BLG 453E

Week 4
Geometric/Coordinate Transforms II
Image Morphing

Dr. Yusuf H. Sahin

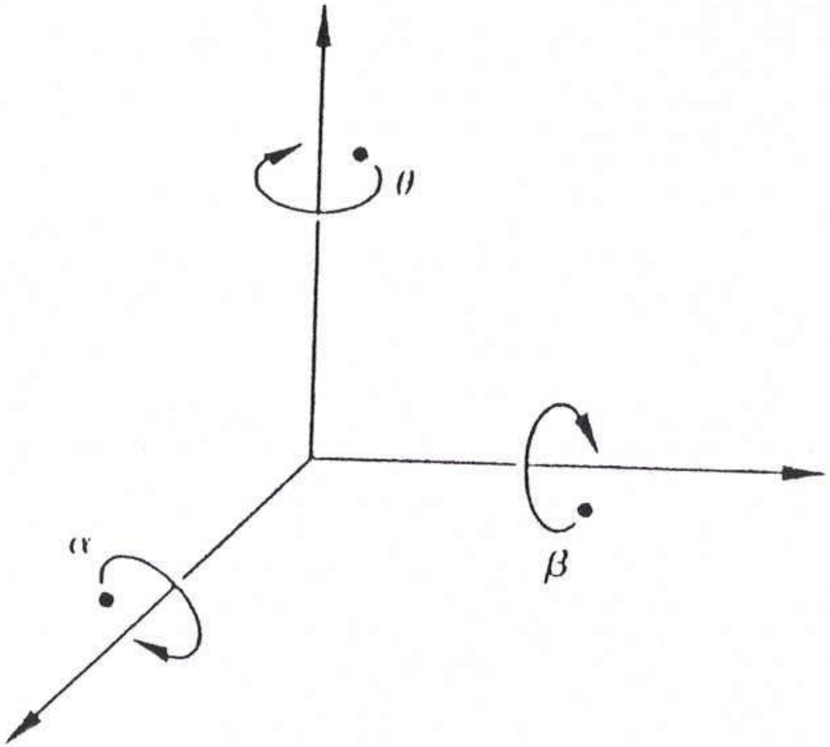
3D Rotations

- Rotation of a 3D point about each of the coordinate axes. Angles are measured clockwise when looking along the rotation axis toward the origin.

$$R_1(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}$$

$$R_2(\beta) = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix}$$

$$R_3(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



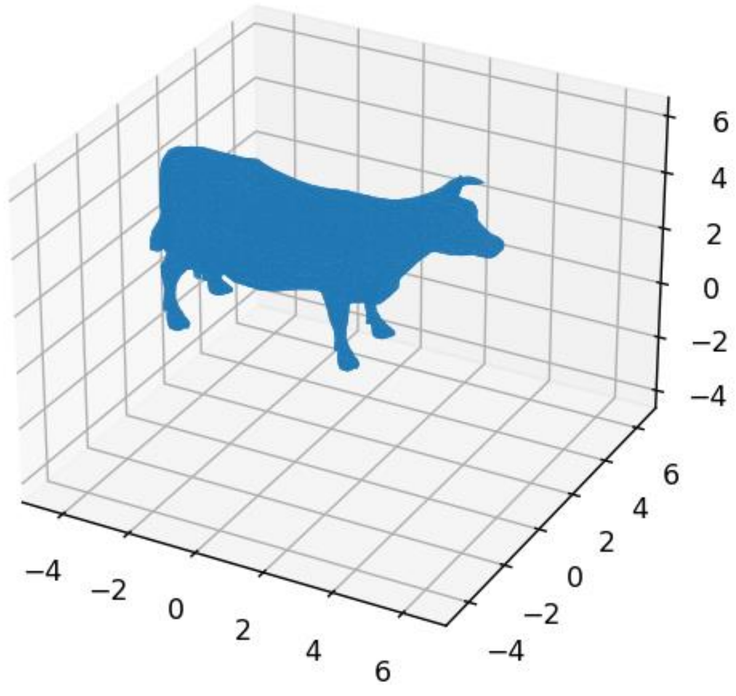
Standard Triangle Language

- A formal definition for a 3D mesh
  - A collection of triangles.

```
solid Mesh
facet
outer loop
vertex 0.0666225 -0.00713973 -0.0520612
vertex 0.0695272 -0.00912108 -0.0509354
vertex 0.0659653 -0.00814601 -0.052367
endloop
endfacet
facet
outer loop
vertex 0.0762163 -0.00201969 -0.0587023
vertex 0.0769302 -0.00441556 -0.0564184
vertex 0.0760299 -0.00791856 -0.0610091
endloop
endfacet
```

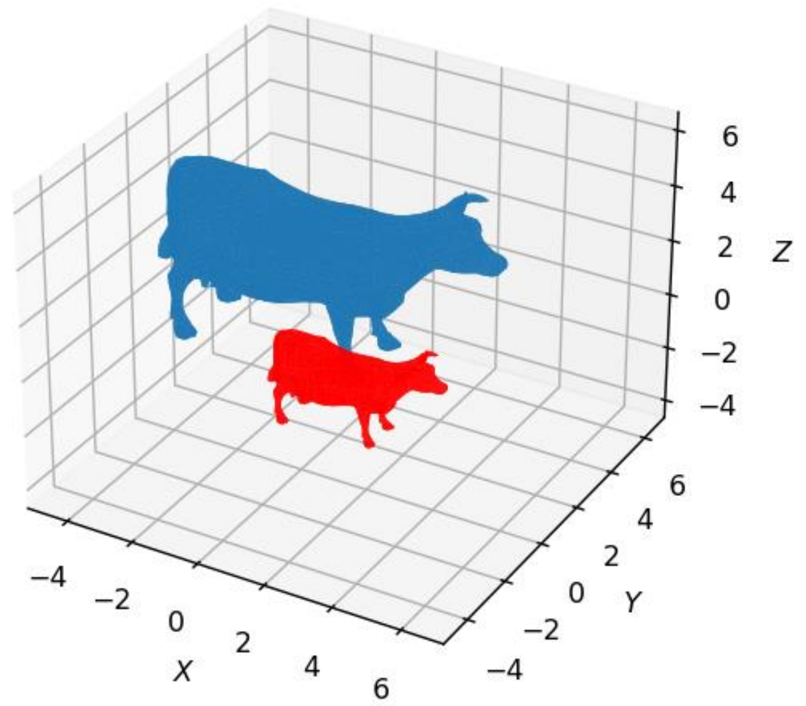
3D Meshes in Python

```
from stl import mesh
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
import numpy as np
import copy

figure = plt.figure()
axes = figure.add_subplot(projection='3d')
mesh_cow = mesh.Mesh.from_file('cow.stl')
print(mesh_cow.points.shape)
#(5804, 9) Each triangular face of the cow in column view
print(mesh_cow.vectors.shape)
#(5804, 3, 3) Each triangular face of the cow in 3x3 view. Each row represents a vertex.
axes.add_collection3d(mplot3d.art3d.Poly3DCollection(mesh_cow.vectors))
#Add the 3D faces to the created matplotlib axes
min = np.min(mesh_cow.vectors.reshape(-1))
max = np.max(mesh_cow.vectors.reshape(-1))
#Find minimum and maximum units to place the cow in a cubular grid.
axes.auto_scale_xyz([min, max], [min, max], [min, max])
plt.show()
```

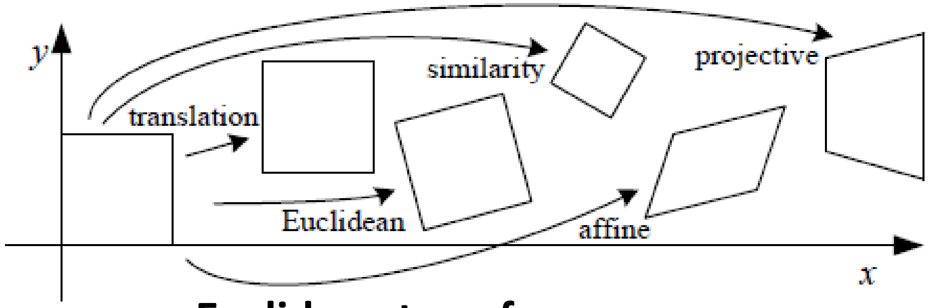


3D Meshes in Python

```python
# Add the Calf (transformed cow)
calf = mesh.Mesh.from_file('cow.stl')

# Homogeneous Transformation matrix for scaling and translation
H = np.array([
    [1/2, 0, 0, 3],     # Scale down by 1/2 in X and translate by 3
    [0, 1/2, 0, -3],    # Scale down by 1/2 in Y and translate by -3
    [0, 0, 1/2, 0],     # Scale down by 1/2 in Z
    [0, 0, 0, 1]        # Homogeneous transformation row
])

# Pad the vertices of the calf with 1s to apply the homogeneous transformation
one_padded = np.concatenate((calf.vectors, np.ones((calf.vectors.shape[0], 3, 1))), axis=-1)

# Apply the transformation using standard matrix multiplication
transformed_vectors = np.zeros_like(one_padded)
for i in range(one_padded.shape[0]):  # Iterate over each triangle
    for j in range(one_padded.shape[1]):  # Iterate over each vertex
        transformed_vectors[i, j, :] = np.dot(H, one_padded[i, j, :])

# Back to inhomogeneous coordinates (dropping the extra dimension)
calf.vectors = transformed_vectors[:, :, :3]

# Add the transformed calf to the plot
axes.add_collection(mplot3d.art3d.Poly3DCollection(calf.vectors, facecolors='r'))

# Set axis limits and labels
min_val = np.min(mesh_cow.vectors.reshape(-1))
max_val = np.max(mesh_cow.vectors.reshape(-1))
axes.auto_scale_xyz([min_val, max_val], [min_val, max_val], [min_val, max_val])
axes.set_xlabel('$X$')
axes.set_ylabel('$Y$')
axes.set_zlabel('$Z$')
plt.show()
```



The set of 2D Transforms



Translation

$$\bar{x}' = \begin{bmatrix} I & t \\ 0^T & 1 \end{bmatrix} \bar{x}$$

2 DOF

Euclidean transform

$$x' = \begin{bmatrix} R & t \end{bmatrix} \bar{x} \qquad RR^T = I$$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad |R| = 1$$

3 DOF

Similarity transform

$$x' = \begin{bmatrix} sR & t \end{bmatrix} \bar{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{x}$$

4 DOF

Affine transform

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

6 DOF

Projective transform

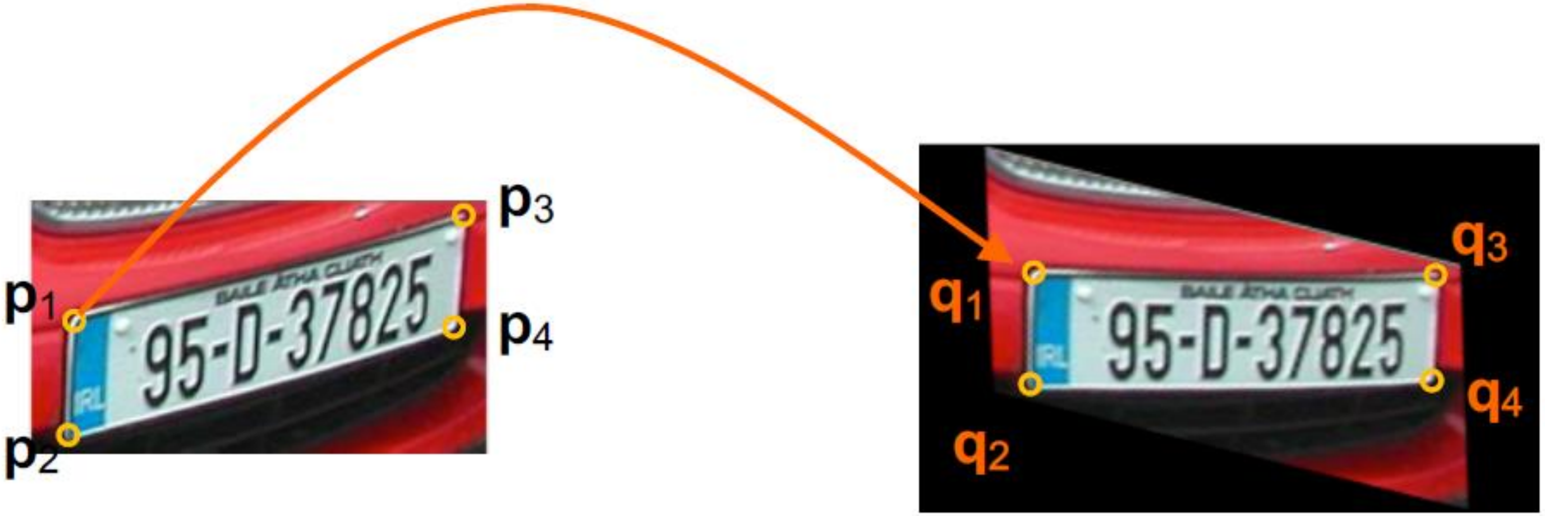
$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix}$$

8 DOF

Estimation of Affine Transform through correspondences

- When the affine transformation is not known in advance, we can estimate it.

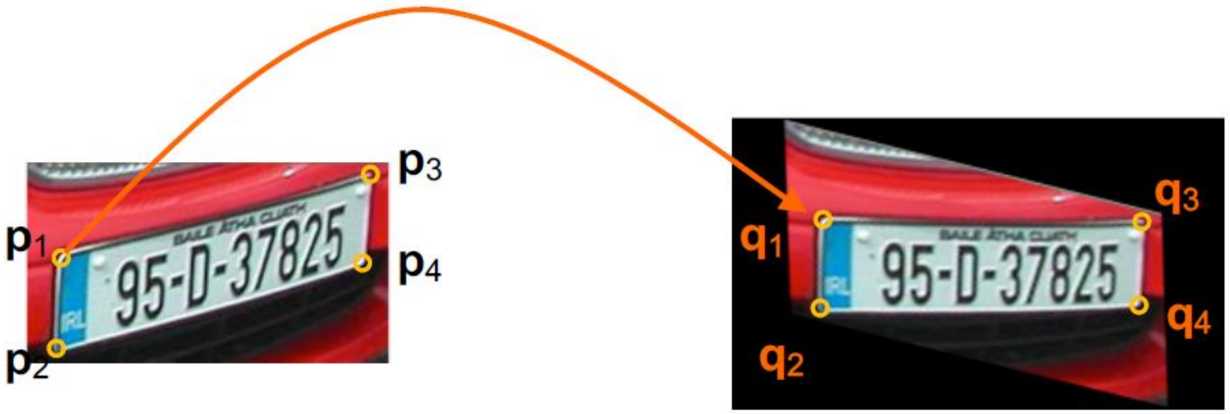
- For example, we could say p_1 corresponds to q_1. What we would like to do is estimate the affine transformation that best aligns the correspondences.



- **Task:** Determine the six coefficients in the *A* matrix.

Estimation of Affine Transform through correspondences

- This can be achieved by finding at least **three** *correspondences*, or matching points.



$$\begin{aligned}
\boldsymbol{p}_1 &= [18, 47]^T & \boldsymbol{q}_1 &= [48, 50]^T \\
\boldsymbol{p}_2 &= [15, 100]^T & \boldsymbol{q}_2 &= [48, 100]^T \\
\boldsymbol{p}_3 &= [178, 6]^T & \boldsymbol{q}_3 &= [212, 50]^T \\
\boldsymbol{p}_4 &= [173, 53]^T & \boldsymbol{q}_4 &= [212, 100]^T
\end{aligned}$$

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

Estimation through correspondences

- Noting that

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + a_{13} \\ a_{21}x + a_{22}y + a_{23} \\ 1 \end{bmatrix}$$

- If we have three correspondences we can write

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix}$$

$$q = Ma$$

$$a = M^{-1}q$$

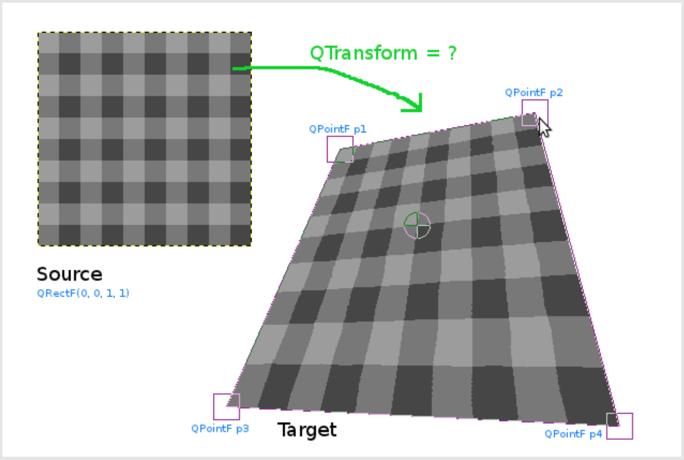
*Use pseudo-inverse!

Projective transformation

- Images normally acquired by photographic cameras are formed by perspective projection. If we view a planar surface not parallel to the image plane, then an affine transformation will not map the shape to a rectangle.
- Instead, we must use a *projective* transformation of the form

$$\begin{bmatrix} x'w \\ y'w \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11}x + p_{12}y + p_{13} \\ p_{21}x + p_{22}y + p_{23} \\ p_{31}x + p_{32}y + 1 \end{bmatrix}$$

To estimate a projective transformation, at least **four** 2D correspondences are needed (due to the eight unknowns).



Projective transformation

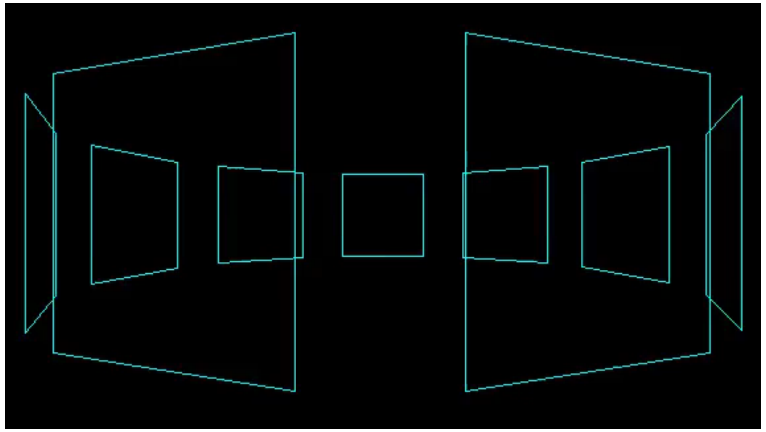
- Example: Find the best fitting warp to transform the game area to a given rectangle:

original

affine

projective

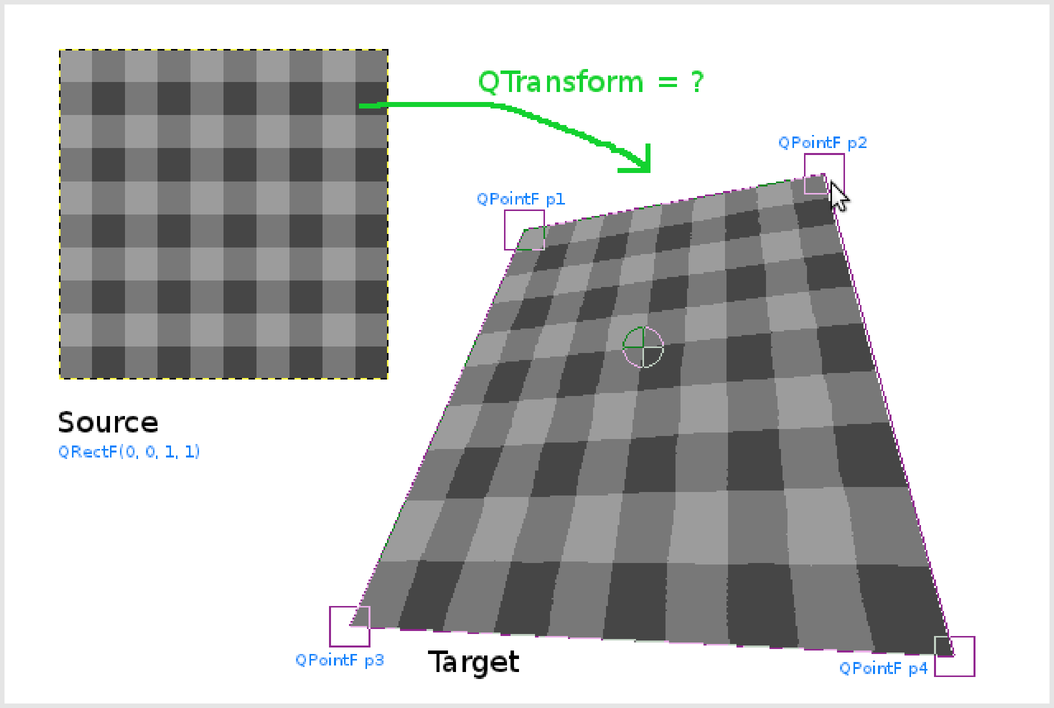
Ex. Project



https://web.itu.edu.tr/sahinyu/hw2_cat.mp4

Projective (Perspective) Transformation

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

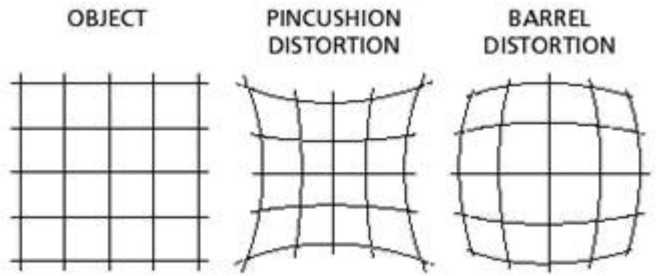


Other transformations

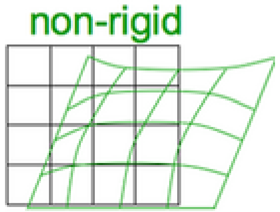
- Other transformations are possible, including those that do not involve a matrix, and instead a more general function that transforms pixel locations. What's needed is a way to describe the transformation

$$\mathbf{x}' = \mathbf{T}(\mathbf{x})$$

- Examples of other common transformations include

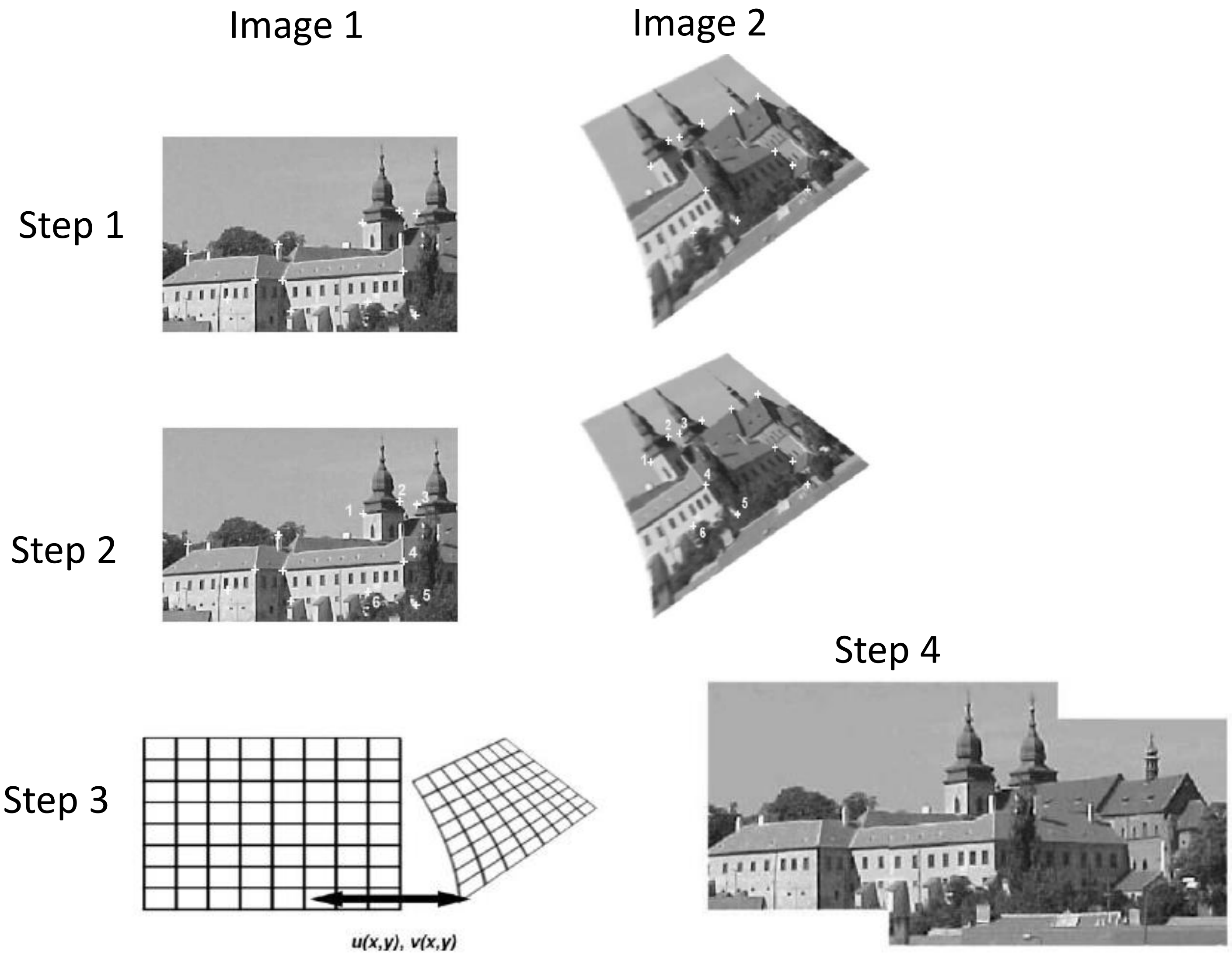


Radial lens distortion



Non-rigid transformation

Image Registration

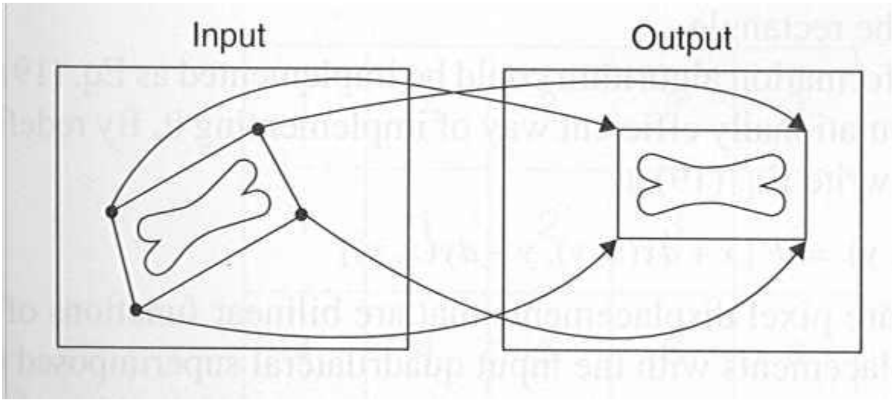


Four steps of image registration:

- Feature detection (corners were used as the features in this case).
- Feature matching by invariant descriptors (the corresponding pairs are marked by numbers).
- Transform model estimation exploiting the established correspondence.
- Image re-sampling and transformation using appropriate interpolation technique.

Registration by Control Points

- Specify the spatial transformation as displacement values for selected *control points* in the image:
  x=a(u,v)
  y=b(u,v)



J(u,v) = I(x,y) = I[a(u,v),b(u,v)]

1. Determine a set of suitable control points, from which the transform parameters (e.g. here polynomial) are determined.

2. Apply the actual correction to the image data using this transform (by finding all corresponding pixel locations in the two images)

3. Remap intensity data (i.e. interpolate)

Polynomial Transformation

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

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

Example: 2nd-order polynomial transformation.

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

Polynomial Transformation

In matrix form, we have

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

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)

12 coefficients need to be estimated, therefore at least six point correspondence are needed.

Image Warping

- Warp a given image to a new purposefully distorted image.

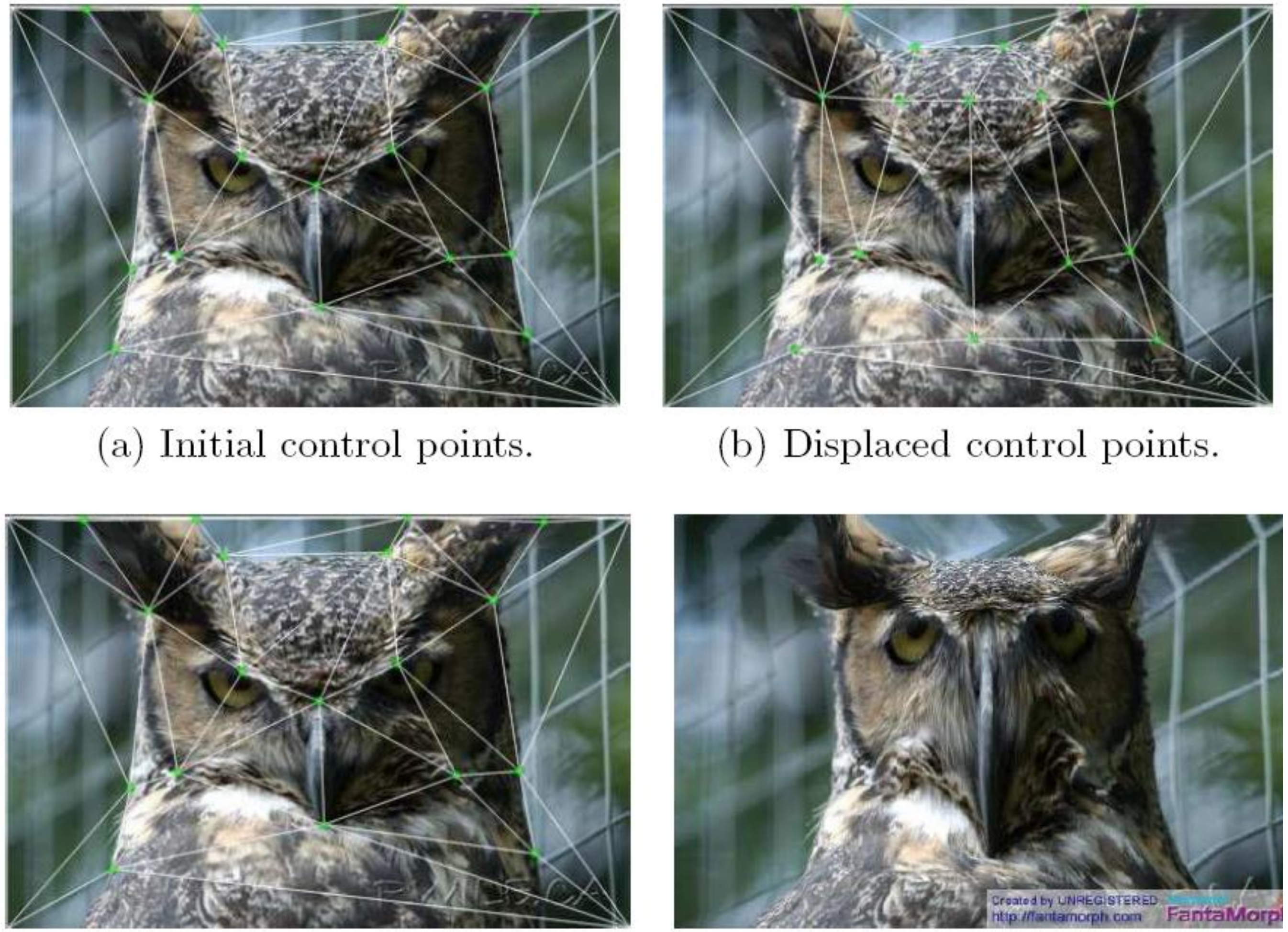
- Given a source image I, and the correspondences between original control points pi in I and desired destination points $q_i, i = 1, \dots, n$

- Generate a Warped image J such that

$$J(q_i) = I(p_i) \quad \forall i$$

Image Warping

- Warp the single image according to a final set of point positions



(a) Initial control points.

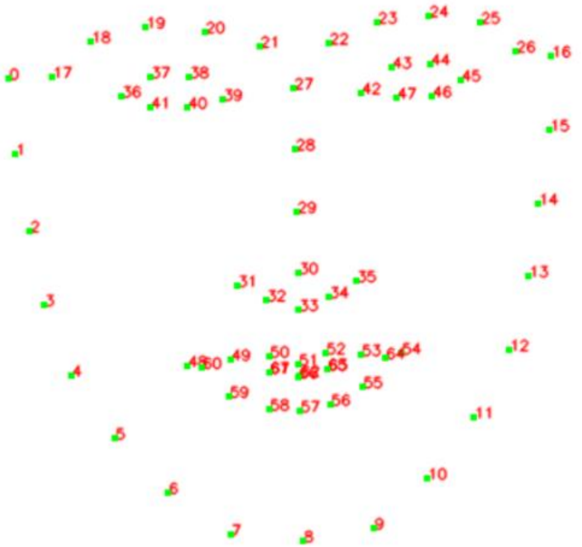
(b) Displaced control points.

(c) Initial image.

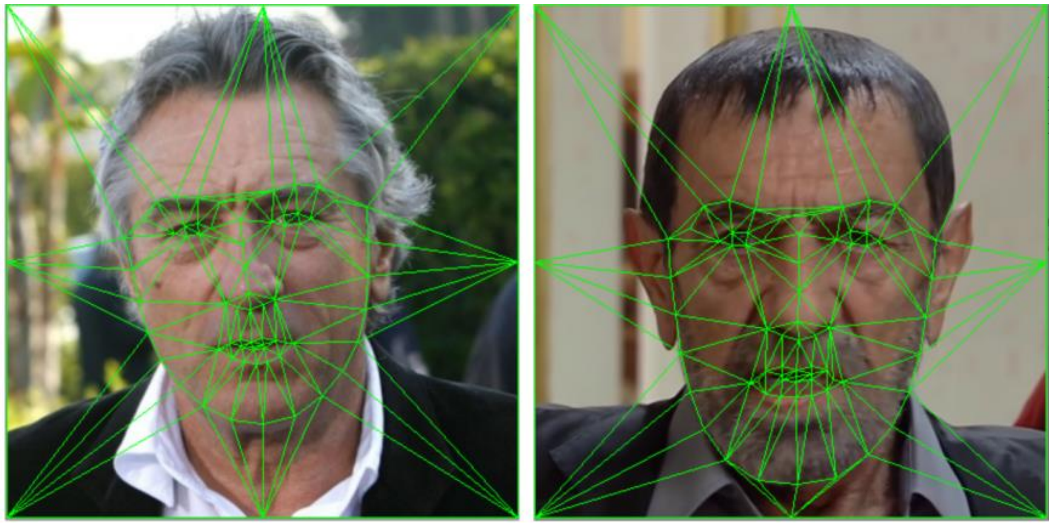
(d) Warped image.

Image Morphing

Find facial landmarks



Divide the images to matching Delaunay triangles



Transform and blend each triangle step by step.

- A Generative Adversarial Network is a network which generates an output depending on an random or conditional input. The term ”adversarial” comes from the training process, since two networks are used: Generator and Discriminator. While Generator network creates an output it also tries to fool the Discriminative network which has a task to differentiate real and fake images.

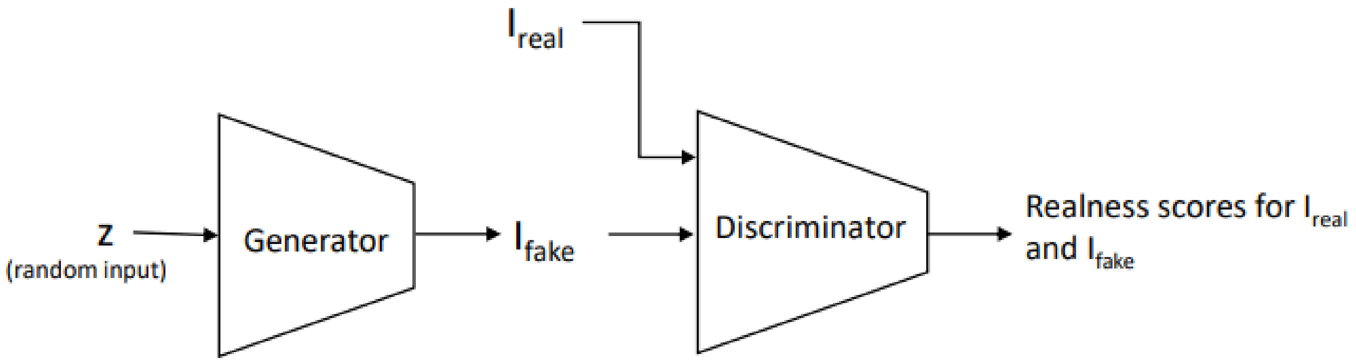


Image Morphing via GANs