Computer Animation & Physical Simulation

Lecture 3: Non-Physically-Based Animation I

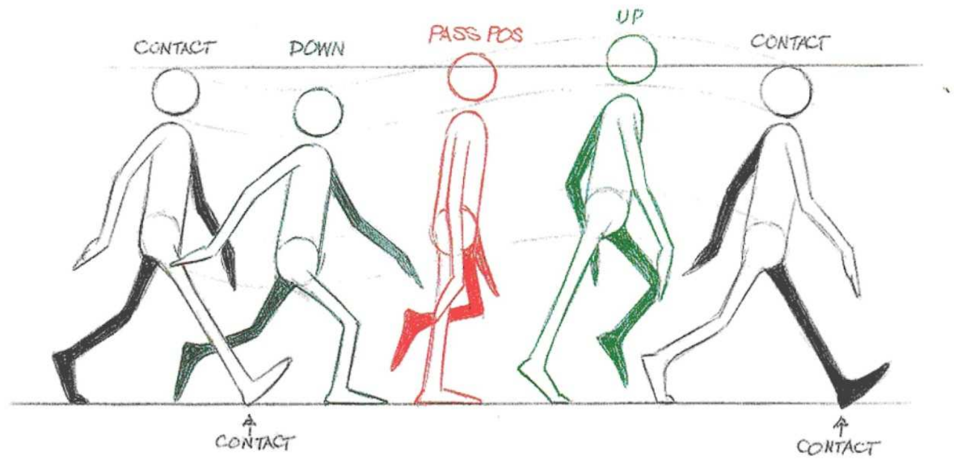
XIAOPEI LIU
School of Information Science and Technology
ShanghaiTech University

Keyframing

- **What is a frame?**
  - One of the many still images which compose the complete *moving picture*
- **What is a keyframe?**
  - A drawing that defines the critical frames of any smooth transition
  - The remaining frames are filled with inbetweens
    - Inbetweening or tweening
      - The process of generating intermediate frames between two images
      - Manually render or adjust inbetween frames by hand
      - Or automatically render inbetween frames using interpolation

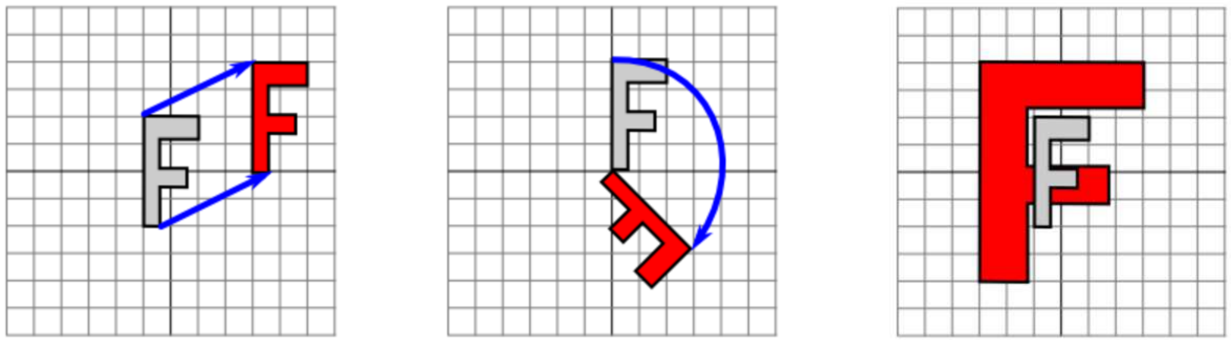
Keyframing

- **An example of keyframe-based animation**
  - Interpolating inbetween frames from keyframes
    - Interpolating critical properties or features



2D Transformations

- **Rigid transformation**
  - Translation
  - Rotation
  - Scaling



Translation Rotation Scaling

Homogeneous Coordinates

- **Given a frame defined by (p, $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$)**
  - Ambiguity between the representation of a point $[p_x, p_y, p_z]^T$ and a vector $[v_x, v_y, v_z]^T$
  - We can write the point as the inner product $[s_1, s_2, s_3, 1][\mathbf{v}_1 , \mathbf{v}_2 , \mathbf{v}_3 , \mathbf{p}_o]^T$
  - We can write the vector as the inner product $[s'_1, s'_2, s'_3, 0][\mathbf{v}_1 , \mathbf{v}_2 , \mathbf{v}_3 , \mathbf{p}_o]^T$
  - These four vectors of three s_i values and a zero or one are called the *homogeneous representations* of the point and the vector

2D Transformations

- **Identity transformation**
  - This transformation is represented by the identity matrix
  - It maps each point and each vector to itself

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2D Transformations

- **Translation transformation**
  - In matrix form, the translation transformation is

$$\mathrm{T}(\Delta x, \Delta y) = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

  - When we consider the operation of a translation matrix on a vector: unchanged as expected

$$\begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}$$

2D Transformations

- **Rotation transformation**
  - Rotation by an angle θ about the z-axis

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

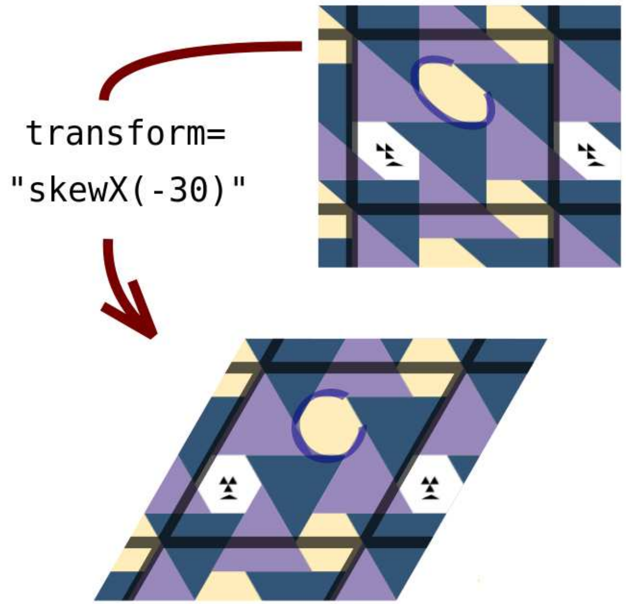
2D Transformation

- **Scaling transformation**
  - Take a point or vector and multiply its components by scale factors in x, y
  - Differentiate between uniform scaling and non-uniform scaling

$$\mathrm{S}(x, y) = \begin{pmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2D Transformations

- **Affine transformation**
  - A function between affine spaces which preserves points, straight lines and planes
  - An affine transformation does not necessarily preserve angles
  - Typically involve
    - Translation
    - Rotation
    - Scaling
    - Shearing



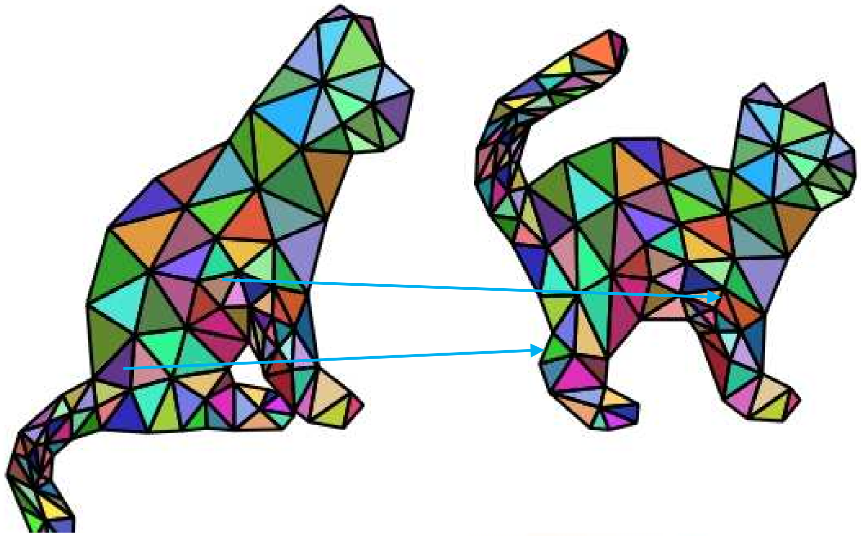
2D Transformation

- **Problem with rigid/affine transformation**
  - Objects cannot have locally varying deformation

- **Non-affine transformation**
  - Based on piecewise affine transformation
    - Triangle-based transformation
  - Based on non-linear transformation
    - Radial basis functions

Triangle-based Deformation

- **Transformation between triangles**
  - Establish correspondence between triangles



Radial Basis Functions

- **A real-valued function**
  - Function value depends only on the distance from some other center points $\boldsymbol{c}_i$

$$\phi(\mathbf{x}, \mathbf{c}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$$

- **RBF Types**
  - Gaussian: $\phi(r) = e^{-(\varepsilon r)^2}$
  - Multiquadric: $\phi(r) = \sqrt{1 + (\varepsilon r)^2}$
  - Thin-plate spline: $\phi(r) = r^2 \ln(r)$

Radial Basis Functions

- **Function interpolation**
  - Mathematical form

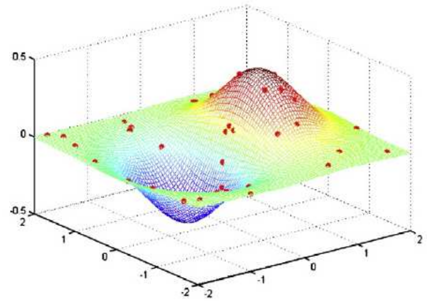
$$f(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|)$$

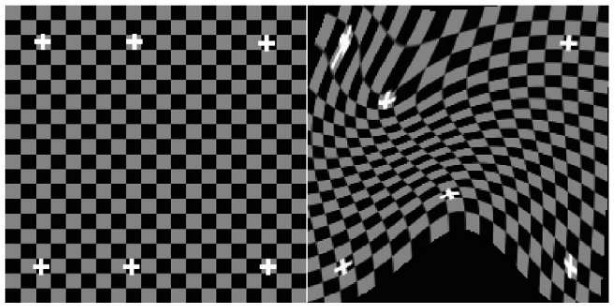
  - Solving the interpolation

$$\begin{bmatrix} \phi(||\mathbf{x}_1 - \mathbf{x}_1||) & \phi(||\mathbf{x}_1 - \mathbf{x}_2||) & \cdots & \phi(||\mathbf{x}_1 - \mathbf{x}_N||) \\ \phi(||\mathbf{x}_2 - \mathbf{x}_1||) & \phi(||\mathbf{x}_2 - \mathbf{x}_2||) & \cdots & \phi(||\mathbf{x}_2 - \mathbf{x}_N||) \\ \vdots & \vdots & & \vdots \\ \phi(||\mathbf{x}_N - \mathbf{x}_1||) & \phi(||\mathbf{x}_N - \mathbf{x}_2||) & \cdots & \phi(||\mathbf{x}_N - \mathbf{x}_N||) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}$$

RBF-based Deformation

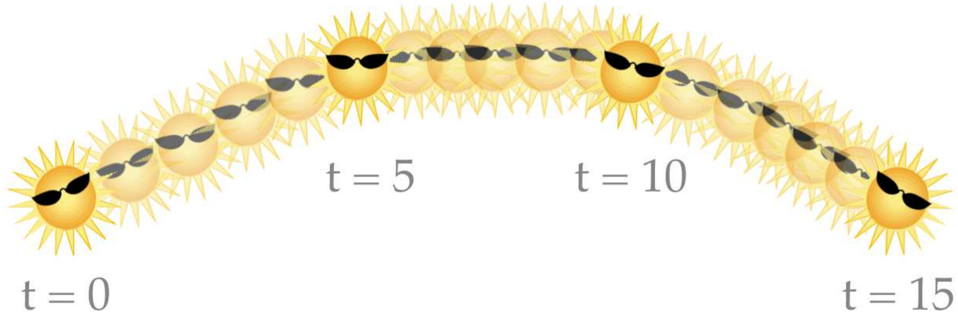
- **Solving deformation field**
  - Based on corresponding points
  - Use RBF interpolation for deformation field





Interpolation

- **Linearly interpolate the parameters between keyframes**



end key start key

$$x = x_0 + \frac{t - t_0}{t_1 - t_0}(x_1 - x_0)$$

end time start time

Interpolation

- **Cubic curve interpolation**
  - We can use three cubic functions to represent a 3D curve
  - Each function is defined within the range $0 \leq t \leq 1$

$$\mathbf{Q}(t) = [x(t) \;\; y(t) \;\; z(t)]$$

or

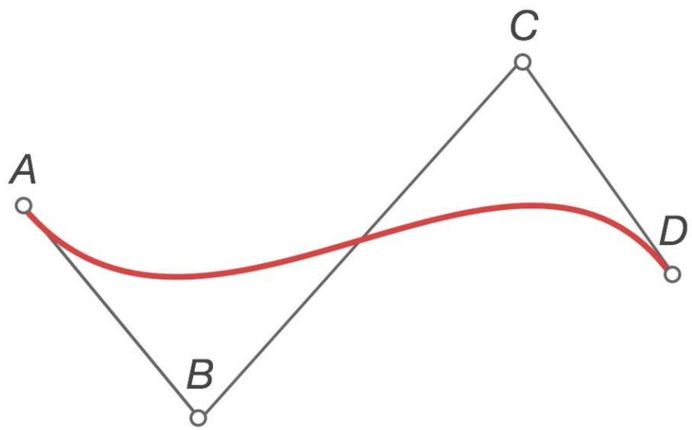
$$Q_x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$Q_y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$Q_z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

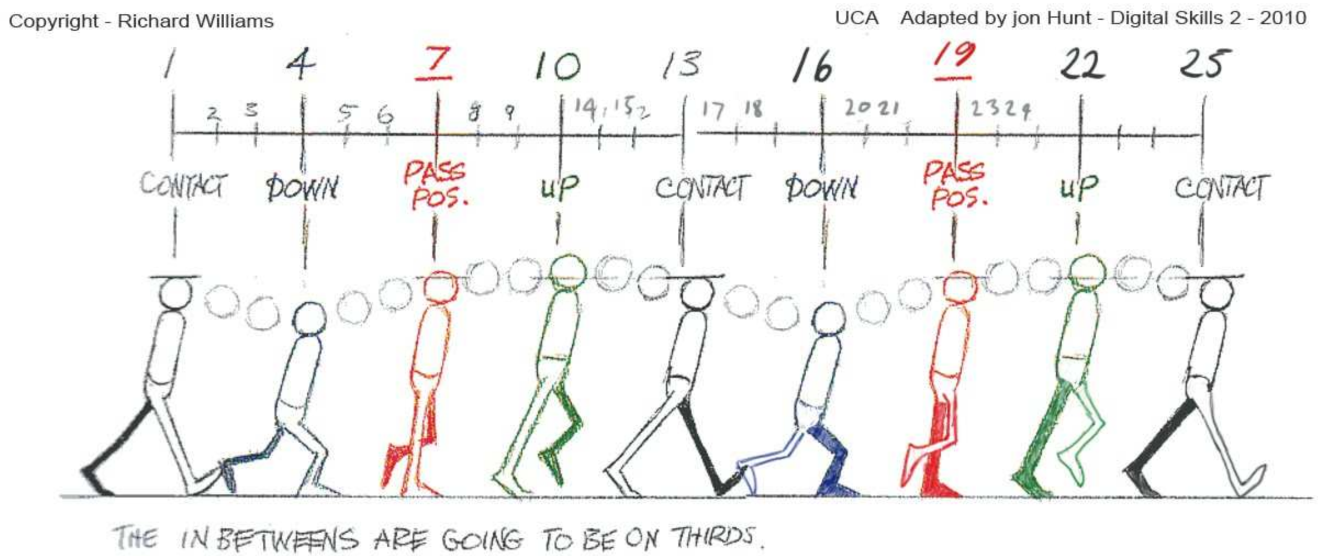
Interpolation

- **Spline interpolation**
  - We may like to have a spline interpolating some of the control points
  - Bezier curves do not necessarily pass through all the control points



Interpolating Keyframes from Spline Curve

- **B-Spline interpolation of features (critical points)**



I. Image Warping & Morphing

Image Warping

- **The process of digitally manipulating an image**
  - Any shapes portrayed in the image have been significantly distorted
  - Used for correcting image distortion as well as for creative purposes

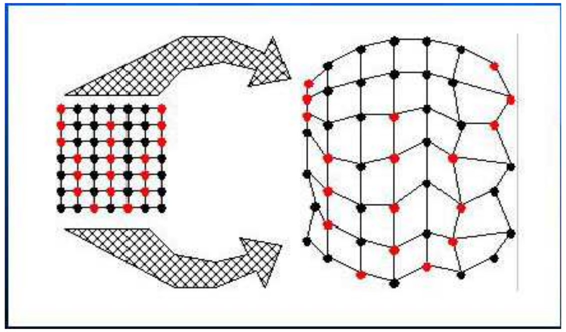


Image Morphing

- **Cross-dissolve**
  - Interpolate whole images:
    - $Image_{halfway}$ = (1-t)*Image1 + t*image2

Image Morphing

- **Morphing procedure**
  - Warping + cross-dissolve

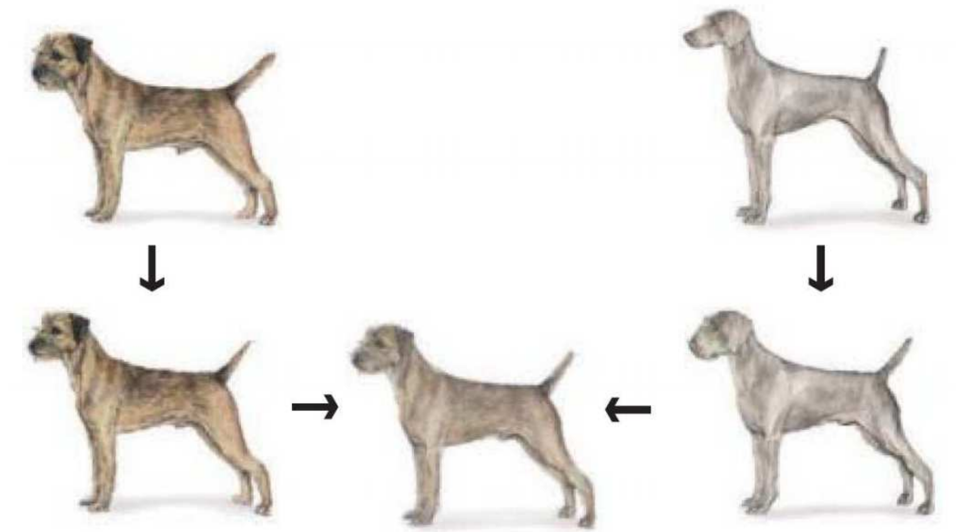


Image Morphing

- **How can we specify the warp?**
  - Specify corresponding points
  - Feature points

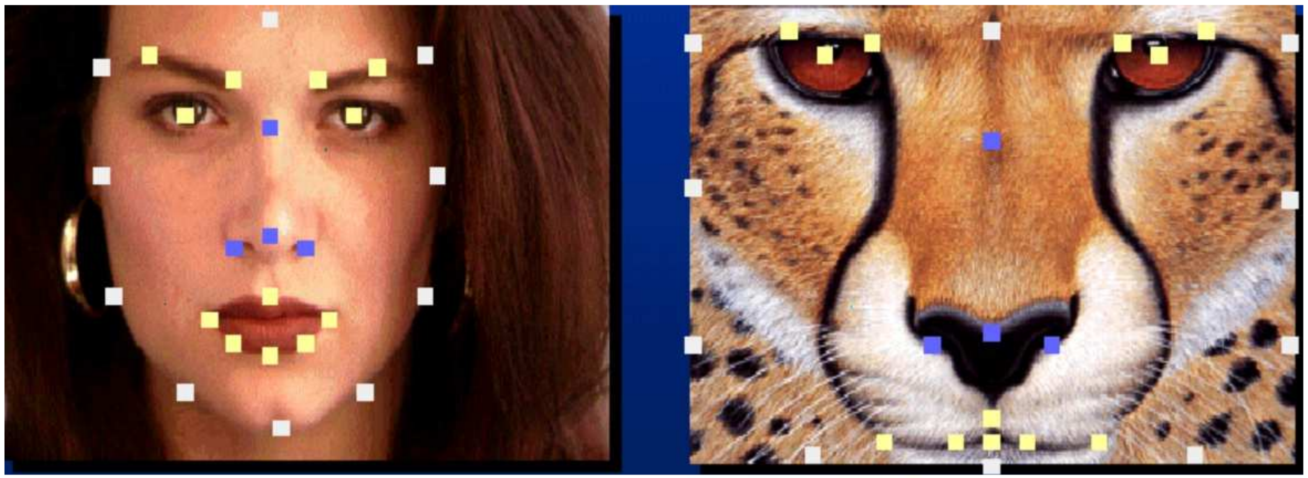


Image Morphing

- **The whole procedure**
  - 1. Create an intermediate shape (by interpolation)
  - 2. Warp both images towards it
  - 3. Cross-dissolve the colors in the newly warped images

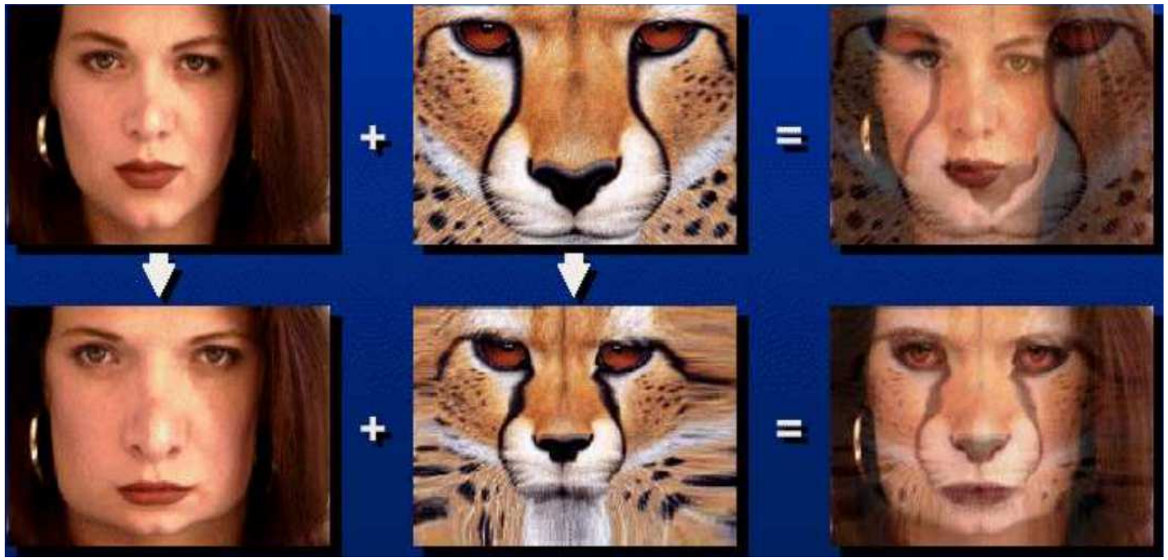


Image Morphing

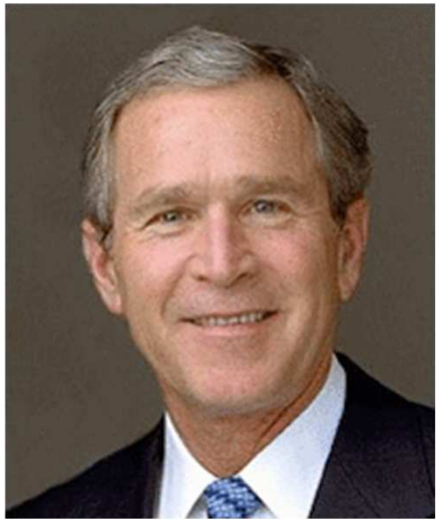
- **Different interpolation parameters**
  - Sample interpolation parameters in [0,1]

(c) $\alpha = 0.0$ (d) $\alpha = 0.2$ (e) $\alpha = 0.4$

(f) $\alpha = 0.6$ (g) $\alpha = 0.8$ (h) $\alpha = 1.0$

Image Morphing

- **A special effect in motion pictures**
  - Most often used to depict one person turning into another object
  - Feature matching with image warping/blending

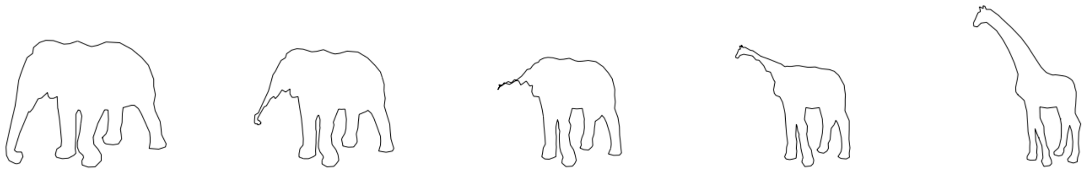


Generating Intermediate Morphs

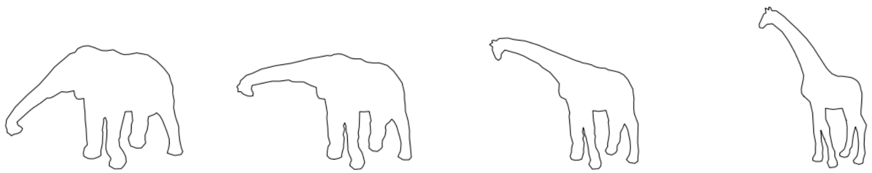
- **As-Rigid-As-Possible shape interpolation**
  - Blends the interiors of given shapes
  - Rigid in the sense that local volumes are least-distorting

Generating Intermediate Morphs

- **Simplest method**
  - Vertex coordinate interpolation



Vertex coordinate interpolation

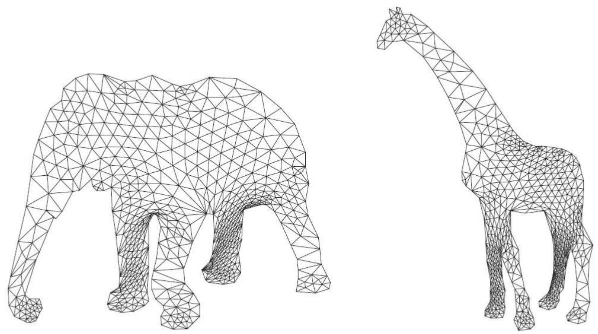


As-rigid-as-possible shape interpolation

Generating Intermediate Morphs

- **Least-distorting triangle-to-triangle morphing**
  - An affine mapping from source to target triangles represented by

$$A\vec{p_i} + \vec{l} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \vec{p_i} + \boxed{\cancel{\begin{pmatrix} l_x \\ l_y \end{pmatrix}}} = \vec{q_i}, \quad i \in \{1, 2, 3\}$$



  - Simplest solution

$$A(t) = (1 - t)I + tA$$

Generating Intermediate Morphs

- **Least-distorting triangle-to-triangle morphing**
  - Some properties of the affine transformation
    - Symmetric with respect to t
    - Rotational angles and scales should change linearly
    - Triangle should keep its orientation (no flipping)
    - The resulting vertex paths should be simple

Generating Intermediate Morphs

- **Least-distorting triangle-to-triangle morphing**
  - Decomposition
    - Singular value decomposition (SVD)

$$A = R_\alpha D R_\beta = R_\alpha \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} R_\beta, \quad s_x, s_y > 0$$

    - A decomposition into <u>single rotation</u> and <u>a symmetric matrix</u> creates visually the best transformation

Generating Intermediate Morphs

- **Least-distorting triangle-to-triangle morphing**
  - Decomposition
    - Variant from SVD

$$
\begin{aligned}
A \quad &= \quad R_\alpha D R_\beta = R_\alpha (R_\beta R_\beta^T) D R_\beta = \\
&(R_\alpha R_\beta)(R_\beta^T D R_\beta) = R_\gamma S = R_\gamma \begin{pmatrix} s_x & s_h \\ s_h & s_y \end{pmatrix}
\end{aligned}
$$

Generating Intermediate Morphs

- **Least-distorting triangle-to-triangle morphing**
  - Interpolation

$$
\begin{aligned}
A \quad &= \quad R_\alpha D R_\beta = R_\alpha (R_\beta R_\beta^T) D R_\beta = \\
&(R_\alpha R_\beta)(R_\beta^T D R_\beta) = R_\gamma S = R_\gamma \begin{pmatrix} s_x & s_h \\ s_h & s_y \end{pmatrix}
\end{aligned}
$$

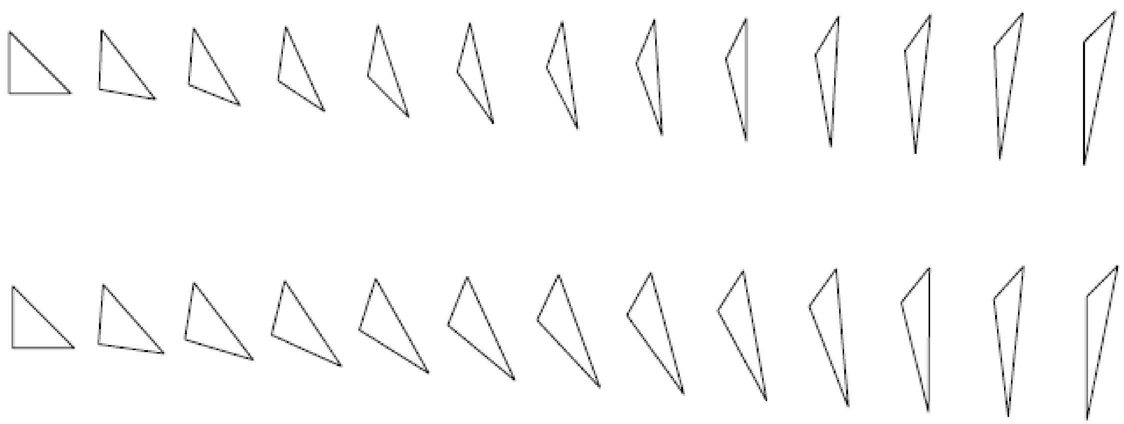
$$\Downarrow$$

$$A_{\alpha,\beta}(t) = R_{t\alpha}((1-t)I + tD)R_{t\beta}$$

$$A_\gamma(t) = R_{t\gamma}((1-t)I + tS)$$

Generating Intermediate Morphs

- **Least-distorting triangle-to-triangle morphing**
  - Comparison for interpolation



Generating Intermediate Morphs

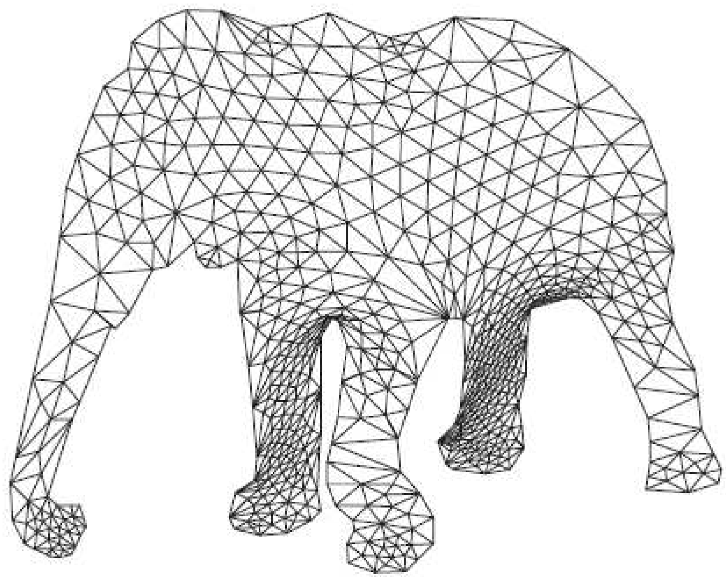
- **Closed-form vertex paths for a triangulation**

$$B_{\{i,j,k\}}(t)\vec{p_f} + \vec{l} = \vec{v_f}(t), \quad f \in \{i,j,k\}$$

Note that B can be represented by vertices

Error functional:

$$E_{V(t)} = \sum_{\{i,j,k\}\in\mathcal{T}} \left\| A_{\{i,j,k\}}(t) - B_{\{i,j,k\}}(t) \right\|^2$$



Generating Intermediate Morphs

- **Closed-form vertex paths for a triangulation**

$$E_{V(t)} = u^T \begin{pmatrix} c & G^T \\ G & H \end{pmatrix} u \quad \Rightarrow \quad H \begin{pmatrix} v_{2_x}(t) \\ v_{2_y}(t) \\ \vdots \end{pmatrix} = -G$$

$$V(t) = -H^{-1} G(t)$$

In practice, we compute the LU decomposition

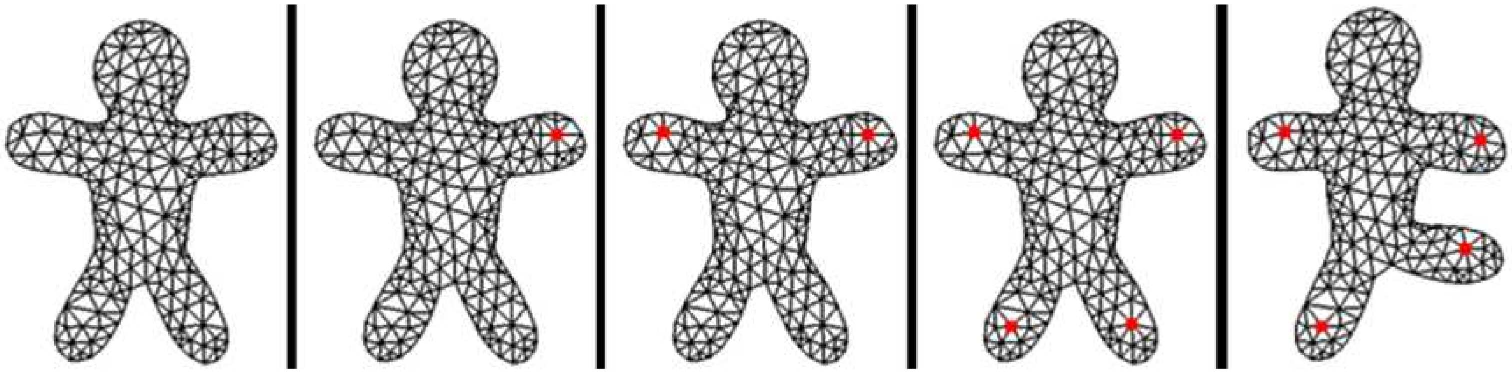
Generating Intermediate Morphs

- **Extension to 3D shape interpolations**

II. Animation by 2D Mesh Deformation

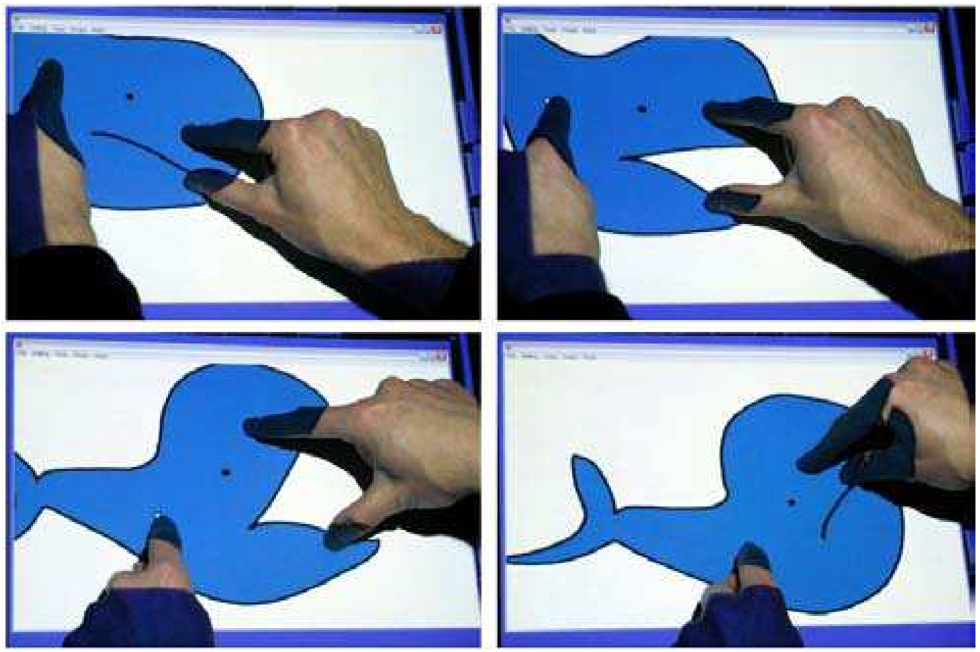
2D Mesh Deformation

- **Why 2D deformation?**
  - To make new keyframes
  - To interpolate intermediate frames



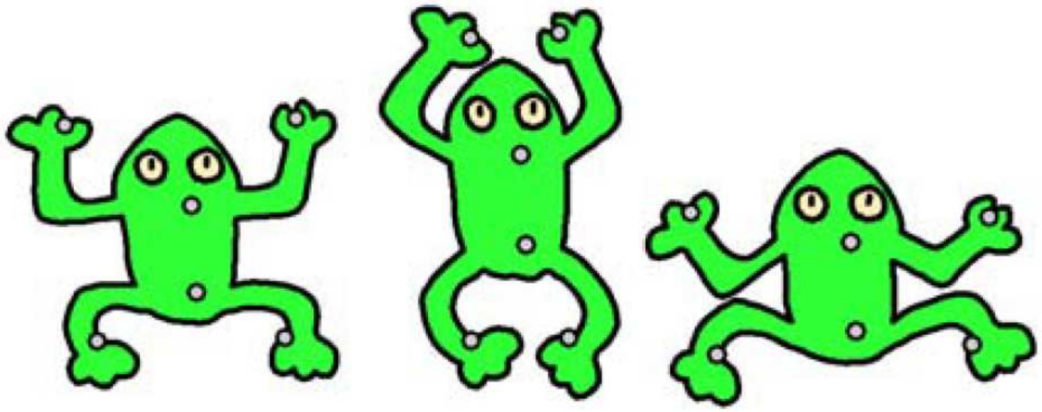
2D Mesh Deformation

- **Mesh-based method**
  - Anchor-point-based deformation
    - Specify very few anchor (feature) points



2D Deformation

- **Mesh-based method**
  - As-rigid-as-possible deformation
    - Compute the deformation based on the change of anchor points
      - Minimize overall distortion

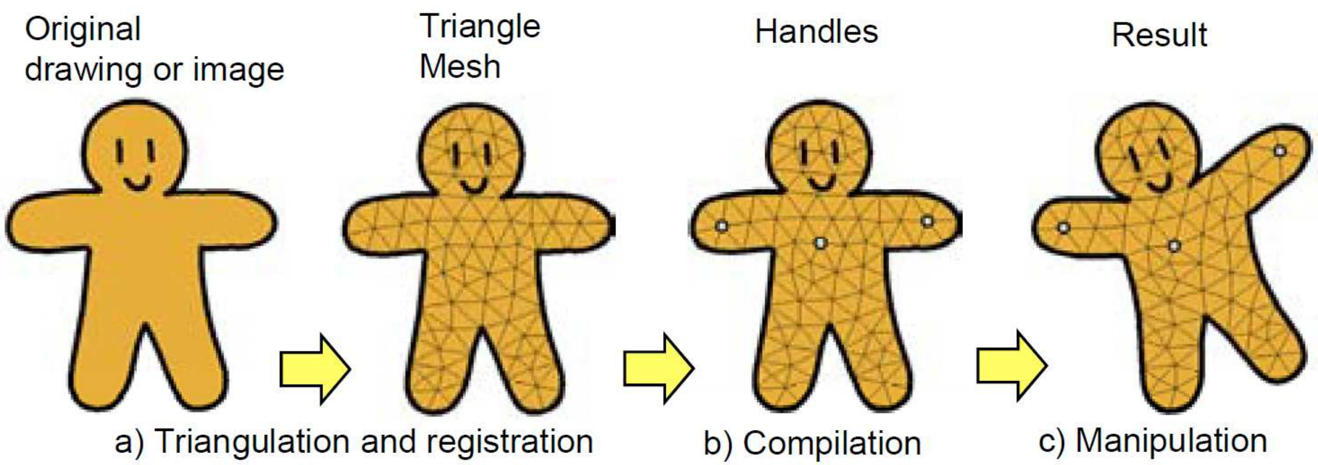


2D Deformation

- **As-rigid-as-possible deformation**
  - General idea
    - Given the coordinates of the constrained vertices (anchor points)
    - **Step 1:generate intermediate result**
      - Prevent shearing and non-uniform stretching
      - Permit rotation and uniform scaling
    - **Step 2: Adjust the scale of each triangle**

2D Deformation

- **As-rigid-as-possible deformation**
  - General idea
    - Process overview



2D Deformation

- **As-rigid-as-possible deformation**
  - Step 1: scale-free construction
    - Generate an intermediate result
      - By minimizing an error function that allows rotation and uniform scaling
    - Input: xy-coordinates of the constrained vertices
    - Output: the xy-coordinates of the remaining free vertices

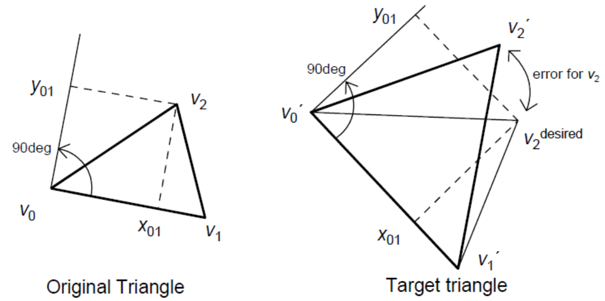
2D Deformation

- **As-rigid-as-possible deformation**
  - Step 1: scale-free construction
    - The error function between rest triangle $\{v_0, v_1, v_2\}$ a deformed triangle $\{v_0', v_1', v_2'\}$
      - First compute relative coordinates $\{x_{01}, y_{01}\}$

$$v_2 = v_0 + x_{01}\overrightarrow{v_0 v_1} + y_{01} R_{90} \overrightarrow{v_0 v_1}$$

      - Given v_0', v_1', x_{01}, and y_{01}, the system can compute the desired location for v_2'.

$$v_2^{desired} = v_0' + x_{01}\overrightarrow{v_0' v_1'} + y_{01} R_{90} \overrightarrow{v_0' v_1'} \text{ where } R_{90} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$



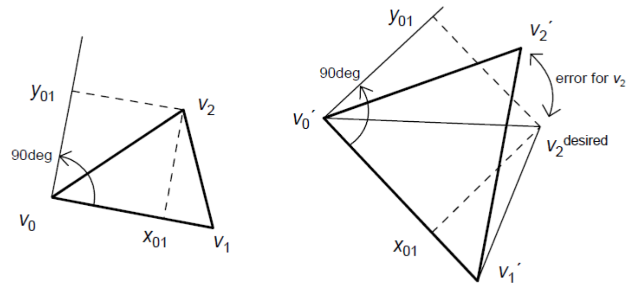
2D Deformation

- **As-rigid-as-possible deformation**
  - Step 1: scale-free construction
    - The error associated with v_2' is then represented as

$$E_{\{v_2\}} = \left\| v_2^{\text{desired}} - v_2' \right\|^2$$

    - We can define v_0^{desired} and v_1^{desired} similarly, and then define the error associated with the triangle as

$$E_{\{v_0, v_1, v_2\}} = \sum_{i=1,2,3} \left\| v_i^{\text{desired}} - v_i' \right\|^2$$



2D Deformation

- **As-rigid-as-possible deformation**
  - Step 1: scale-free construction
    - The error for the entire mesh is the sum of errors for all triangles
    - Since the error metric is quadratic in $v'=(v_{0x}', v_{0y}', \dots, v_{nx}', v_{ny}')^T$, we can express it:

$$E_{1\{\mathbf{v}'\}} = \mathbf{v}'^{\mathrm{T}} \mathbf{G} \mathbf{v}'$$

2D Deformation

- **As-rigid-as-possible deformation**
  - <u>Step 1: scale-free construction</u>
    - Error minimization
      - Setting the partial derivatives of the function $E_1\{\mathbf{v}'\}$ with respect to the free variables $\mathbf{u} = (u_{0x}, u_{0y}, \ldots, u_{mx}, u_{my})^T$ in $\mathbf{v}'$ to zero
      - By reordering v′ to put the free variables first we can write

$$\mathbf{v}'^{T} = (\mathbf{u}^{T}\ \mathbf{q}^{T})$$

Where **q** represents the constrained vertices

2D Deformation

- **As-rigid-as-possible deformation**
  - <u>Step 1: scale-free construction</u>
    - Error minimization

$$\mathbf{v}'^{\mathrm{T}} = (\mathbf{u}^{\mathrm{T}}\ \mathbf{q}^{\mathrm{T}})$$

$$\Downarrow$$

$$E_1 = \mathbf{v}'^{\mathrm{T}} \mathbf{G} \mathbf{v}' = \begin{pmatrix} \mathbf{u} \\ \mathbf{q} \end{pmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{G}_{00} & \mathbf{G}_{01} \\ \mathbf{G}_{10} & \mathbf{G}_{11} \end{bmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{q} \end{pmatrix}$$

$$\Downarrow$$

$$\frac{\partial E_1}{\partial \mathbf{u}} = (\mathbf{G}_{00} + \mathbf{G}_{00}{}^{\mathrm{T}})\mathbf{u} + (\mathbf{G}_{01} + \mathbf{G}_{10}{}^{\mathrm{T}})\mathbf{q} = \mathbf{0}$$

2D Deformation

- **As-rigid-as-possible deformation**
  - Step 1: scale-free construction
    - Error minimization

$$\frac{\partial E_1}{\partial \mathbf{u}} = (\mathbf{G}_{00} + \mathbf{G}_{00}{}^{\mathrm{T}})\mathbf{u} + (\mathbf{G}_{01} + \mathbf{G}_{10}{}^{\mathrm{T}})\mathbf{q} = \mathbf{0}$$

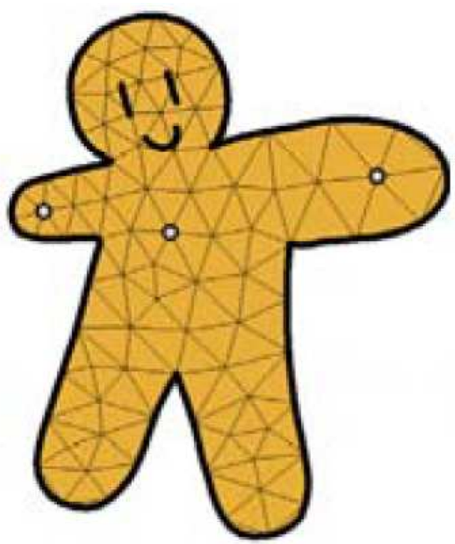
$$\Downarrow$$

$$\mathbf{G}'\mathbf{u} + \mathbf{B}\mathbf{q} = \mathbf{0}$$

    - Note
      - **G′** and **B** are fixed
      - Only **q** changes during manipulation
      - Pre-computing $\mathbf{G}'^{-1}\mathbf{B}$ at the beginning

2D Deformation

- **As-rigid-as-possible deformation**
  - Step 1: scale-free construction
    - Example results



Problem: scale and orientation not restricted

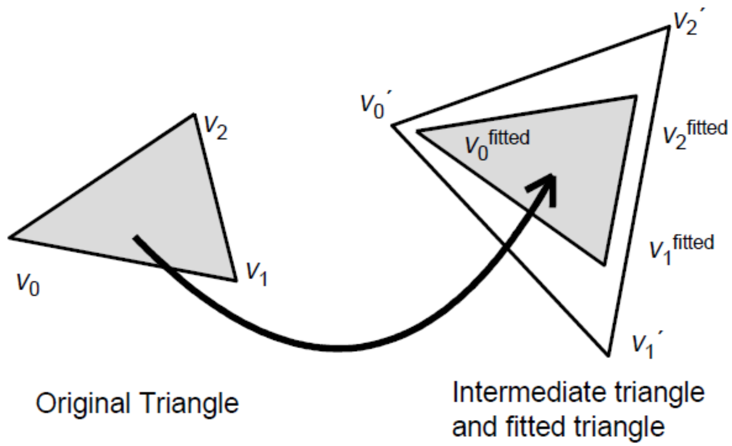
2D Deformation

- **As-rigid-as-possible deformation**
  - Step two: scale adjustment
    - Fitting the original triangle to the intermediate triangle
      - Allowing rotation and translation
      - No shearing and scaling
    - Given a triangle $\{v_0', v_1', v_2'\}$ in the intermediate result and corresponding triangle in the rest shape $\{v_0, v_1, v_2\}$
      - Find a new triangle $\{v_0^{\text{fitted}}, v_1^{\text{fitted}}, v_2^{\text{fitted}}\}$ that is congruent to $\{v_0, v_1, v_2\}$
      - Minimize the following functional

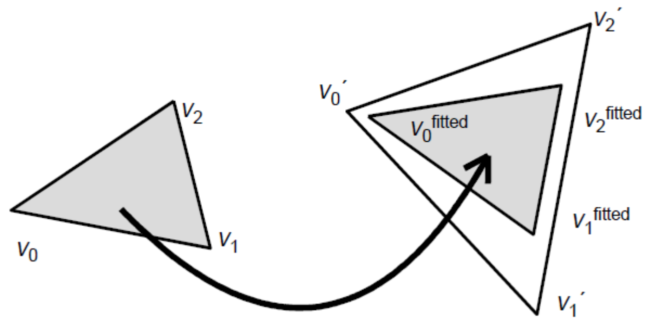
$$E_{\mathrm{f}\,\{v_0^{\text{fitted}}, v_1^{\text{fitted}}, v_2^{\text{fitted}}\}} = \sum_{i=1,2,3} \left\| v_i^{\text{fitted}} - v_i' \right\|^2$$

2D Deformation

- **As-rigid-as-possible deformation**
  - <u>Step two: scale adjustment</u>
    - Fitting the original triangle to the intermediate triangle
      - Approximate by first minimizing the error allowing uniform scaling and then adjusting the scale afterwards



2D Deformation



Original Triangle

Intermediate triangle and fitted triangle

- **As-rigid-as-possible deformation**
  - Step two: scale adjustment
    - Fitting the original triangle to the intermediate triangle
      - Using the coordinates x_{01} and y_{01}, we can express v_2^{fitted} using v_0^{fitted} and v_1^{fitted}:

$$v_2^{\text{fitted}} = v_0^{\text{fitted}} + x_{01}\overrightarrow{v_0^{\text{fitted}} v_1^{\text{fitted}}} + y_{01} R_{90} \overrightarrow{v_0^{\text{fitted}} v_1^{\text{fitted}}}$$

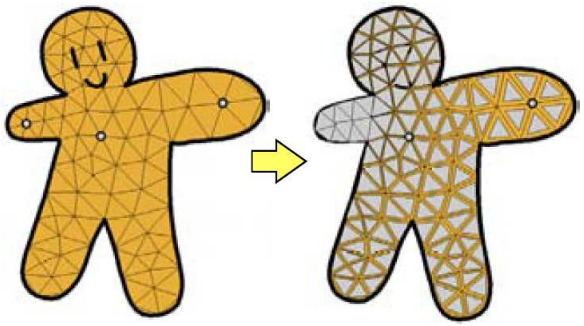
      - The fitting functional becomes
        - A function of just the coordinates of v_0 $^{\text{fitted}}$ and v_1 $^{\text{fitted}}$
        - A quadratic in the four free variables of $w=(v_{0x}^{\text{fitted}}, v_{0y}^{\text{fitted}}, v_{1x}^{\text{fitted}}, v_{1y}^{\text{fitted}})^T$
      - We minimize E_f by

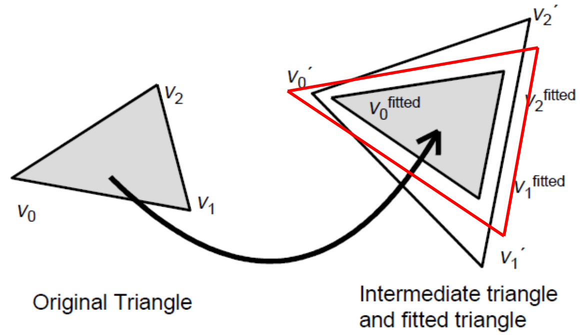
$$\frac{\partial E_{\mathrm{f}}}{\partial \mathbf{w}} = \mathbf{F}\mathbf{w} + \mathbf{C} = \mathbf{0}$$

F is fixed for a given mesh
C is defined by the result of step one

2D Deformation

- **As-rigid-as-possible deformation**
  - Step two: scale adjustment
    - Fitting the original triangle to the intermediate triangle
      - We obtain a newly fitted triangle similar to the original triangle
      - Make it congruent simply by scaling the fitted triangle by the factor of $||v_0^{fitted}-v_1^{fitted}||/||v_0-v_1||$
      - Apply this fitting operation to all triangles in the mesh
    - Fitting result



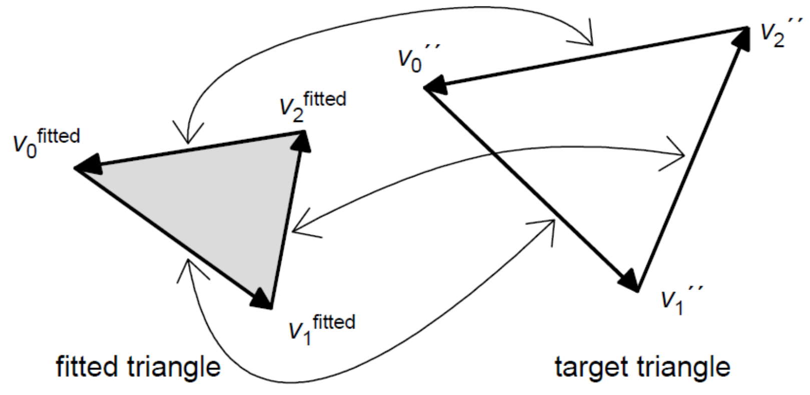


2D Deformation

- **As-rigid-as-possible deformation**
  - <u>Step two: scale adjustment</u>
    - Generating the final result using the fitted triangles
      - Assemble all the fitted triangles
      - Given the corresponding fitted triangle $\{v_0^{fitted}, v_1^{fitted}, v_2^{fitted}\}$, we define a quadratic error function by:

$$E_{2\{v_0'' v_1'', v_2''\}} = \sum_{(i,j)\in\{(0,1),(1,2),(2,0)\}} \left\| \overrightarrow{v_i'' v_j''} - \overrightarrow{v_i^{\text{fitted}} v_j^{\text{fitted}}} \right\|^2$$

      - we associate an error with each edge



2D Deformation

- **As-rigid-as-possible deformation**
  - Step two: scale adjustment
    - Generating the final result using the fitted triangles
      - The error for the entire mesh can be represented in a matrix form

$$E_{2\{v''\}} = \mathbf{v}''^{\mathrm{T}}\mathbf{H}\mathbf{v}'' + \mathbf{f}\mathbf{v}'' + c$$

      - Note
        - **H** is defined by the connectivity of the original mesh
        - **f** and c are determined by the fitted triangles
      - We minimize E_2 by setting the partial derivatives of E_2 over free vertices **u** to zero
        - Reordering **v''**, we can write $\mathbf{v}''^{\mathrm{T}} = (\mathbf{u}^{\mathrm{T}}\ \mathbf{q}^{\mathrm{T}})$

2D Deformation

- **As-rigid-as-possible deformation**
  - Step two: scale adjustment
    - Generating the final result using the fitted triangles

$$E_2 = \mathbf{v}''^{\mathrm{T}}\mathbf{H}\mathbf{v}'' + \mathbf{f}\mathbf{v}'' + c = \begin{pmatrix}\mathbf{u}\\ \mathbf{q}\end{pmatrix}^{\mathrm{T}}\begin{bmatrix}\mathbf{H}_{00} & \mathbf{H}_{01}\\ \mathbf{H}_{10} & \mathbf{H}_{11}\end{bmatrix}\begin{pmatrix}\mathbf{u}\\ \mathbf{q}\end{pmatrix} + \begin{pmatrix}\mathbf{f}_0 \mathbf{f}_1\end{pmatrix}\begin{pmatrix}\mathbf{u}\\ \mathbf{q}\end{pmatrix} + c$$

$$\Downarrow$$

$$\frac{\partial E_2}{\partial \mathbf{u}} = (\mathbf{H}_{00} + \mathbf{H}_{00}{}^{\mathrm{T}})\mathbf{u} + (\mathbf{H}_{01} + \mathbf{H}_{10}{}^{\mathrm{T}})\mathbf{q} + \mathbf{f}_0 = \mathbf{0}$$

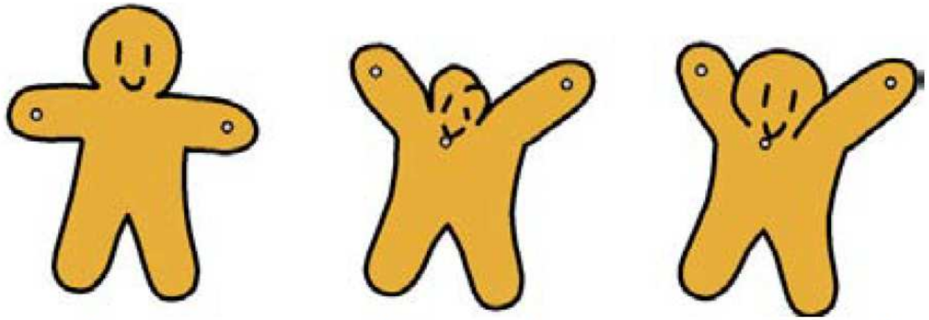
$$\Downarrow$$

$$\mathbf{H}'\mathbf{u} + \mathbf{D}\mathbf{q} + \mathbf{f}_0 = \mathbf{0}$$

H′ and **D** are fixed
q and $\mathbf{f_0}$ change during manipulation

2D Deformation

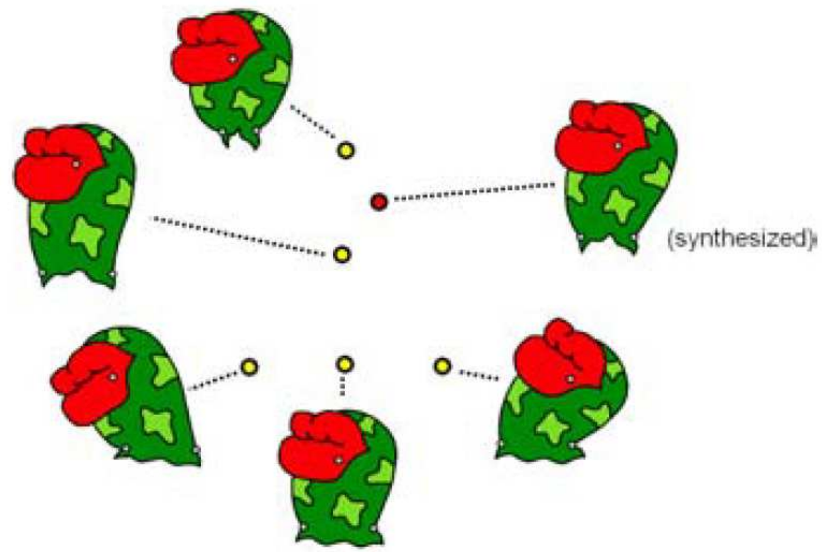
- **As-rigid-as-possible deformation**
  - Extensions
    - Weights for controlling rigidity
      - Add weights in front of energy function to control rigidity
      - Can manually designed



rest shape without weights with weights

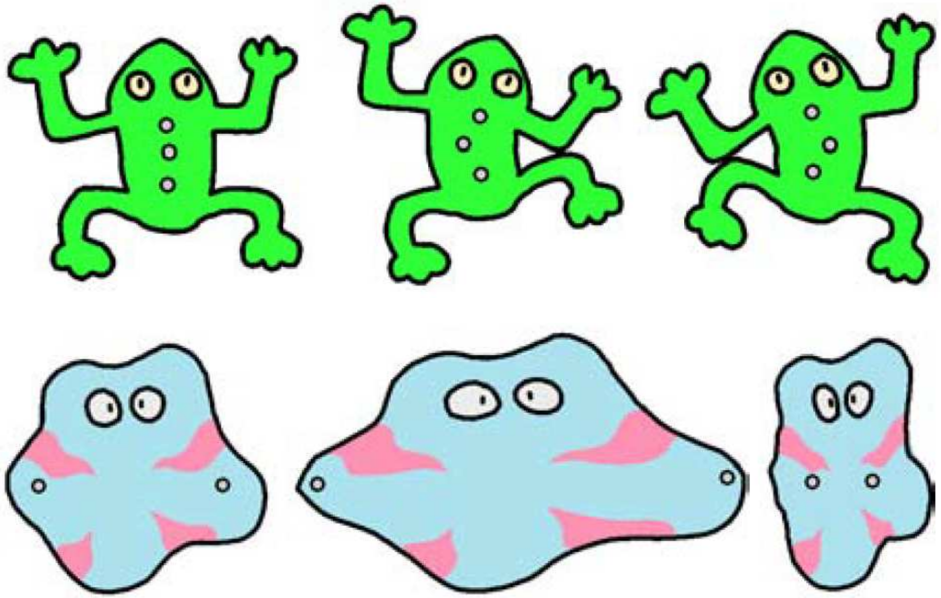
2D Deformation

- **As-rigid-as-possible deformation**
  - Animations
    - Manual manipulation to create keyframes
    - RBF to interpolate anchor points for intermediate frames



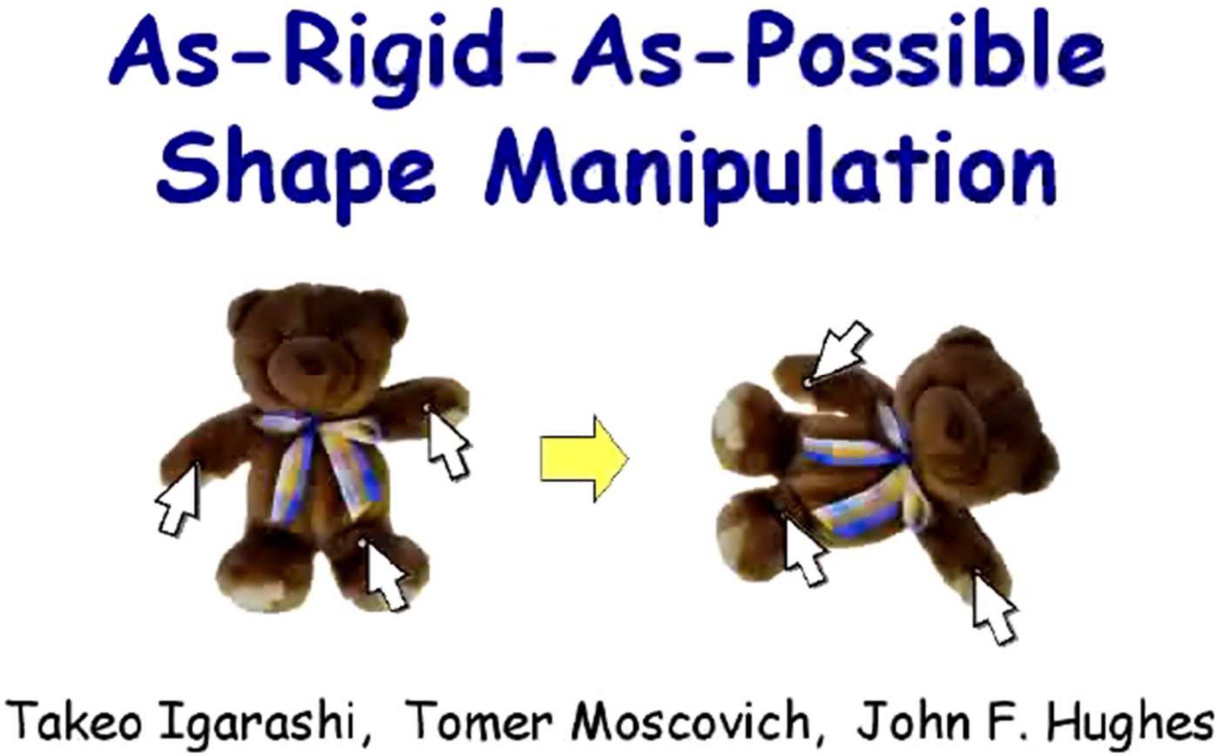
2D Deformation

- **As-rigid-as-possible deformation**
  - Different results



2D Deformation

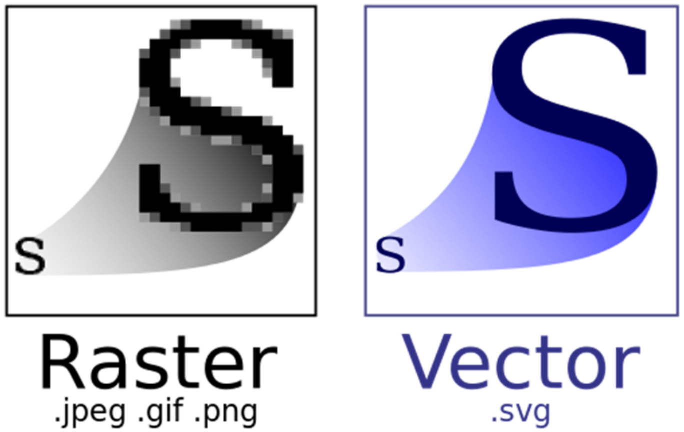
- **As-rigid-as-possible deformation**
  - Video



III. Animation by 2D Vectorization

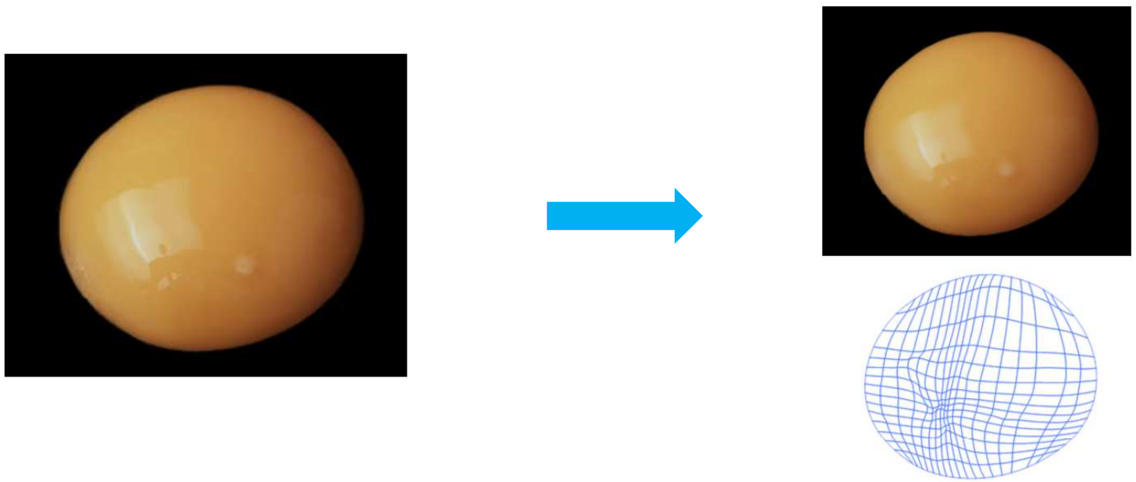
Image Vectorization

- **The process of conversion**
  - From raster graphics into vector graphics
    - Unlimited resolution
    - No aliasing artifacts
    - Smaller storage
    - Easier to edit and manipulate



Gradient Mesh

- **A powerful vector graphics representation**
  - Draw multicolored mesh objects with smooth transitions
  - Can be used to vectorize a raster image



Gradient Mesh

- **Parametric surface patch**
  - A general parametric surface representation has the form

$$S = \{(x,y,z) : x = X(u,v), y = Y(u,v), z = Z(u,v)\}$$

  - A tensor product patch is defined as

$$m(u,v) = F(u)QF^T(v)$$

    - m(u,v) is the position vector of a point (u,v)
    - F vectors consist of the basis functions
    - Q matrix is a function of the control points
    - Bezier bicubic, rational biquadratic, B-splines, etc.
      - Require control points lying outside the surface

Gradient Mesh

- **Ferguson patch**
  - Defined with control points lying on the surface
  - Suitable for image vectorization
  - Definition

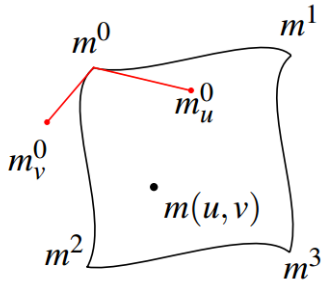
$$m(u,v) = F(u)QF^T(v) = UCQC^TV \qquad Q = \begin{bmatrix} m^0 & m^2 & m_v^0 & m_v^2 \\ m^1 & m^3 & m_v^1 & m_v^3 \\ m_u^0 & m_u^2 & m_{uv}^0 & m_{uv}^2 \\ m_u^1 & m_u^3 & m_{uv}^1 & m_{uv}^3 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ -2 & 2 & 1 & 1 \end{bmatrix},$$

$$U = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix}, \text{ and } V = \begin{bmatrix} 1 & v & v^2 & v^3 \end{bmatrix}.$$

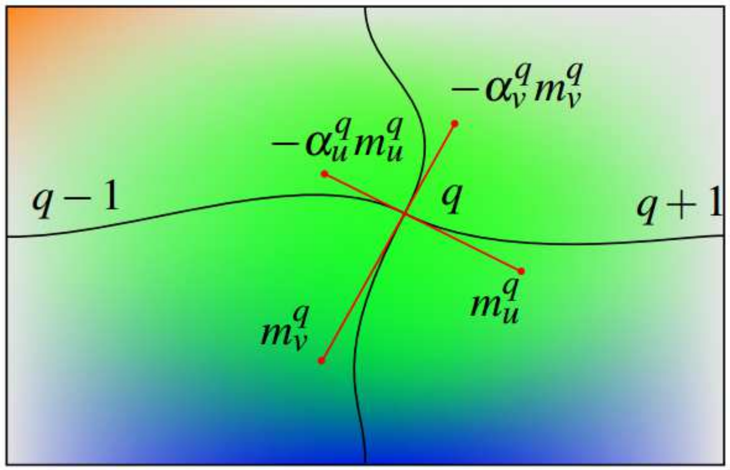
    - m_u, m_v, m_{uv} are the partial derivatives
    - In practice, the values of m_{uv} are usually set to zero

Gradient Mesh

- **Ferguson patch**
  - An example

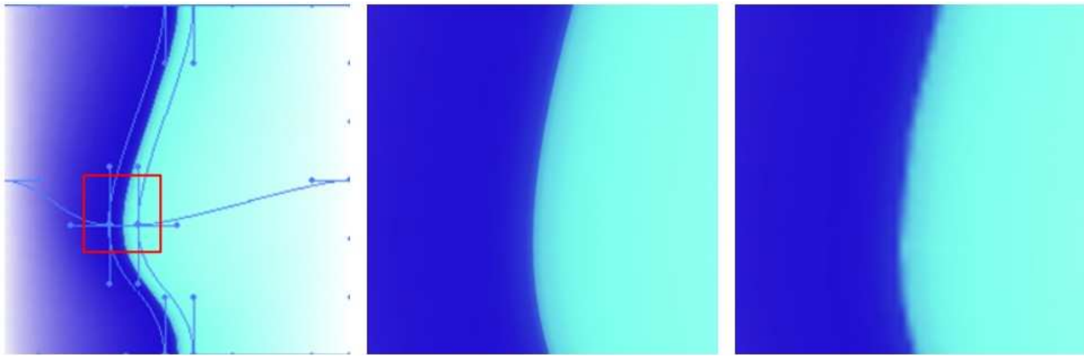


- **Gradient mesh**
  - Topologically planar rectangular Ferguson patches



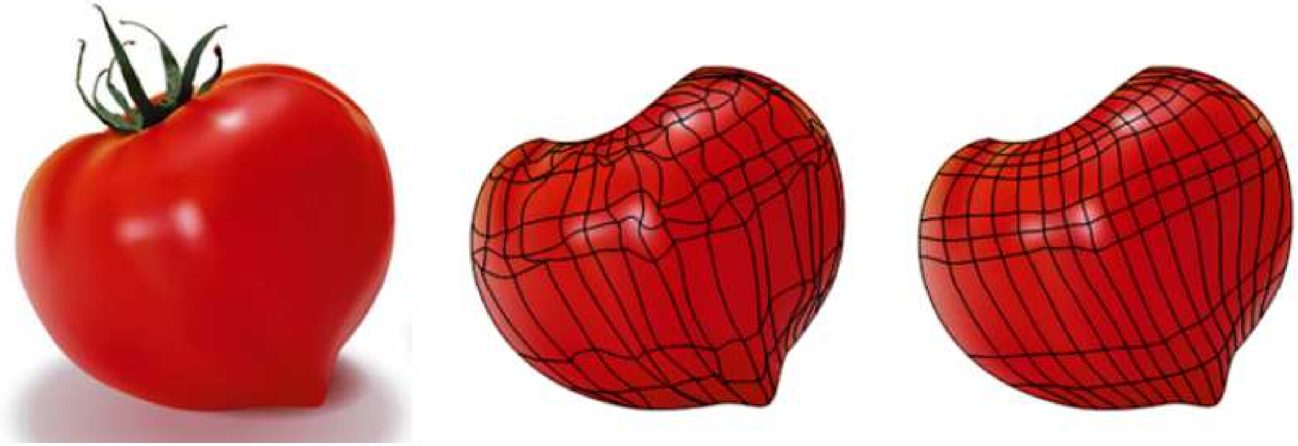
Gradient Mesh

- **Gradient mesh**
  - For boundaries, each consists of one or more cubic Bezier splines
  - For each control point q in the mesh
    - Position, derivatives, and RGB color can be edited
  - Scalability
    - Gradient meshes can be scaled with fewer artifacts



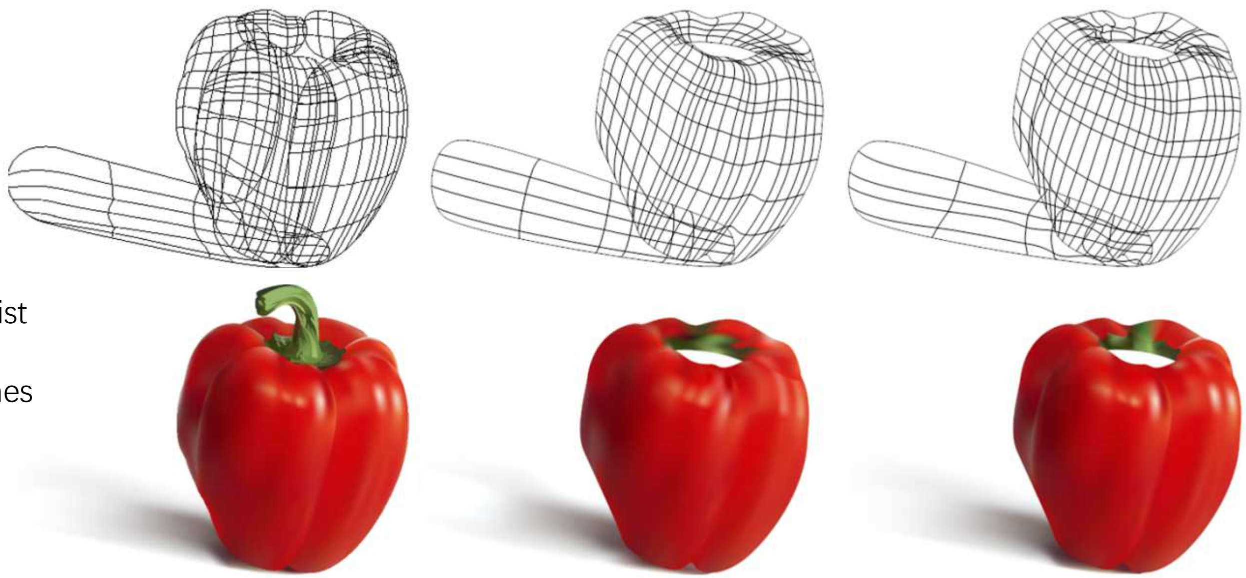
Gradient Mesh

- **Image reconstruction**
  - Fitting optimized gradient mesh on image
    - Optimization without and with mesh smoothness



Gradient Mesh

- **Image vectorization results**



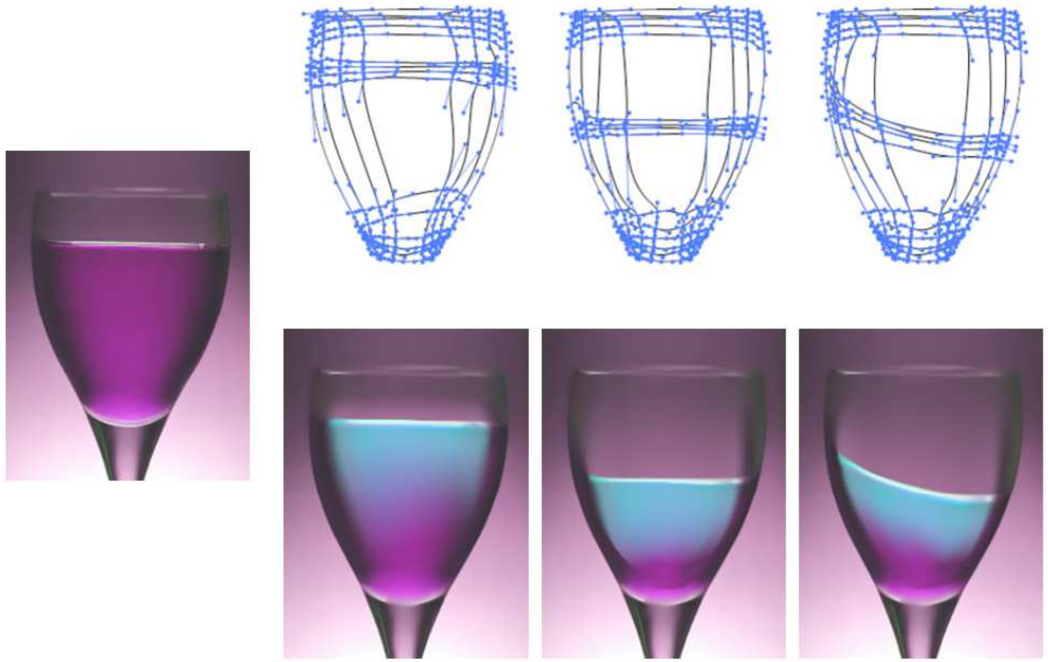
Left: gradient meshes by an artist
Middle: initial gradient meshes
Right: optimized gradient meshes

Gradient Mesh

- **Image vectorization results**

Gradient Mesh

- **Animation**
  - Editing gradient meshes to create keyframes
  - Interpolate the gradient mesh to create animations

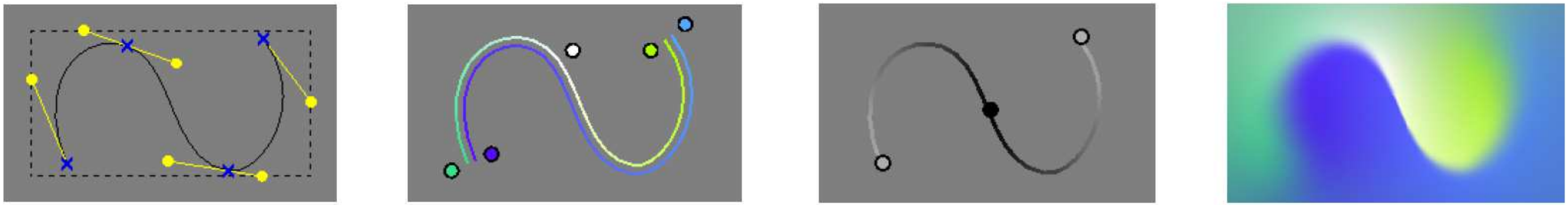


Diffusion Curve

- **How to create vector graphics easily?**
  - A diffusion curve partitions the space through which it is drawn
    - Define different colors on either side
    - Support a variety of operations
      - Geometry-based editing, keyframe animation, and ready stylization

Diffusion Curve

- **Data structure**
  - Graphical illustration



Diffusion Curve

- **Rendering smooth gradients from diffusion curves**
  - **Color sources**
    - Distance the colors a little bit
    - Gradient constraint
      - Expressed as a gradient field **w**
      - Zero everywhere except on the curve, where it is equal to the color derivative across the curve

$$w_{x,y} = (cl - cr)N_{x,y}$$

Diffusion Curve

- **Rendering smooth gradients from diffusion curves**
  - **Diffusion**
    - Given the color sources and gradient fields
    - Compute the color image I
      - From the steady state diffusion of the color sources
      - Subject to the gradient constraints
      - The solution to a Poisson equation

$$
\begin{aligned}
\Delta I &= \operatorname{div} \mathbf{w} \\
I(x,y) &= C(x,y) \text{ if pixel } (x,y) \text{ stores a color value}
\end{aligned}
$$

where Δ and div are the Laplace and divergence operators.

Diffusion Curve

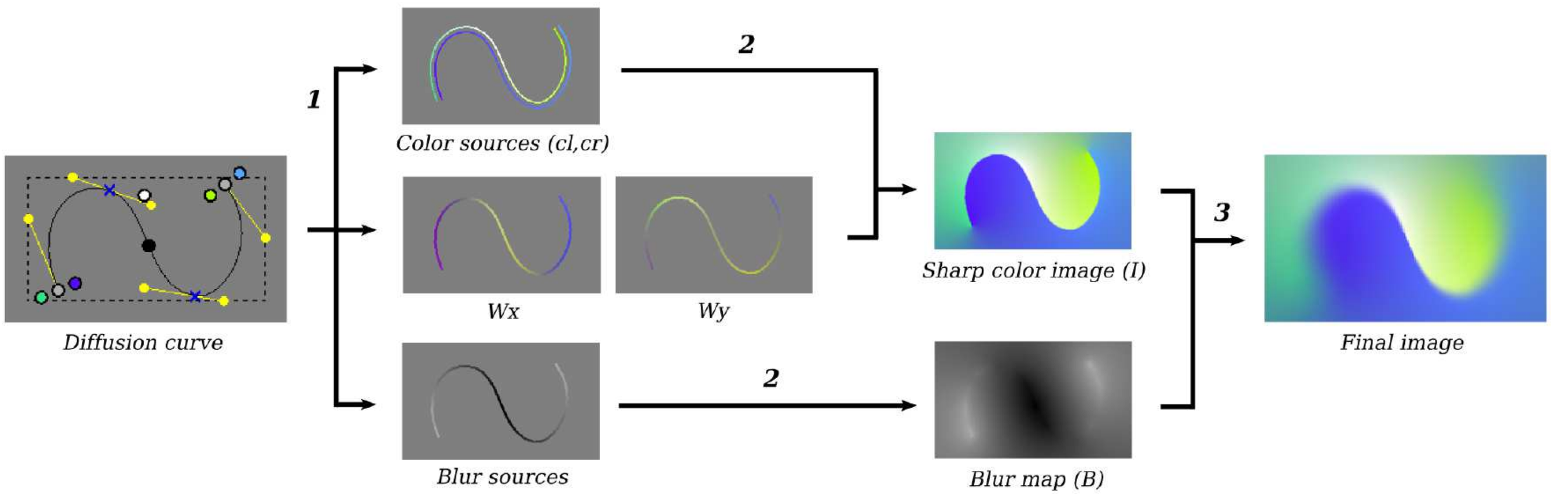
- **Rendering smooth gradients from diffusion curves**
  - **Reblurring**
    - Color diffusion according to blur values stored along each curve
    - Blur values are stored only on curves
      - Diffuse the blur values over the image by solving Laplacian problem

$$
\begin{aligned}
\Delta B &= 0 \\
B(x,y) &= \sigma(x,y) \text{ if pixel } (x,y) \text{ is on a curve}
\end{aligned}
$$

      - With blur field, apply a spatially varying blur on the color image by image filtering

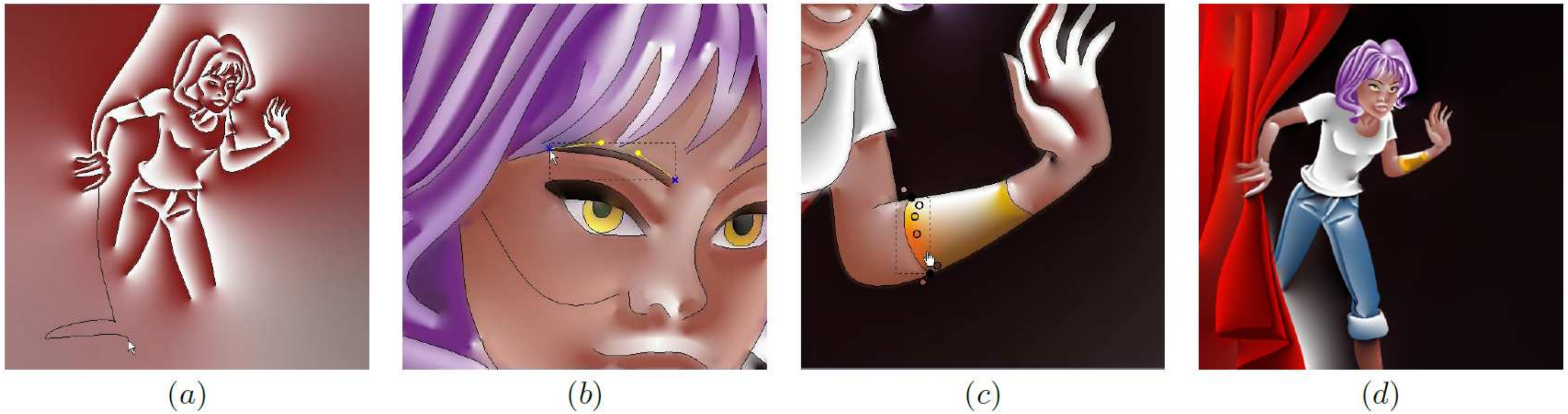
Diffusion Curve

- **Rendering smooth gradients from diffusion curves**
  - The whole process



Creating Diffusion Curves

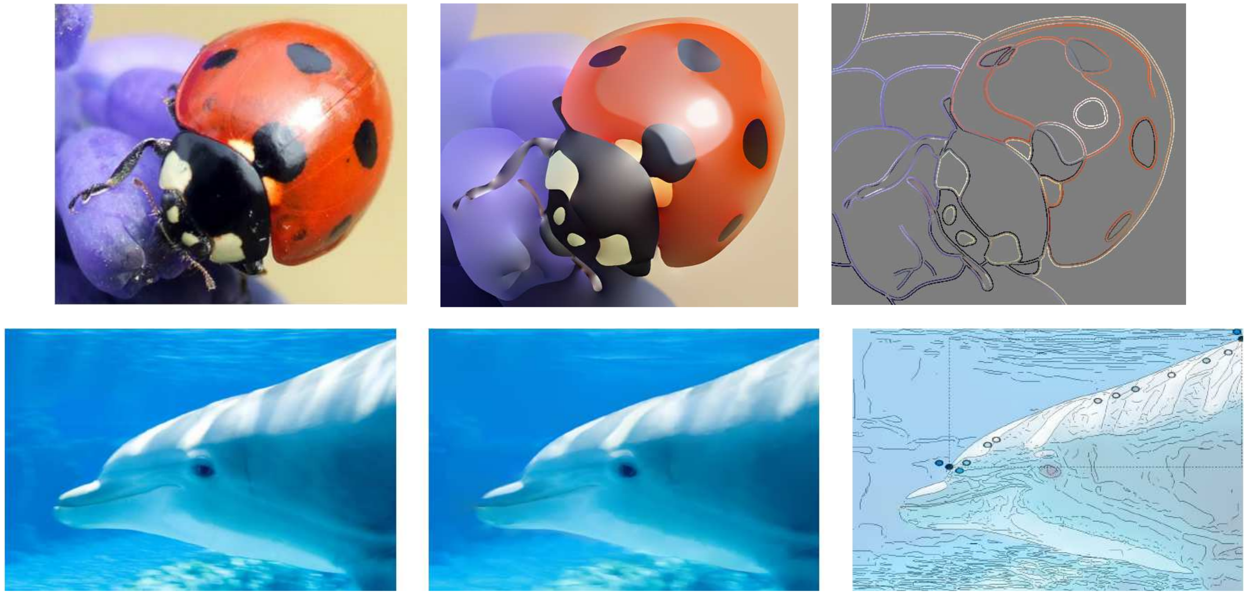
- **Manual creation**



(a) sketching the curves
(b) adjusting the curve’s position
(c) setting colors and blur along the diffusion curve
(d) the final result.

Creating Diffusion Curves

- **Tracing an image**



Creating Diffusion Curves

- **Keyframe-based animation**
  - Edit keyframes based on diffusion curves
  - Create intermediate frames by control parameter interpolation

Next Lecture: Non-Physically-Based Animation II