Image Warping and Morphing

CS5245 Vision & Graphics for Special Effects

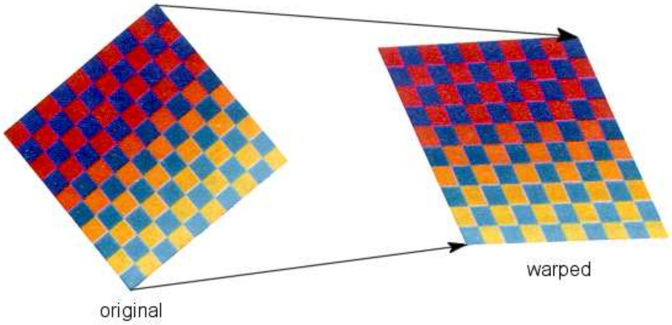
Leow Wee Kheng

Department of Computer Science
School of Computing
National University of Singapore

Image Warping

Objective: Change appearance of image by performing geometric transformation, i.e., change the position of a point in the image to a new position.

Example:

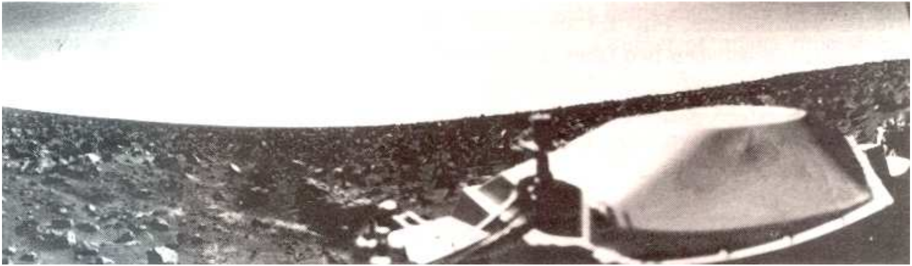


Note:

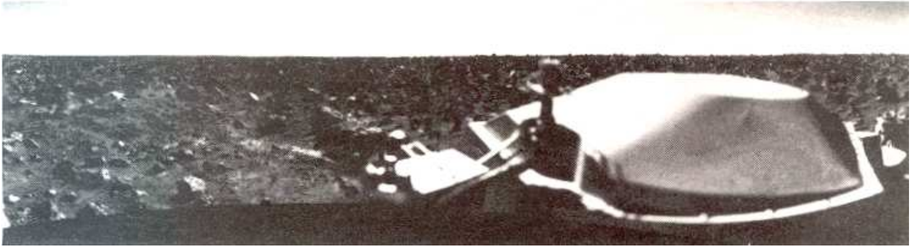
- Only the positions of the points are changed.
- The colors (or intensities) of the corresponding points in the two images are the same.

Earliest work on image warping comes from remote sensing.

- Capture images at various positions and/or angles.
- Then, stitch them together, i.e., image mosaicking (see CS4243 Computer Vision and Pattern Recognition).
- Camera lens often has distortion. So, need to undistort.



(a) Viking Lander 2 image distorted due to downward tilt.



(b) Undistorted image.

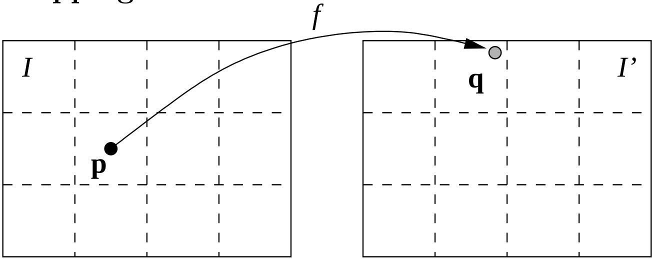
In image warping, want to purposely distort images.

Image Warping

> *Given source image I and the correspondence between the original position $\mathbf{p}_i = (u_i, v_i)^T$ of a point in I and its desired new position $\mathbf{q}_i = (x_i, y_i)^T$, $i = 1, \ldots, n$, generate a warped image I' such that $I'(\mathbf{q}_i) = I(\mathbf{p}_i)$ for each point i. I and I' represent the intensities or colors of the images.*

The idea of correspondence is defined by a mapping function f.

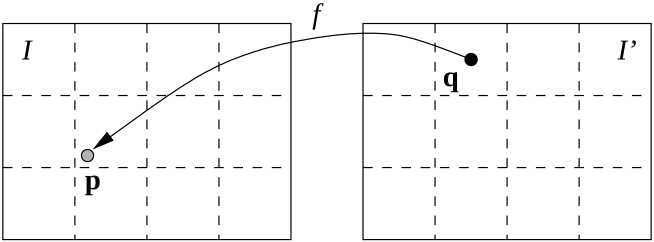
Forward Mapping



$$\mathbf{q} = f(\mathbf{p}) \tag{1}$$

- f maps a point $\mathbf{p}$ in I into a point $\mathbf{q}$ in I'.
- $I'(\mathbf{q}) = I(\mathbf{p})$.
- Problem: $\mathbf{q} = (x, y)$ has real-valued coordinates.
- If round off (x, y) to integer coordinates, will have error and misalignment.

Inverse or Backward Mapping



$$\mathbf{p} = f(\mathbf{q}) \tag{2}$$

- f maps an *integer*-coordinate point $\mathbf{q}$ in I' into a *real*-coordinate point $\mathbf{p}$ in I.
- Use the colors of neighboring integer-coordinate points in I to estimate $I(\mathbf{p})$: bilinear interpolation (see CS4243 Computer Vision and Pattern Recognition).
- Then, $I'(\mathbf{q}) = I(\mathbf{p})$.
- Advantage: No round-off error.

Notes:

- In practice, correspondence is given for only a small number of points.
- Need to derive the correspondence for the other points.
- General idea: determine f using the known corresponding points.
- Geometric transformation is most conveniently expressed as a matrix operation:

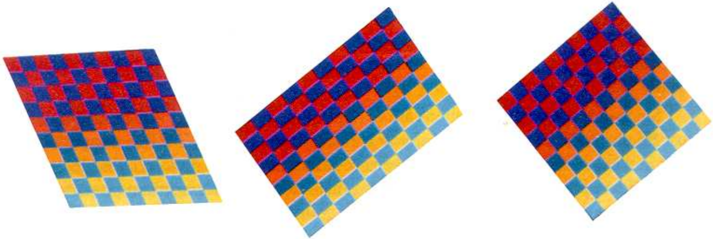
$$\mathbf{p} = \mathbf{T}\,\mathbf{q} \tag{3}$$

where $\mathbf{T}$ is the transformation matrix.

Affine Transformation

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4}$$

- **p** and **q** are homogeneous coordinates.
- Affine transformation is a linear transformation.



- How many corresponding pairs needed to solve for the parameters?

Method 1

From Eq. 4,

$$\begin{aligned} u_i &= a_{11}\,x_i + a_{12}\,y_i + a_{13} \\ v_i &= a_{21}\,x_i + a_{22}\,y_i + a_{23} \end{aligned} \tag{5}$$

for $i = 1, \ldots, n$.

Now, we have two sets of linear equations of the form

$$\mathbf{M}\,\mathbf{a} = \mathbf{b} \tag{6}$$

First set:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \tag{7}$$

Second set:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \tag{8}$$

- Can compute best fitting a_{kl} for each set independently using standard methods.

Method 2

Compute the sum-squared error E as

$$E = \sum_{i=1}^{n} \|\mathbf{p}_i - \mathbf{T}\,\mathbf{q}_i\|^2 \tag{9}$$

If E is small, then the fit is good.

So, we want to find the best fitting $\mathbf{T}$ that minimizes E.

So, do the usual thing: $\partial E / \partial\,\mathbf{T} = 0$

$$\begin{gathered} \frac{\partial E}{\partial\,\mathbf{T}} = -2 \sum_i (\mathbf{p}_i - \mathbf{T}\,\mathbf{q}_i)\,\mathbf{q}_i^T = 0 \\ \sum_i \mathbf{T}\,\mathbf{q}_i\,\mathbf{q}_i^T = \sum_i \mathbf{p}_i\,\mathbf{q}_i^T \end{gathered} \tag{10}$$

That is,

$$\sum_i \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} = \sum_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} \tag{11}$$

Rearranging terms yield

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix} \mathbf{a} = \mathbf{b} \tag{12}$$

where

$$
\begin{aligned}
\mathbf{M} &= \begin{bmatrix} \sum_i x_i^2 & \sum_i x_i y_i & \sum_i x_i \\ \sum_i x_i y_i & \sum_i y_i^2 & \sum_i y_i \\ \sum_i x_i & \sum_i y_i & \sum_i 1 \end{bmatrix} \\
\mathbf{0} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
\mathbf{a} &= \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} \end{bmatrix}^T \\
\mathbf{b} &= \begin{bmatrix} \sum_i u_i x_i & \sum_i u_i y_i & \sum_i u_i & \sum_i v_i x_i & \sum_i v_i y_i & \sum_i v_i \end{bmatrix}^T
\end{aligned}
$$

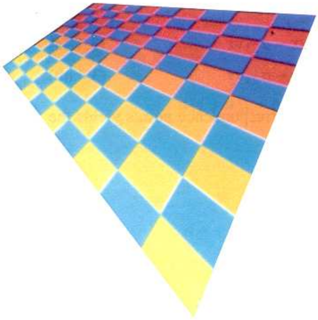
Now, we again have a linear system of equations to solve for a_{ij}.

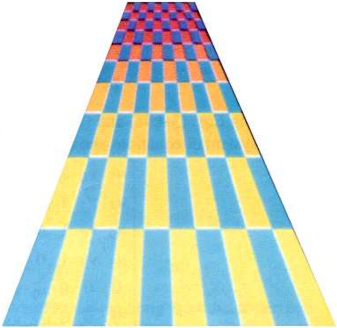
Perspective Transformation

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{13}$$

- Eq. 13 is a set of linear equations.
- But, perspective transformation is a nonlinear transformation.
- Linear equations describe nonlinear transformation.
  Seems paradoxical, but there is nothing wrong. Why?

Examples:





Polynomial Transformation

In general, any polynomial transformation can be expressed as follows:

$$\begin{aligned} u &= \sum_k \sum_l a_{kl}\, x^k y^l \\ v &= \sum_k \sum_l b_{kl}\, x^k y^l \end{aligned} \tag{14}$$

Example: 2nd-order polynomial transformation.

$$\begin{aligned} u &= a_{20}x^2 + a_{02}y^2 + a_{11}xy + a_{10}x + a_{01}y + a_{00} \\ v &= b_{20}x^2 + b_{02}y^2 + b_{11}xy + b_{10}x + b_{01}y + b_{00} \end{aligned} \tag{15}$$

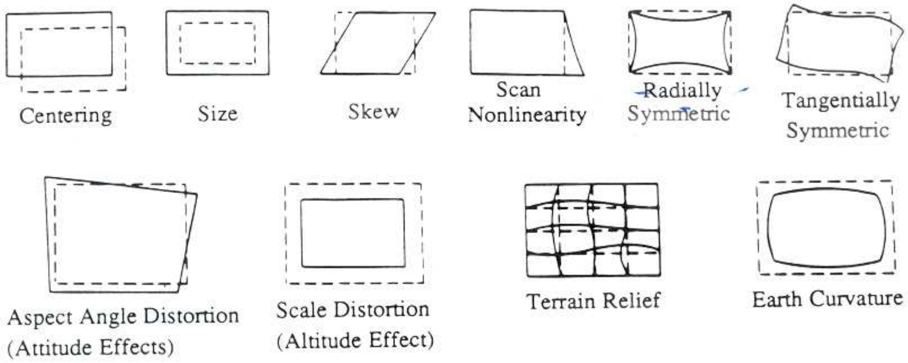
In matrix form, we have

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_{20} & a_{02} & a_{11} & a_{10} & a_{01} & a_{00} \\ b_{20} & b_{02} & b_{11} & b_{10} & b_{01} & b_{00} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \\ x \\ y \\ 1 \end{bmatrix} \tag{16}$$

If $a_{20} = a_{02} = a_{11} = b_{20} = b_{02} = b_{11} = 0$, then it becomes an affine transformation.

Again, given a set of corresponding points $\mathbf{p}_i$ and $\mathbf{q}_i$, can form a system of linear equations to solve for the a_{kl} and b_{kl}. (Exercise)

Example polynomial transformations due to lens distortion:



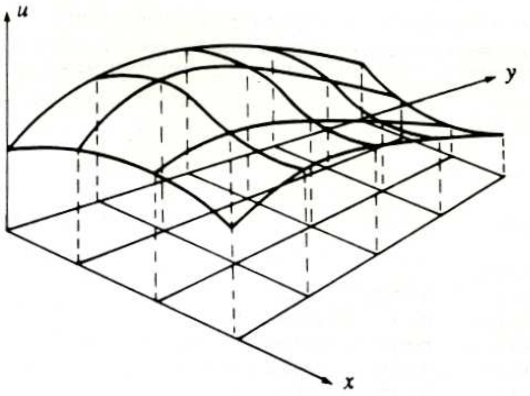
Global Transformation

The methods described in the previous sections all perform global transformation: a single function relates all the points in the whole image.

Solving for the transform parameters can be regarded as a surface fitting problem:

> *Given a set of corresponding points* $\mathbf{p}_i = (u_i, v_i)$ *and* $\mathbf{q}_i = (x_i, y_i)$, $i = 1, \ldots, n$, *determine the best fitting surface* $U(x, y)$ *that passes through the points* (x_i, y_i, u_i), *and the best fitting surface* $V(x, y)$ *that passes through the points* (x_i, y_i, v_i).

Example surface:



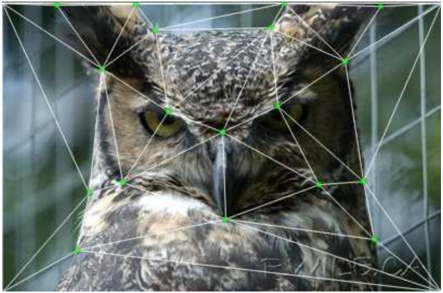
Local Transformation

- Global transformation imposes a single mapping function on the whole image.
- It is not convenient for describing local distortions that differ at different locations.
- Local transformation applies a different mapping function to a different part of the image.

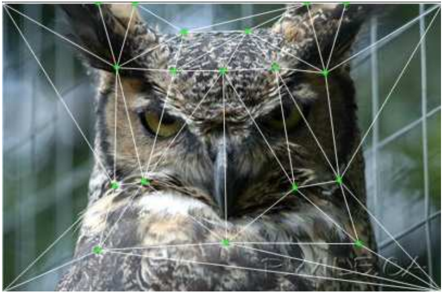
Basic Ideas:

- First, perform triangulation to partition the image into triangular regions based on the control points $\mathbf{p}_i$ and $\mathbf{q}_i$ (see CS4235 Computational Geometry, CS5237 Computational Geometry and Applications).
- Estimate partial derivatives of U (and similarly of V) with respect to x and y at each control point. This is required only if the local surface patches are to be joined smoothly.
- For each triangular region, fit a smooth surface (low-order bivariate polynomial) through the vertices satisfying the constraints imposed by the partial derivatives.
- For each point (x, y), determine its enclosing triangle and compute the corresponding (u, v) using the fitted surface parameters.

Sample triangulation of an image:

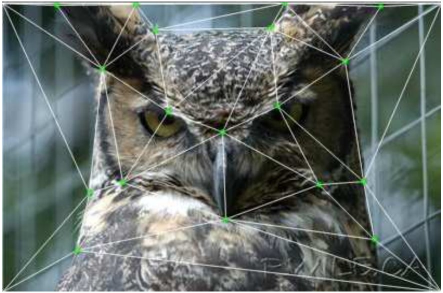


(a) Initial control points.



(b) Displaced control points.

Image warping example:



(c) Initial image.

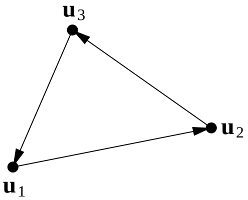
(d) Warped image.

Linear Triangular Patch

Fit a plane in each triangular region.

Given three vertices $\mathbf{u}_i = (x_i, y_i, u_i)$, $i = 1, 2, 3$, the plane that passes the vertices is given by the equation:

$$[(\mathbf{u}_2 - \mathbf{u}_1) \times (\mathbf{u}_3 - \mathbf{u}_2)] \cdot (\mathbf{u}_1 - \mathbf{u}_3) = 0 \tag{17}$$



- For a more detailed equation, refer to [2], p. 78.
- Piecewise linear mapping functions do not provide a smooth transition across patches.

Cubic Triangular Patch

Fit a cubic surface in each triangular patch.

Many algorithms using N-degree polynomials, $N = 2$ to 5 [2].

The following is a bivariate cubic patch:

$$\begin{aligned} U(x,y) = a_1 + a_2 x + a_3 y + \\ a_4 x^2 + a_5 xy + a_6 y^2 + a_7 x^3 + a_8 x^2 y + a_9 xy^2 + a_{10} y^3 \end{aligned} \tag{18}$$

Solve equation for parameters a_k using the following constraints:

- Coordinates of the 3 vertices (3 constraints).
- Partial derivatives with respect to x and y at the three vertices (6 constraints).
- Partial derivatives of two neighboring patches are the same in the direction normal to the common edge (3 constraints).
- Total 12 constraints, enough to solve for the 10 parameters.

Image Morphing

Given two images I and J, generate a sequence of images $M(t)$, $0 \leq t \leq 1$ that changes smoothly from I to J.

Basic Ideas:

- Select the corresponding points $\mathbf{p}_i$ in I and $\mathbf{q}_i$ in J.
- The corresponding point $\mathbf{r}_i(t)$ in $M(t)$ lies in between $\mathbf{p}_i$ and $\mathbf{q}_i$, e.g.,

$$\mathbf{r}_i(t) = (1 - t)\,\mathbf{p}_i + t\,\mathbf{q}_i \tag{19}$$

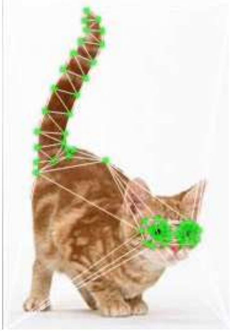
- Compute mapping functions between I and $M(t)$ and between J and $M(t)$.
- Use the mapping functions to warp I to $I(t)$ and J to $J(t)$.
- Blend $I(t)$ and $J(t)$:

$$M(t) = (1 - t)\,I(t) + t\,J(t) \tag{20}$$

- Repeat for different values of t from 0 to 1.
- When the sequence is played, $\mathbf{r}_i(t)$ moves from $\mathbf{p}_i$ to $\mathbf{q}_i$, and $M(t)$ changes from I to J.

For more advanced methods, refer to [1].

If J is a warped version of I, then can do 2D animation.



(a) Original image.

(b) Displaced control points.

(c) Warped image.

References

- S.-Y. Lee and S. Y. Shin.
  Warp generation and transition control in image morphing.
  In *Interactive Computer Animation*. Prentice Hall, 1996.
- G. Wolberg.
  Digital Image Warping.
  IEEE Computer Society Press, 1990.