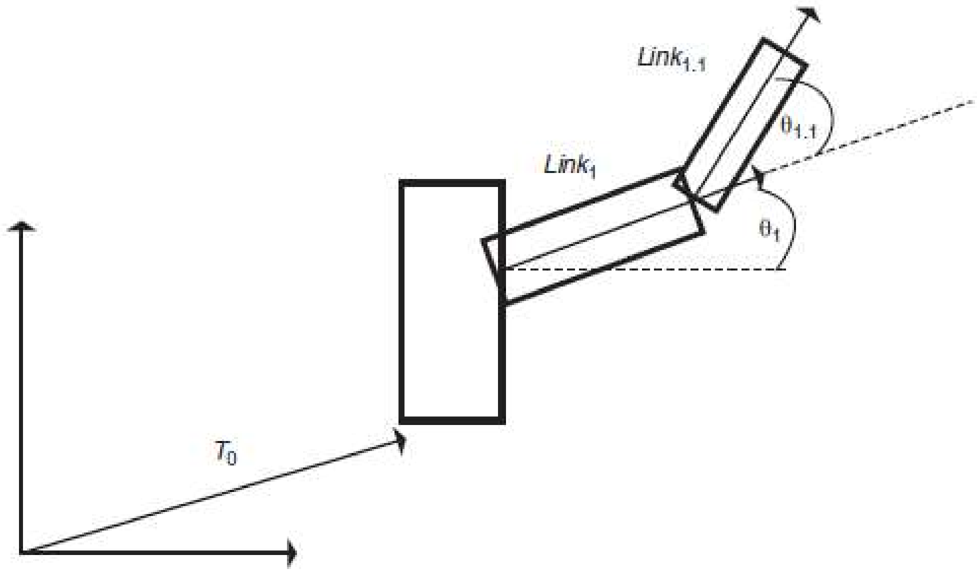
- Hierarchical Modelling
  - Data structure (tree)

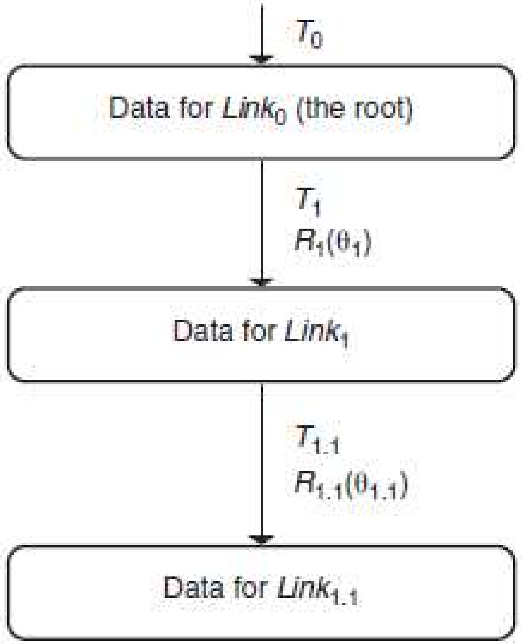




Kinematic Linkages

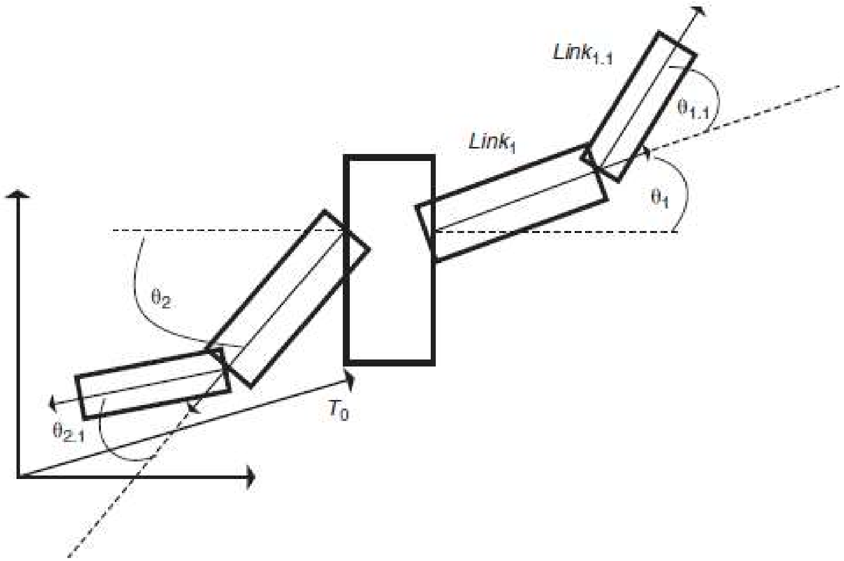
- Hierarchical Modelling
  - Data structure (tree)

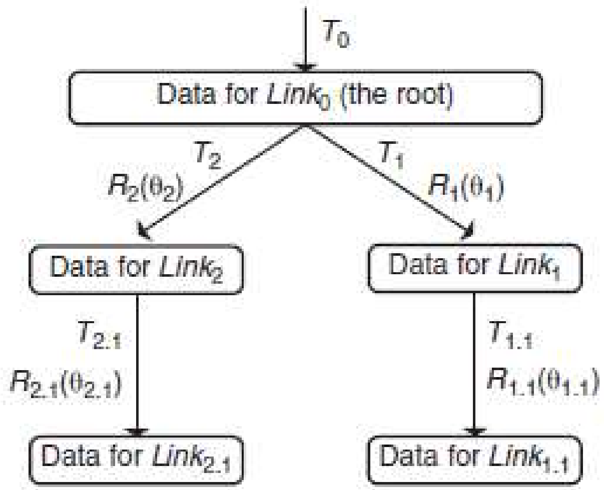




Kinematic Linkages

- Hierarchical Modelling
  - Data structure (tree)





Kinematic Linkages

- Hierarchical Modelling
  - Local coordinate frames
    - Every child link can be represented in the local coordinate of its parent link
    - Local <-> global conversions by multiplying transformation matrices along the hierarchy

Kinematic Linkages

- Hierarchical Modelling
  - Forward Kinematics

nodePtr: A pointer to a node that holds the data to be articulated by the arc.
Lmatrix: A matrix that locates the following (child) node relative to the previous (parent) node.

Amatrix: A matrix that articulates the node data; this is the matrix that is changed in order to animate (articulate) the linkage.
arcPtr: A pointer to a sibling arc (another child of this arc's parent node); this is NULL if there are no more siblings.

dataPtr: Data (possibly shared by other nodes) that represent the geometry of this segment of the figure.
Tmatrix: A matrix to transform the node data into position to be articulated (e.g., put the point of rotation at the origin).
ArcPtr: A pointer to a single child arc.

```
traverse(arcPtr,matrix)
{
   ; get transformations of arc and concatenate to current matrix
   matrix = matrix*arcPtr->Lmatrix      ; concatenate location
   matrix = matrix*arcPtr->Amatrix      ; concatenate articulation
   ; process data at node
   nodePtr = arcPtr->nodePtr            ; get the node of the arc
   push(matrix)                         ; save the matrix
   matrix = matrix * nodePtr->matrix    ; ready for articulation
   articulatedData = transformData(matrix,dataPtr) ; articulate the data
   draw(articulatedData);               ; and draw it
   matrix = pop()                       ; restore matrix for node's children
   ; process children of node
   if (nodePtr->arcPtr!= NULL) {        ; if not a terminal node
      nextArcPtr = nodePtr->arcPtr      ; get first arc emanating from node
      while (nextArcPtr != NULL) {      ; while there's an arc to process
         push(matrix)                   ; save matrix at node
         traverse(nextArcPtr,matrix)    ; traverse arc
         matrix = pop()                 ; restore matrix at node
         nextArcPtr = nextArcPtr->arcPtr  ; set next child of node
      }
   }
}
```