Computer Vision

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

University of Bucharest, 2nd semester, 2023-2024

Exam – evaluation in June

Grade = min(10, P1 + P2 + bonus)

- P1 = project 1 = 5 points (+ some bonus?)
- P2 = project 2 = 5 points (+ some bonus?)
- bonus = discuss later
- no constraints, with 4.99 you fail, with 5 you pass

Project 1 – good to know

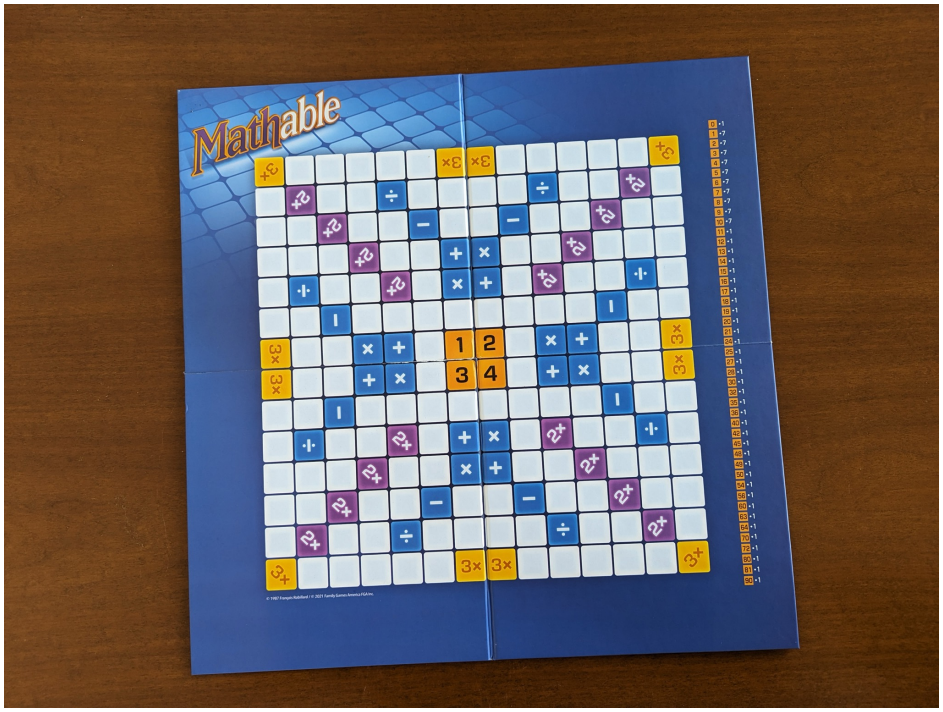
- one task to solve (several scenarios) = 4.5p + 0.5p + 0.5 = 5.5p maximum
- deadline: in ~ 4 weeks time, Tuesday, 14th of May 2024, 23:59
  - **hard deadline – no late submission policy**
  - upload your source code
  - on Wednesday, 15th of May 2024 we will release the test set. You will run your (untouched) code on test images and send us your results. Immediately we will published the intermediate grades (based on your results). Final grades for Project 1 will be published after carefully checking your code, your pdf report and listening to your oral presentation (~2 weeks time).

- do not share/copy the solution with/from your colleagues: you + your colleague/s will get 0 points

Project 1:
Mathable score calculator

Mathable

Mathable is a game based upon mathematical equations which must be formed on the playing board. It has been described as being like playing Scrabble but using numbers (Figure 1). In Mathable, the players make use of a playing board with normal squares (white squares), squares imposing some constraints (blue squares) and squares containing an award (purple squares marked 2× and orange squares marked 3×), as well as 106 tiles with numbered tokens. The game can be played by two, three of four players with the goal of forming valid mathematical equations and scoring as many points as possible.



Tokens

In completing mathematical equations on the board, players use tiles with numbered tokens. For simplicity, we refer to them as simply *tokens*. There are in total 46 different types of tokens, which usually can help in completing a mathematical equation. The tokens are:

- digits 0-9 (for a total of 10 tokens in interval [0,9]);
- numbers 10-19 (for a total of 10 tokens in interval [10,19]);
- numbers 20, 21, 24, 25, 27, 28 (for a total of 6 tokens in interval [20,29]);
- numbers 30, 32, 35, 36 (for a total of 4 tokens in interval [30,39]);
- numbers 40, 42, 45, 48, 49 (for a total of 5 tokens in interval [40,49]);
- numbers 50, 54, 56 (for a total of 3 tokens in interval [50,59]);
- numbers 60, 63, 64 (for a total of 3 tokens in interval [60,69]);
- numbers 70, 72 (for a total of 2 tokens in interval [70,79]);
- numbers 80, 81 (for a total of 2 tokens in interval [80,89]);
- number 90 (for a total of 1 token in interval [90,99]).

The specific number of tokens available in the game is specified on the right edge of the board. For each value between 1 and 10 there are 7 tokens in the game (for a total of 70 tokens) while for the rest of 36 values there is a single token with that value (for a total of 36 tokens). So, in total, there are 106 tokens in the game.

Tokens

In completing mathematical equations on the board, players use tiles with numbered tokens. For simplicity, we refer to them as simply *tokens*. There are in total 46 different types of tokens, which usually can help in completing a mathematical equation. The tokens are:

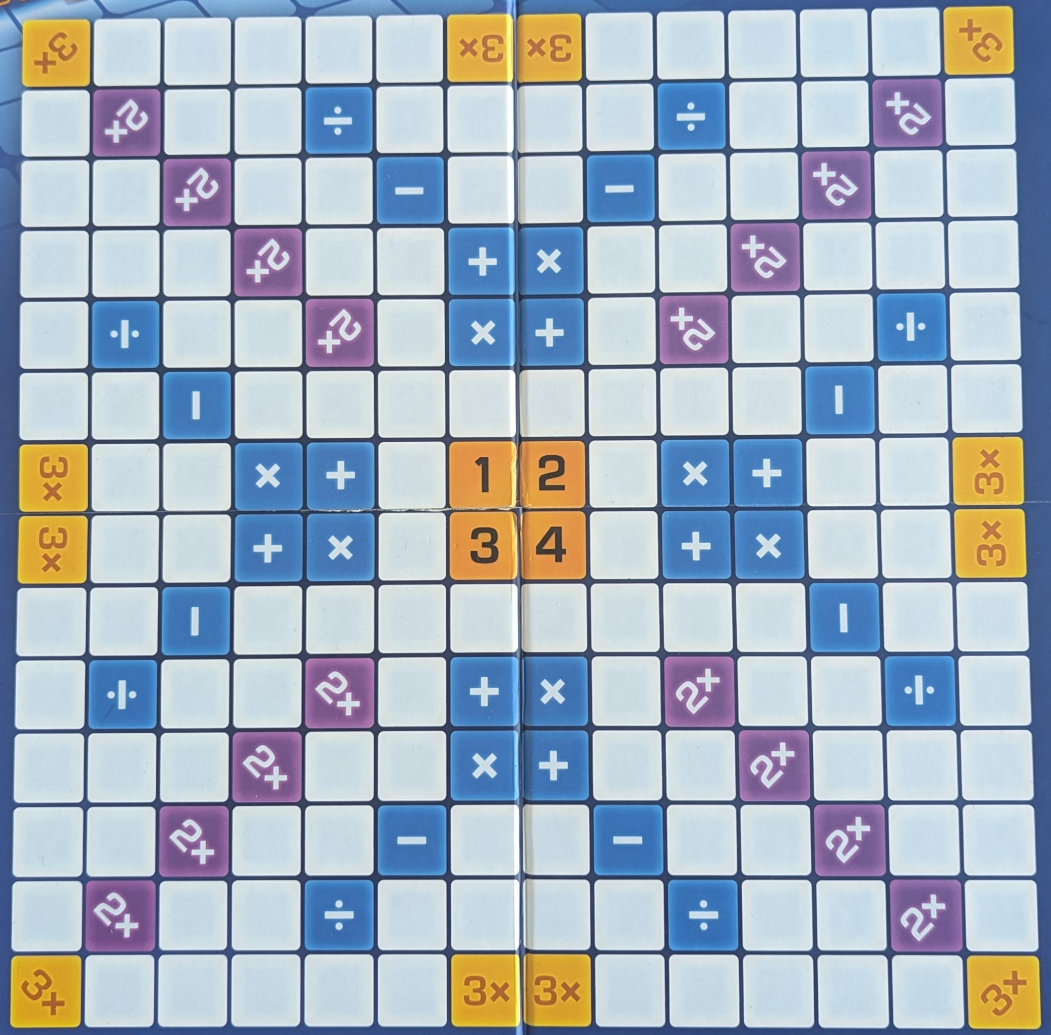
Board

The Mathable playing board (Figure 1) is divided into a 14 × 14 grid of squares. The board is marked with coloured squares, some of them containing an award and thus increasing the score and some of the imposing constraints. There are in total 28 coloured squares with an award:

- 16 purple squares for double points;
- 12 orange squares for triple points.

The constraint squares are marked with blue and contain an addition, subtraction, multiplication or division sign. In order to occupy a specific square by placing a token there, the player must make an equation that corresponds to the sign of that square. There are in total 32 squares with constraints.

Mathable



0	·1
1	·7
2	·7
3	·7
4	·7
5	·7
6	·7
7	·7
8	·7
9	·7
10	·7
11	·1
12	·1
13	·1
14	·1
15	·1
16	·1
17	·1
18	·1
19	·1
20	·1
21	·1
24	·1
25	·1
27	·1
28	·1
30	·1
32	·1
35	·1
36	·1
40	·1
42	·1
45	·1
48	·1
49	·1
50	·1
54	·1
56	·1
60	·1
63	·1
64	·1
70	·1
72	·1
80	·1
81	·1
90	·1

Valid and invalid configurations

A mathematical equation is completed by adding, subtracting, multiplying or dividing two adjacent numbers putting a token with the result on the next empty square, be in to the right or left, below or above the two original tokens, never diagonally or between tokens. Figure 2 shows valid and invalid configurations on a general Mathable board. The

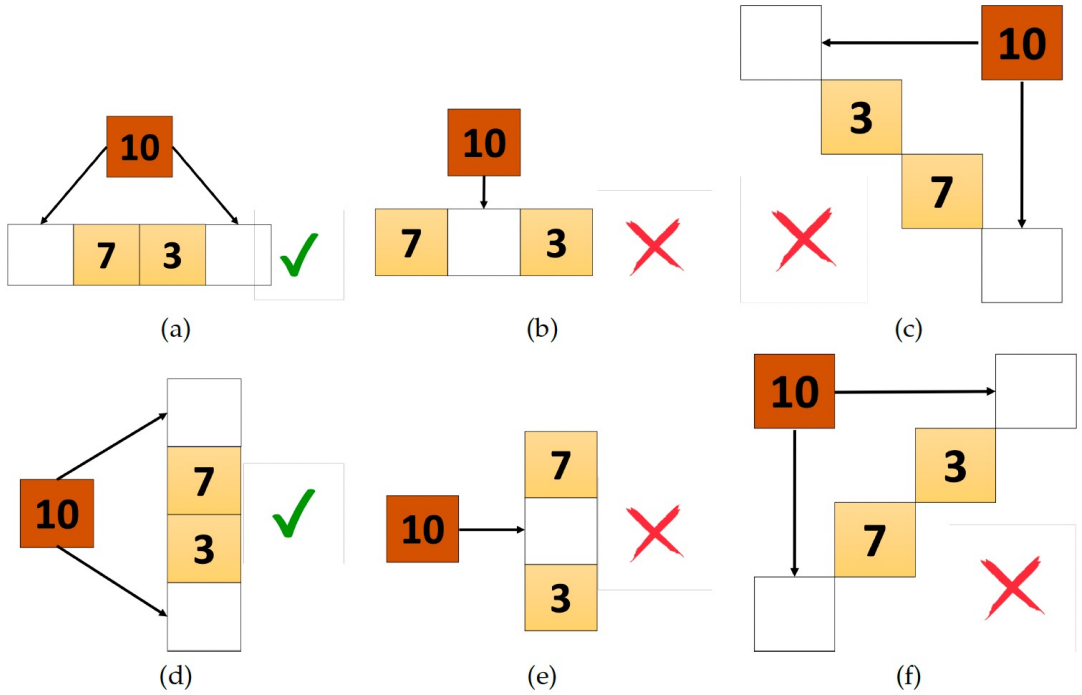


Figure 2: **Valid and invalid configurations in Mathable.** *Examples in (a) and (d) illustrate valid moves, where the token* 10 *can be placed either in the right or in the left (case (a)) or in below or above (case (d)) the two original tokens* 7 *and* 3 *completing the addition. Examples in (b) and (e) show invalid moves, where the token* 10 *cannot be placed horizontally between tokens (case (b)) or vertically between tokens (case (e)). Examples in (c) and (f) show invalid moves, where the token* 10 *cannot be placed diagonally wrt tokens* 7 *and* 3.

The play

We consider the scenario with only two players, Player 1 and Player 2. Each player receives at the beginning of the game 7 tokens drawn from a bag with all tokens (initially there are 106 tokens in the bag). At each round, the current player can make several moves by completing mathematical equations on the board. After each move, the current player gets a score. The maximum number of moves per round is limited to 7. At the end of his turn, each player draws tokens randomly from the bag, to bring the amount of his holder back to 7. The score obtained by the current player in the round is obtained by adding the scores obtained at each move.

Scoring

The score of the current player after each move is based on the token placed on the board and can increase in some cases.

Score values of a token. Each token has a score value equal to the number on the token.

Bonus for multiple equations. If a token, when being placed, completes more than one equation, the points are gained for each equation completed.

Squared with an award. A purple square marked 2× double the amount of points of the token on that square, an orange square marked 3× triples the amount of points.

Scoring example

Figure 3 shows twelve moves made at the beginning of a game. We list below the twelve moves and offer detailed explanations about computing the corresponding score of each player after each move. The two players in this example have the following tokens:
Player 1: 1, 2, 8, 12, 16, 17, 42;
Player 2: 3, 4, 6, 7, 7, 8, 21.

Examples of scoring

Move 1. Player 1 may choose one of the four mathematical operations using the numbers 1 to 4 positioned in the center of the playing board. Player 1, who has a token 12 on his holder, chooses to multiply the numbers 3 and 4. He may place token 12 either to the left of the 3 or to the right of the 4. He chooses to place it at the right. Since the equation is correct, the player earns 12 points.

Examples of scoring

Move 2. Player 1 places his second token. Since he has an 8, he decides to multiply 2 and 4 and chooses to place the result above the 2. An alternative move would have been to position the token 8 bellow the 4. Thus, player 1 earns another 8 points.

Examples of scoring

Move 3. Player 1 places his third token. He takes the token 2 from the holder, subtracts 1 from 3 and places the 2 below the 3. He could also place the 2 above the 1, but chooses not to do so. The player adds 2 points to his total.

Examples of scoring

Move 4. Player 1 places his fourth token. He subtracts 1 from 2 and places token 1 left of the 1 already on the board. He could also place it at the right side. Thus, player 1 earns another 1 point.

Examples of scoring

Move 5. Player 1 places his fifth token. At this point, the square to the right of the square occupied by token 12 is blue, marked with a + sign, meaning that the player has to add the two tokens if he wishes to occupy this square. The player takes token 16 from his holder and places it on the blue square, thus fulfilling the constraint. The player adds 16 points to his score. The player cannot place another token, and so his turn is over. He totals all points earned in this turn, and finds his total score is 39.

Examples of scoring

Move 6. Second player's turn. Player 2 decides to add 3 and 4, since he has a 7 on his holder. He places the token to the left of the 3. He receives 7 points.

Examples of scoring

Move 7. Player 2 places his second token. Subtracting 1 from 7 gives him a total of 6. He places the token 6 under the 7 (he could also have placed it above the 1). He adds 6 points to his total.

Examples of scoring

Move 8. Player 2 places his third token. The player decides to add 6 and 2, totalling 8. He has two possibilities: Placing the 8 to the left of the 6, giving him 8 points or placing the 8 to the right of the 2, and thus profiting from the Bonus for Multiple Equations, since the vertical 2×4 also results in 8. The player opts for the second choice, adding another 8 points (a total of 16) to his score. REMARK: If the two tokens above this square do not complete the equation, the player may still place a token there, through only counting 8 points for the addition.

Examples of scoring

Move 9. Player 2 places his fourth token. This time he uses token 4, adding 3 and 1. He places the token above the 1. He earns 4 points.

Examples of scoring

Move 10. Player 2 places his fifth token. To make use of token 3 he divides 6 by 2. The result 3 is placed next to the 6. He adds 3 points.

Examples of scoring

Move 11. Player 2 places his sixth token. Multiplying 7 by 3 totals 21. This token is placed to the left of the 7. He adds 21 points. Please also note that, again, the current player fulfills the constraint of the blue square on which he has placed his token.

Examples of scoring

Move 12. Player 2 places his seventh token. He places the 7 below the 3 since 21 divided by 3 is 7. He cannot place the token above the 21, since a constraint square does not allow him to divide. Placing the 7 below the 3, he lands on an award square (2×) which doubles the amount of points earned in this move to $14 = 7 \times 2$. The total points in his turn is thus 71.

Training data

-training data: 4 games, each game has 50 images (about each 4 moves there is a turn)
-training data: in total there are 200 images + 200 ground-truth annotations

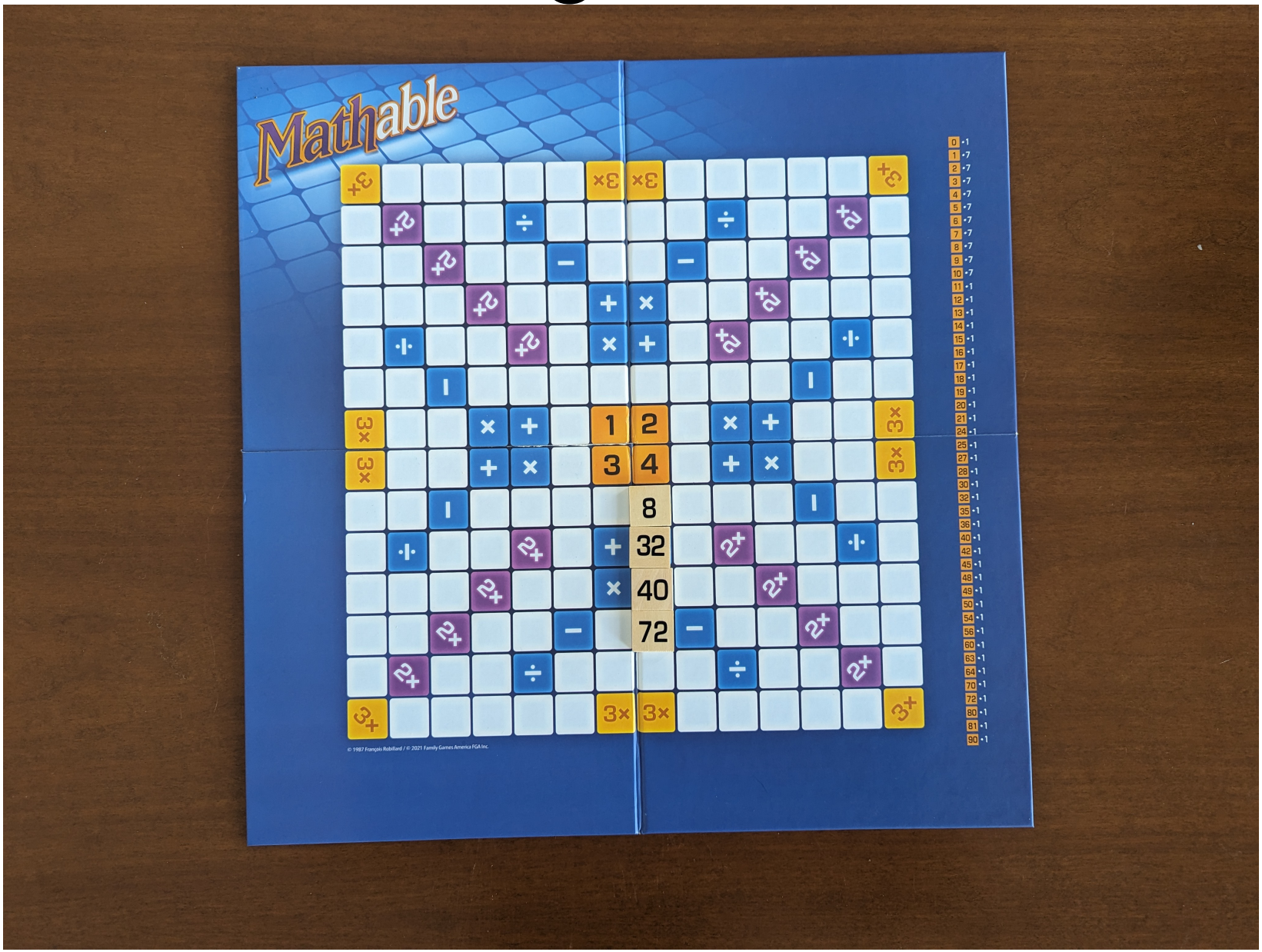
Game 1: contains 50 images with regular views, photos taken from above the board
1_01.jpg, 1_02.jpg, …, 1_50.jpg

Game 2: contains 50 images with rotated views, photos taken from above the board with some rotation
2_01.jpg, 2_02.jpg, …, 2_50.jpg

Game 3: contains 50 images with perspective views, photos taken by tilting the phone
3_01.jpg, 3_02.jpg, …, 3_50.jpg

Game 4: contains 50 images with mixed view (regular, rotation, perspective)
4_01.jpg, 4_02.jpg, …, 4_50.jpg

Regular view



Rotated view

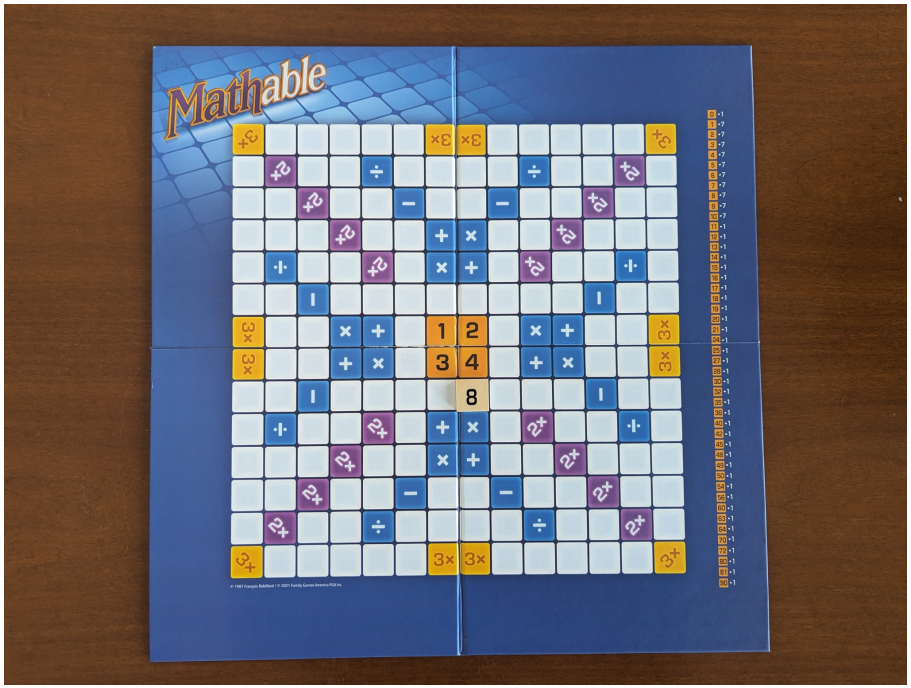
Perspective view

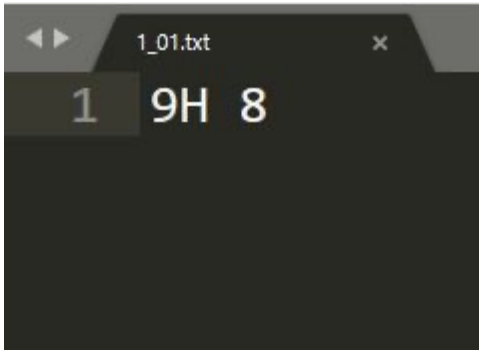
Ground-truth annotations

For each image we annotate:

- the position of the token added to the board, taken in the order from left to right and from top to down. We specify the position using numbers 1-14 for rows and letters A-N for columns. Take into consideration that only one token is placed at each move.
- the number on the token at the corresponding position on the board.

1_01.jpg

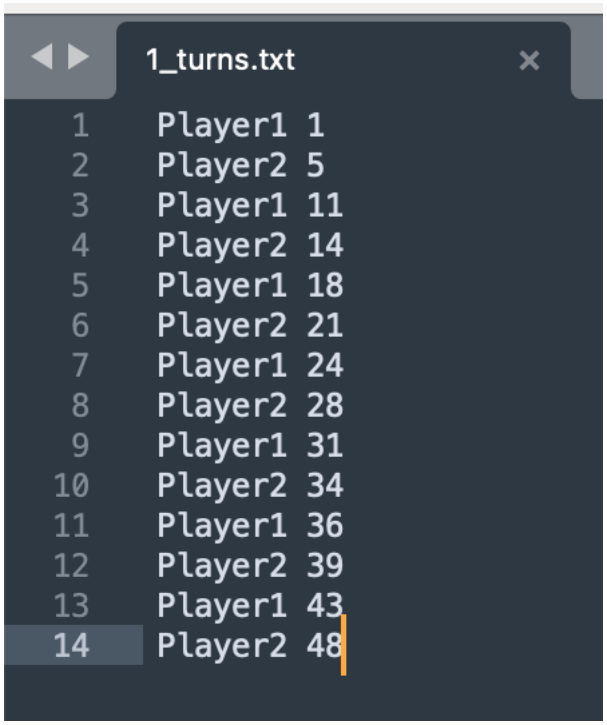


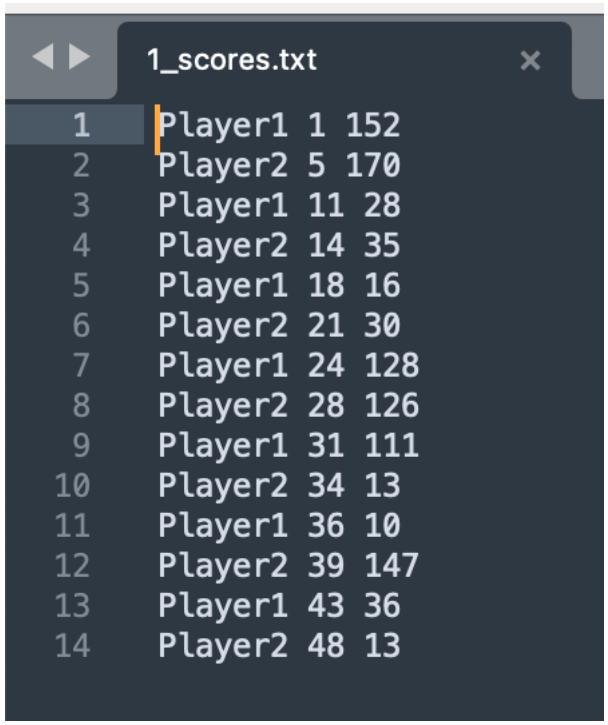


Other annotations

For each game:

- there is a .txt file that annotates the beginning of a turn for a player (g_turns.txt where g is the game number). The .txt file g_scores.txt, where g is the game number, contains the scores of each player after each round, expanding thus the file g turns.txt.





Test Data

-same as training data: 4 games, each game has 50 images (about each 4 moves there is a turn)
-testing data: 200 images (will be made available after the deadline for submitting the source code, on Wednesady 15th of May)

For each test image you should produce a similar txt file (like in the annotations) containing:

1. the positions of the token added to the board (form left to right, up to down) – use notations 1-14 for rows and A-N for columns
2. the token
3. the score after each turn (g_scores.txt)

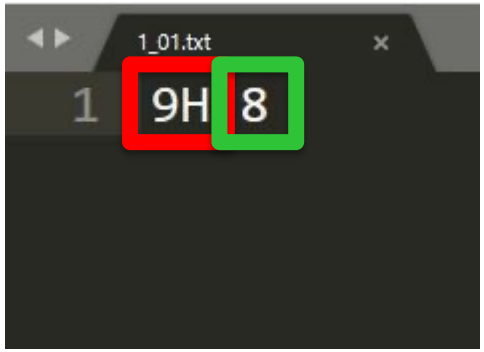
Each correctly solved image worth 0.0175 points:

- correct position: 0.01 points
- correct token: 0.0075 points

Total: 200 images x 0.0175p = 3.5 points

- corect score after each turn: 0.02 points

Total: 50 turns x 0.02p = 1 point



Ex officio: 0.5 points (correct format + pdf)
Oral presentation: 0.5 point (mandatory)

Requirements

Your job is to write a program in Python/Jupyter notebook that automatically solves the task of extracting information of the current move depicted in a test image. For each test image you have to output the corresponding information similar to the annotation files, thus specifying the position of the newly added token to the board and the token. Additionally, you have to provide a similar .txt file of the form (*g_scores.txt*, where *g* is the test game number, containing the scores of each player after each round, expanding thus the file *g_turns.txt*.

This project worths 5.5 points. We will grade your project based on the performance achieved by your algorithms on each of 200 test images.

You will receive a test set containing 200 testing images organized in 4 games of 50 moves. The distribution of images in the test data follows the distribution of train data, meaning that the images were acquired in the same conditions and also that the first game will contain images with regular views, the second game will contain images with rotated views, the third game will contain images with perspective view, the fourth game will contain images with mixed views. For each test image you have to output a .txt file containing information similar to the annotation files and for every game a .txt file with scores after each round. Each correctly solved test image worths 0.0175 points. For correctly specifying the position of the added token you receive 0.01 points per image (move), for correctly specifying the token placed at the corresponding position on the board you receive 0.0075 points per image (move) and for correctly specifying the total score of the current player after a turn you receive 0.02 point per turn. You receive 0.5

points from *ex officio* conditioned on the fact that you respect the format of the submitted results, such that our evaluation script works smoothly on your provided results.

The oral presentation of this project (face-to-face or online) will be scheduled in the weeks 14 – 17 or 20 – 24 of May. It will take around 10-15 minutes in which Alexandra or Bogdan will ask question regarding implementation. The oral presentation will count for 0.5 points and is mandatory for each student submitting his solution for this project.

Deadlines

Submit a *zip archive* containing your code (python files or Jupyter notebook files), all auxiliary data that you are using (templates, models, etc.) and a pdf file describing your approach until Tuesday, 14^{th} of May using the following link https://tinyurl.com/CV-2024-PROJECT1-SUBMISSIONS. Please do not include in your zip archive any unuseful data (like training images, we already have them!!!). Notice that this is a hard deadline, no projects will be accepted after the deadline. Your code should include a README file (see the example in the materials for this project) containing the following information: (i) the libraries required to run the project including the full version of each library; (ii) indications of how to run the solution and where to look for the output file. Students who do not describe their approach (using a pdf file) will incur a penalty of 0.5 points.

On Wednesday 15^{th} of May we will make available the test data. You will have to run your solution on the test images provided by us and upload your results in the same day as a zip archive using the following link https://tinyurl.com/CV-2024-PROJECT1-RESULTS.

Materials for Project 1

tinyurl.com/CV-2024-Project1

Lecture 8

Course structure

1. **Features and filters: low-level vision**

   Linear filters, color, texture, edge detection, template matching

2. **Grouping and fitting: mid-level vision**

   Fitting curves and lines, robust fitting, RANSAC, Hough transform, segmentation

3. **Multiple views**

   Local invariant feature and description, epipolar geometry and stereo, object instance recognition

4. **Object Recognition: high – level vision**

   Object classification, object detection, part based models, bovw models

5. **Video understanding**

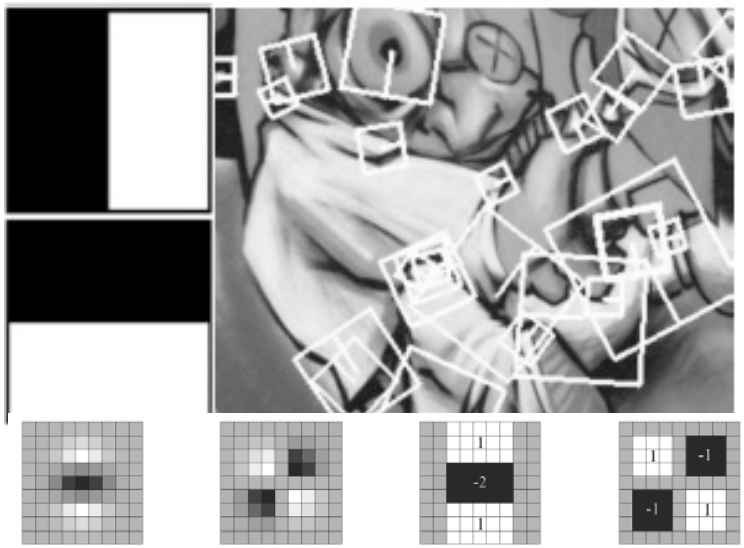
   Object tracking, background subtraction, motion descriptors, optical flow

SIFT descriptor

- Extraordinarily robust matching technique
  - Can handle changes in viewpoint
    - Up to about 60 degree out of plane rotation
  - Can handle significant changes in illumination
    - Sometimes even day vs. night (below)
  - Fast and efficient—can run in real time
  - Lots of code available, e.g. http://www.vlfeat.org/overview/sift.html

Steve Seitz

Local Descriptors: SURF



Fast approximation of SIFT idea

Efficient computation by 2D box filters & integral images

⇒ 6 times faster than SIFT

Equivalent quality for object identification

GPU implementation available

Feature extraction @ 200Hz (detector + descriptor, 640×480 img)

http://www.vision.ee.ethz.ch/~surf

Many other efficient descriptors are also available

[Bay, ECCV'06], [Cornelis, CVGPU'08]

Local Descriptors: ORB

- Many similarities to SIFT/SURF
- Designed for efficiency and robustness to orientation
- Not designed for scale robustness
- Used for tracking and long-range matching in ORB-SLAM

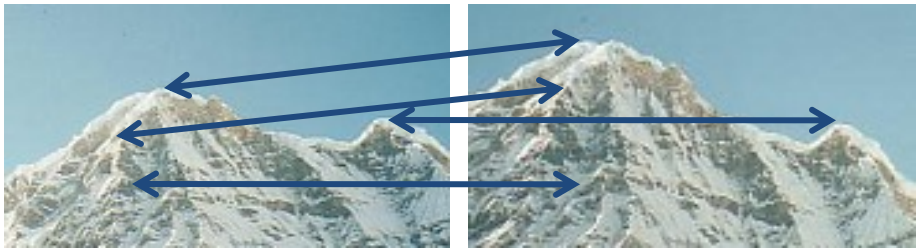
http://www.willowgarage.com/sites/default/files/orb_final.pdf (ICCV 2011)
http://webdiis.unizar.es/~raulmur/orbslam/

Local features: main components

1) Detection: Identify the interest points

2) Description:Extract vector feature descriptor surrounding each interest point.

3) Matching: Determine correspondence between descriptors in two views



Slide credit: Kristen Grauman

Matching local features

Slide credit: Kristen Grauman

Matching local features

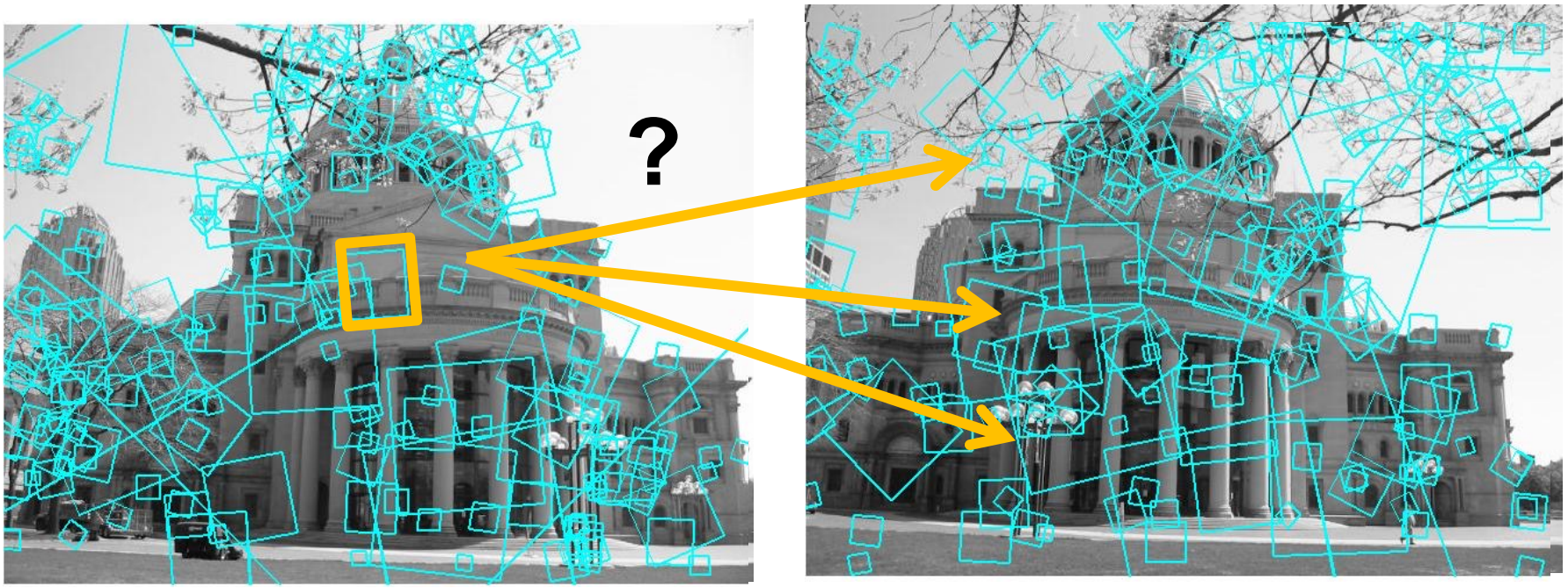


Image 1 Image 2

To generate **candidate matches**, find patches that have the most similar appearance (e.g., lowest SSD)

Simplest approach: compare them all, take the closest (or closest k, or within a thresholded distance)

Slide credit: Kristen Grauman

Ambiguous matches

Image 1

Image 2

At what SSD value do we have a good match?

To add robustness to matching, consider **ratio** :

dist to best match / dist to second best match

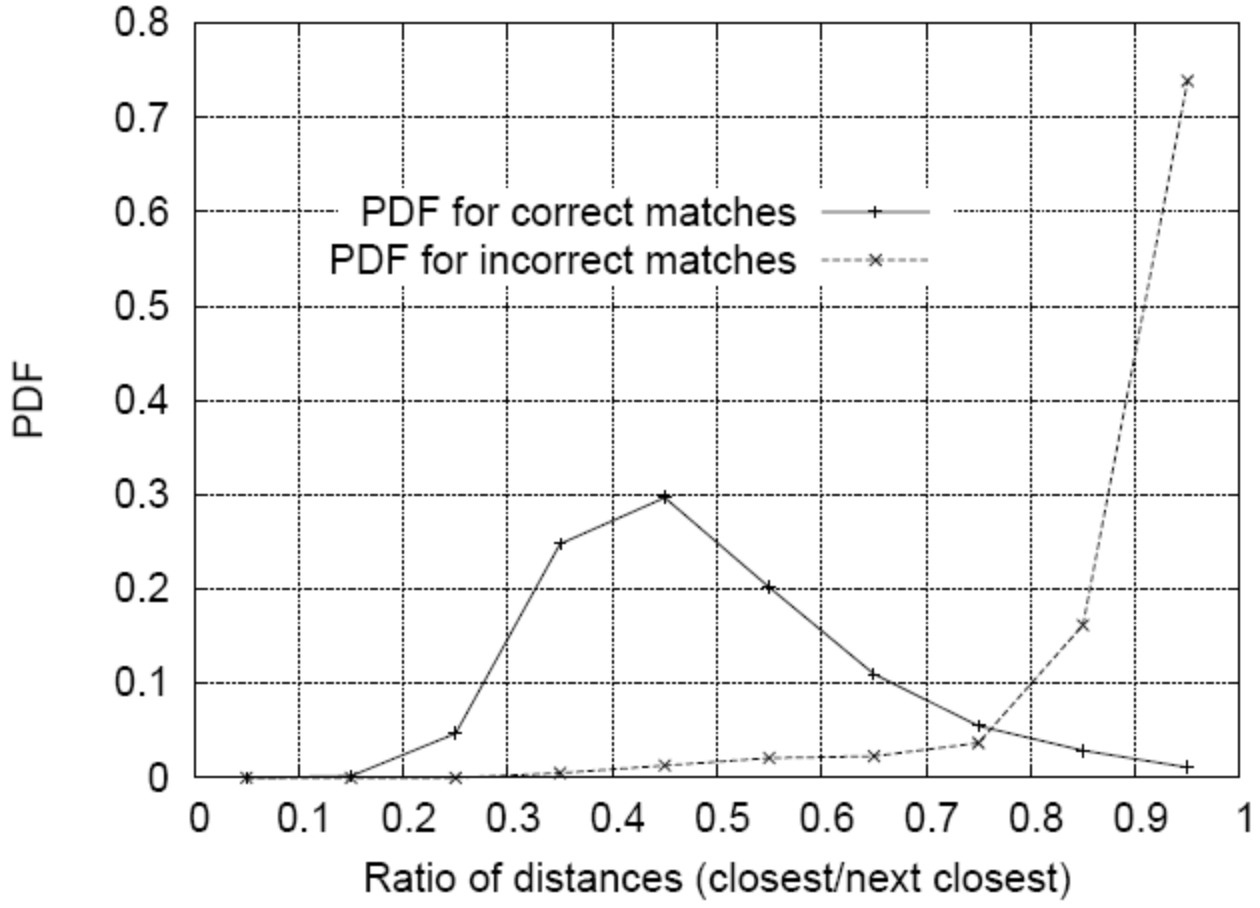
If low, first match looks good.

If high, could be ambiguous match.

Slide credit: Kristen Grauman

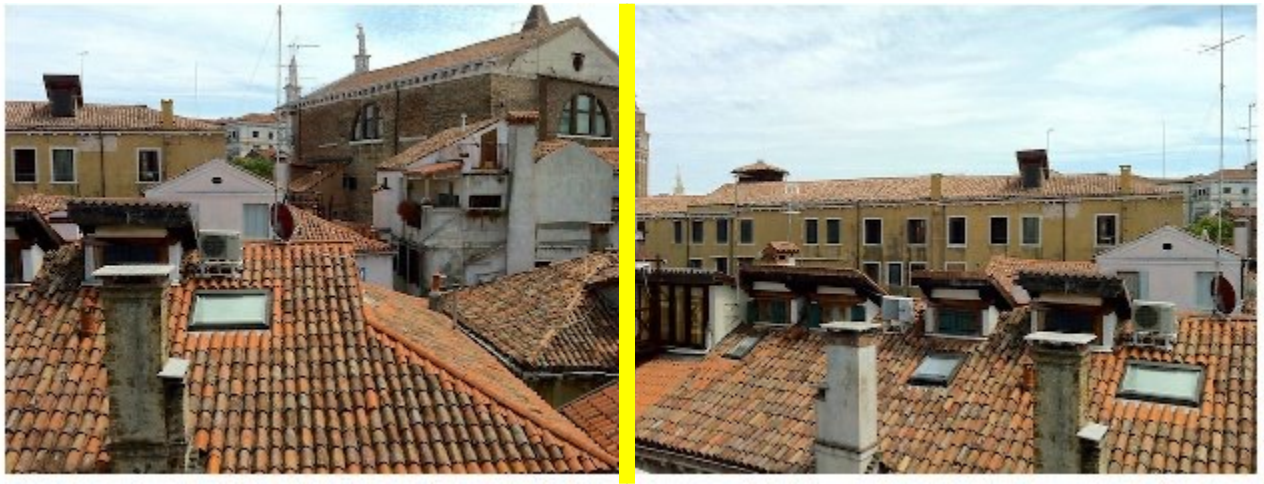
Matching SIFT Descriptors

- Nearest neighbor (Euclidean distance)
- Threshold ratio of nearest to 2nd nearest descriptor

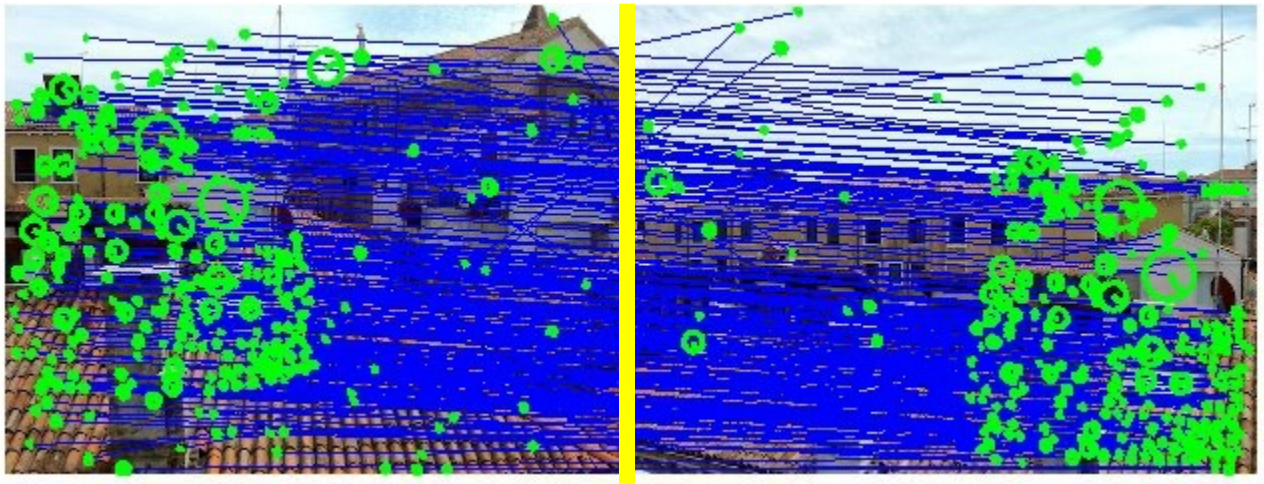


Lowe IJCV 2004

SIFT (preliminary) matches



img1 img2



img1 img2

http://www.vlfeat.org/overview/sift.html

Image alignment

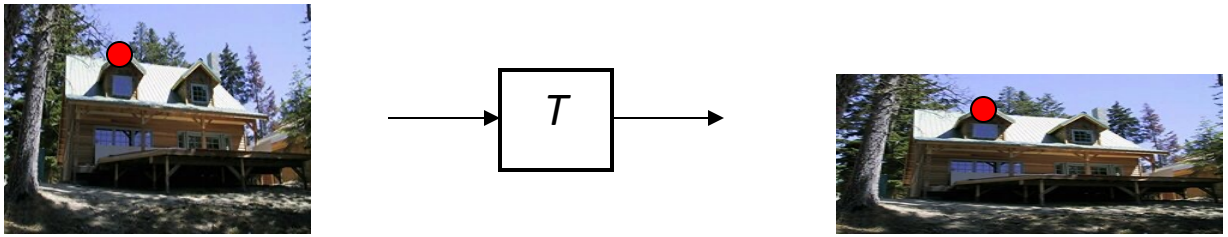
- Alignment: find parameters of model that maps one set of points to another

- Typically want to solve for a global transformation that accounts for most true correspondences

- Difficulties
  - Noise (typically 1-3 pixels)
  - Outliers (often 30-50%)
  - Many-to-one matches or multiple objects

Lab class 5: image morphing

Parametric (global) warping



p = (x,y)

p' = (x',y')

Transformation T is a coordinate-changing function:

$$p' = T(p)$$

What does it mean that *T* is global?

- Is the same for any point p
- can be described by just a few numbers (parameters)

For linear transformations, we can represent T as a matrix

$$p' = \mathbf{T}p$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Common transformations

original

Transformed

translation

rotation

aspect

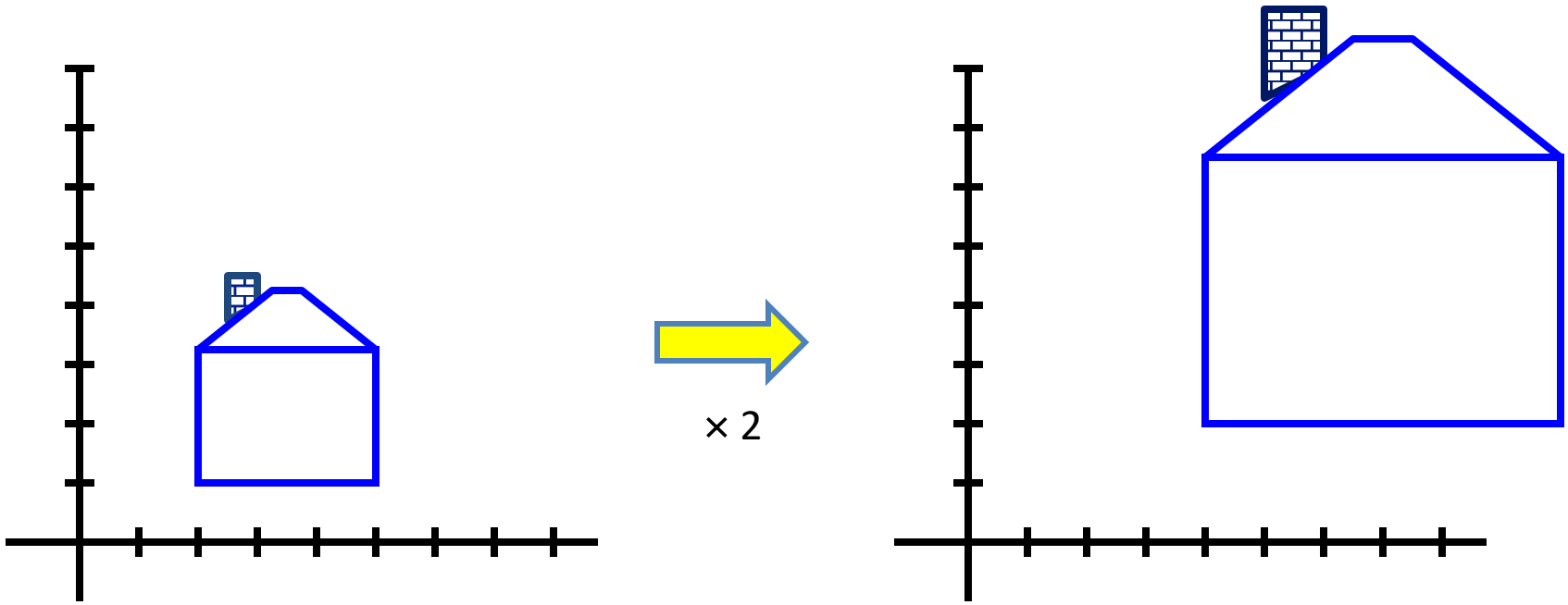
affine

perspective

Slide credit A. Efros and/or S. Seitz

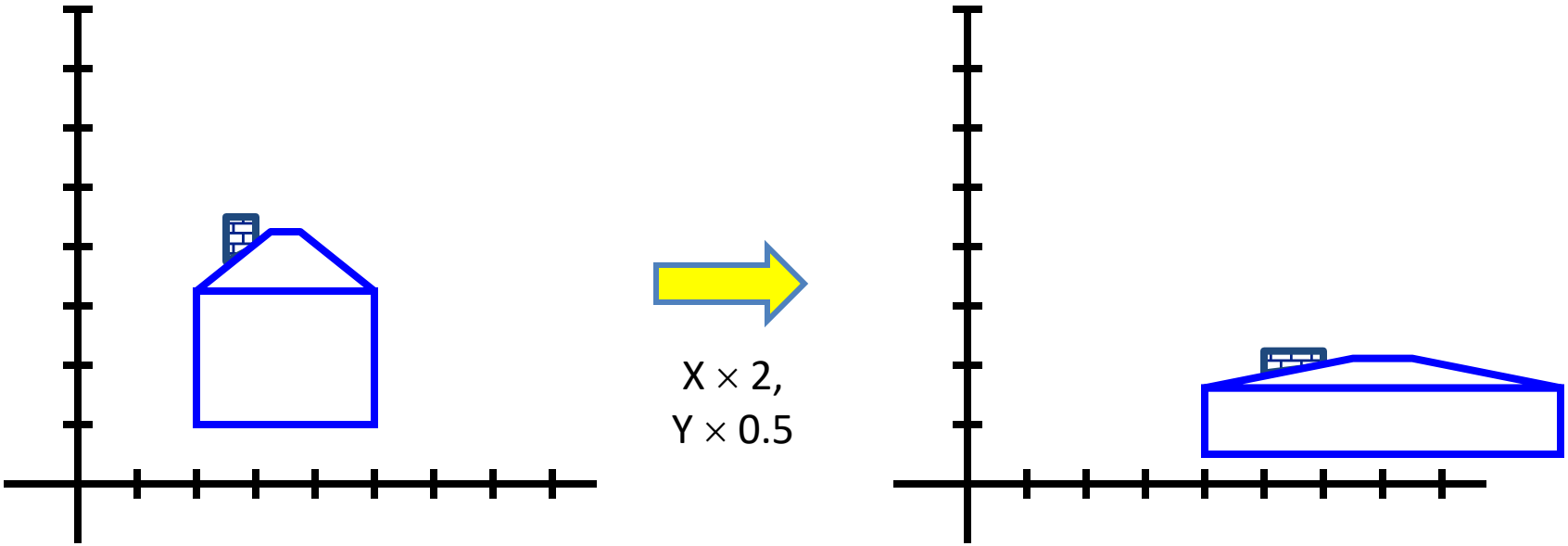
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component:



Scaling

- Scaling operation:

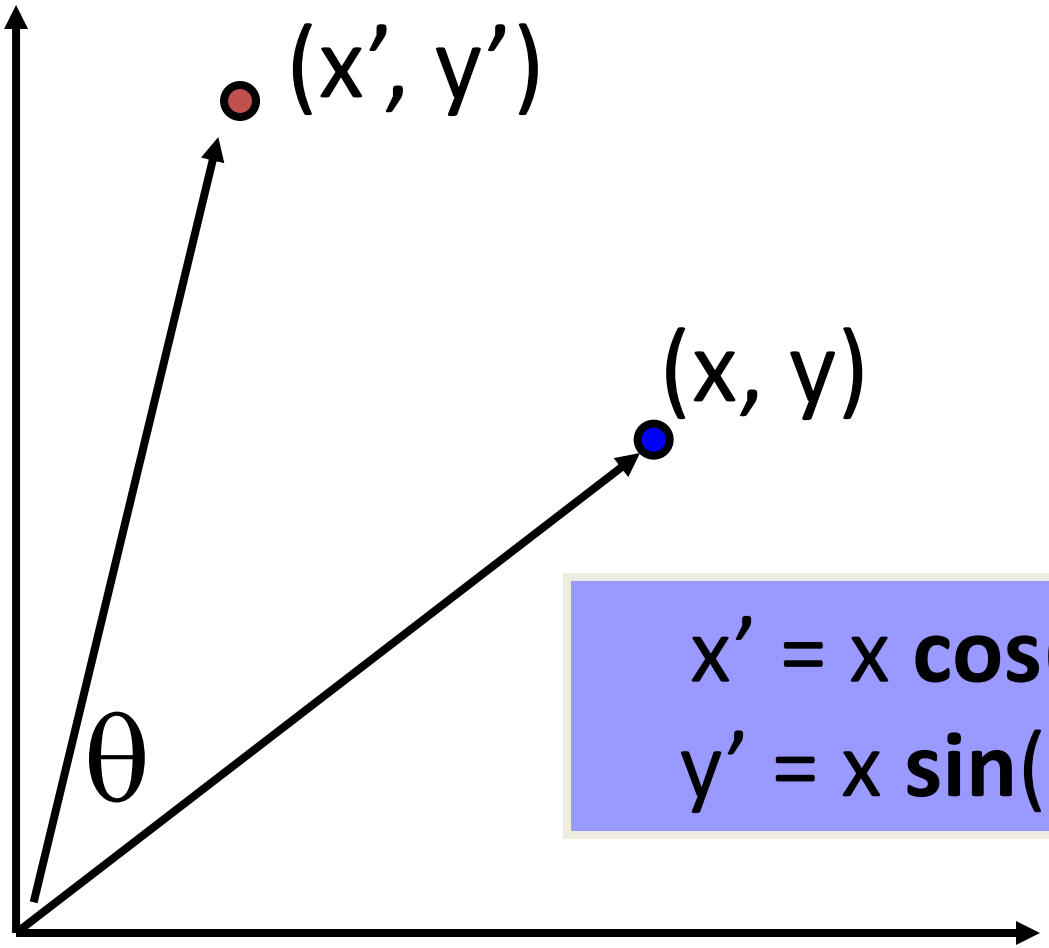
$$x' = ax$$

$$y' = by$$

- Or, in matrix form:

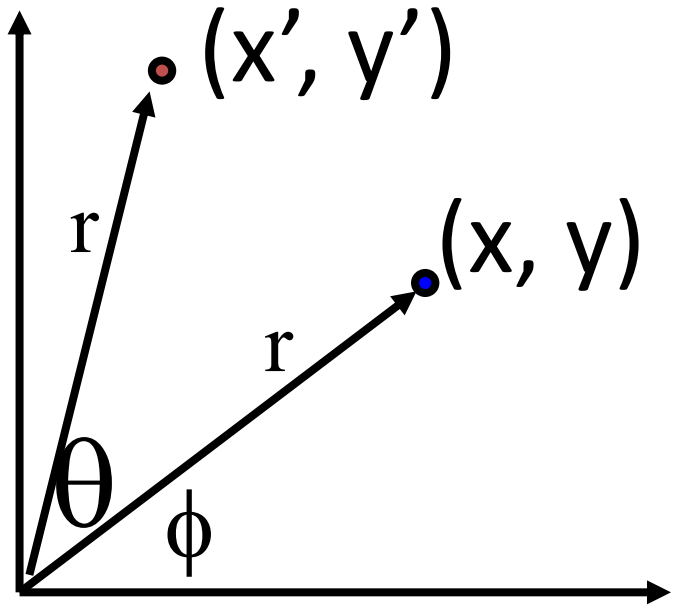
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

2-D Rotation



$$x' = x \cos(\theta) - y \sin(\theta)$$
$$y' = x \sin(\theta) + y \cos(\theta)$$

2-D Rotation



Polar coordinates…

$x = r \cos(\phi)$

$y = r \sin(\phi)$

$x' = r \cos(\phi + \theta)$

$y' = r \sin(\phi + \theta)$

Trig Identity…

$x' = r \cos(\phi) \cos(\theta) - r \sin(\phi) \sin(\theta)$

$y' = r \sin(\phi) \cos(\theta) + r \cos(\phi) \sin(\theta)$

Substitute…

$x' = x \, \mathbf{cos}(\theta) - y \, \mathbf{sin}(\theta)$

$y' = x \, \mathbf{sin}(\theta) + y \, \mathbf{cos}(\theta)$

2-D Rotation

This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

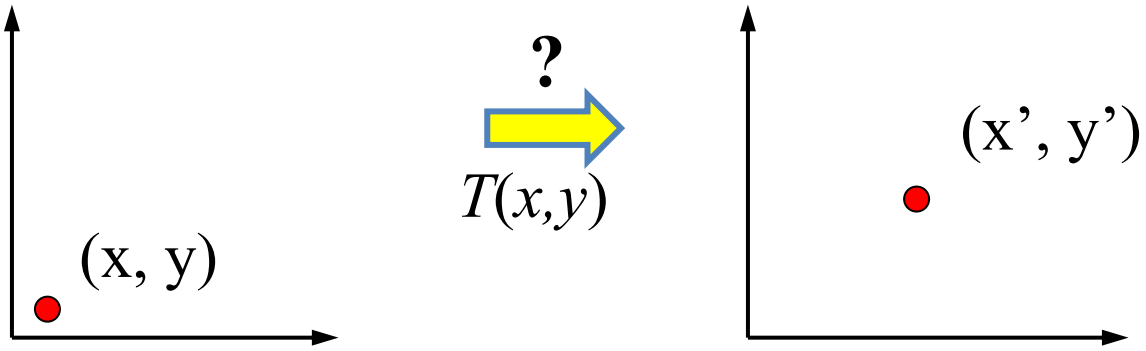
Even though sin(θ) and cos(θ) are nonlinear functions of θ,

- ***x' is a linear combination of x and y***
- ***y' is a linear combination of x and y***

What is the inverse transformation?

- Rotation by –θ
- For rotation matrices $\mathbf{R}^{-1} = \mathbf{R}^{T}$

Translation



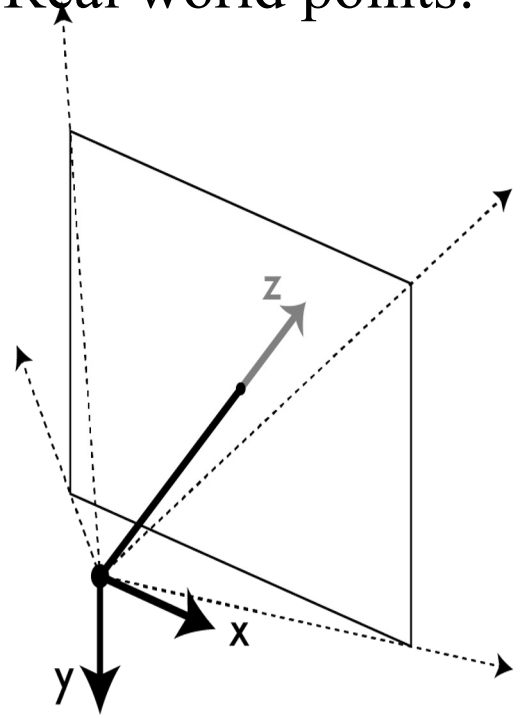
Homogenous coordinates

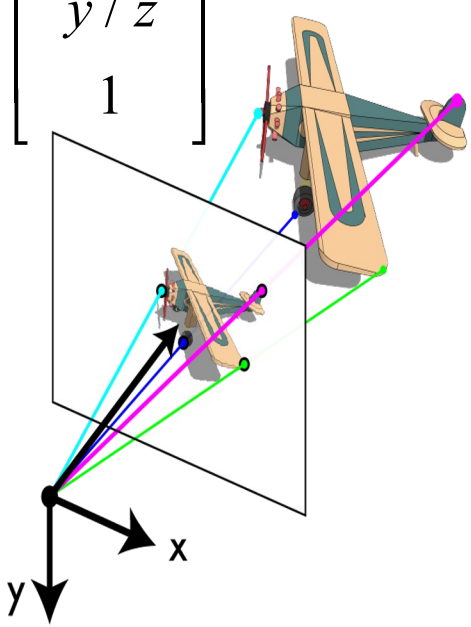
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

translation matrix T

Homogeneous coordinate system

- A 2D image is the projection of the 3D world
- Pinhole camera model: every point in 3D projects onto our viewing plane through our aperture. This means that each point in 2D is actually a vector (ray) in 3D
- Consider the image as the plane z = 1
- Real world points: $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ Image points: $\begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$





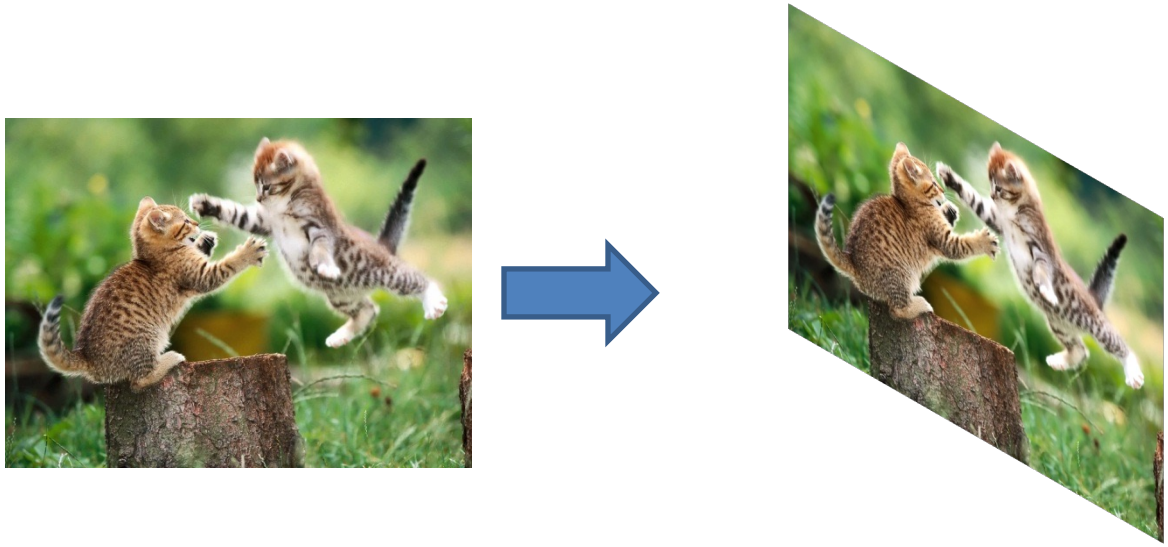
Homogeneous coordinate system

- A 2D image is the projection of the 3D world
- Pinhole camera model: every point in 3D projects onto our viewing plane through our aperture. This means that each point in 2D is actually a vector (ray) in 3D
- Consider the image as the plane z = 1
- Real world points: $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ Image points: $\begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$

(x, y, z) ⟷ (x/z, y/z)

Homogenous coordinates — Cartesian coordinates

Shear



$$\begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Basic 2D transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & \alpha_x \\ \alpha_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Shear

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

Rotate

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

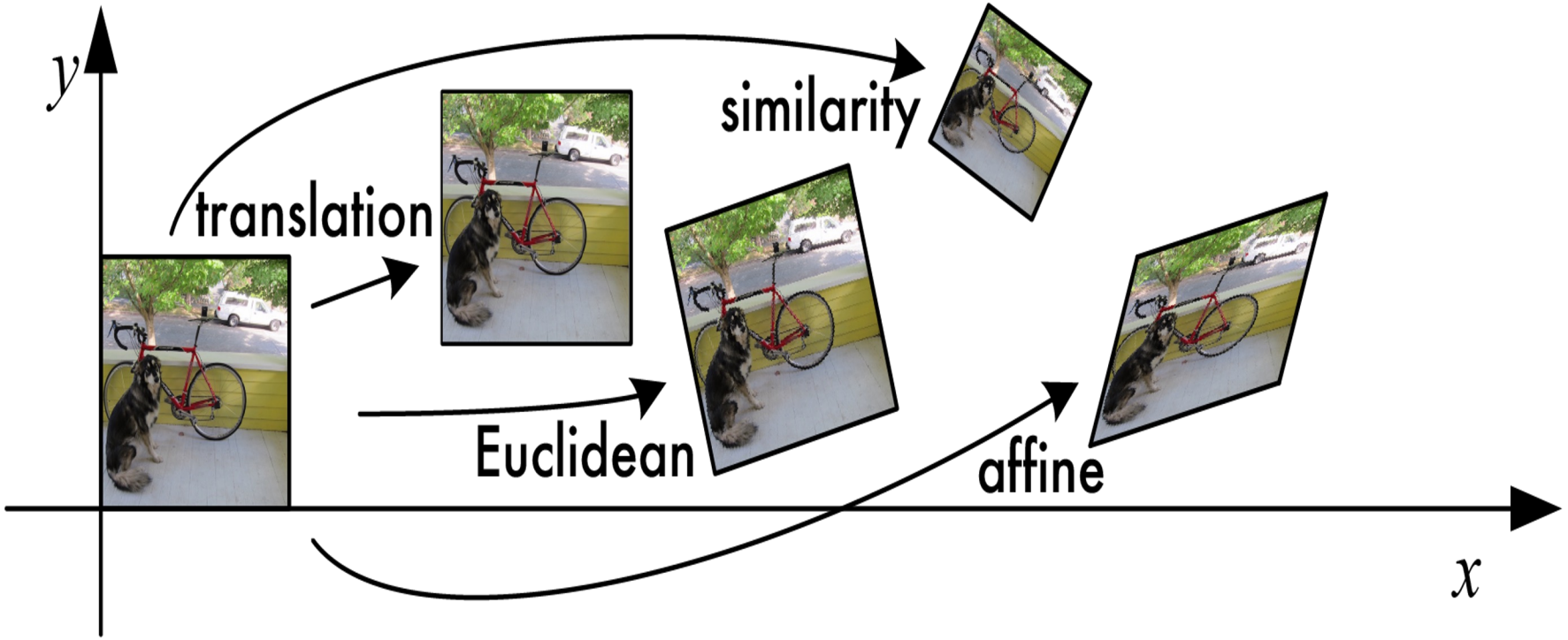
Translate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine

Affine is any combination of translation, scale, rotation, shear

Affine: scale, rotate, translate, shear



Affine Transformations

Affine transformations are combinations of

- Linear transformations, and
- Translations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

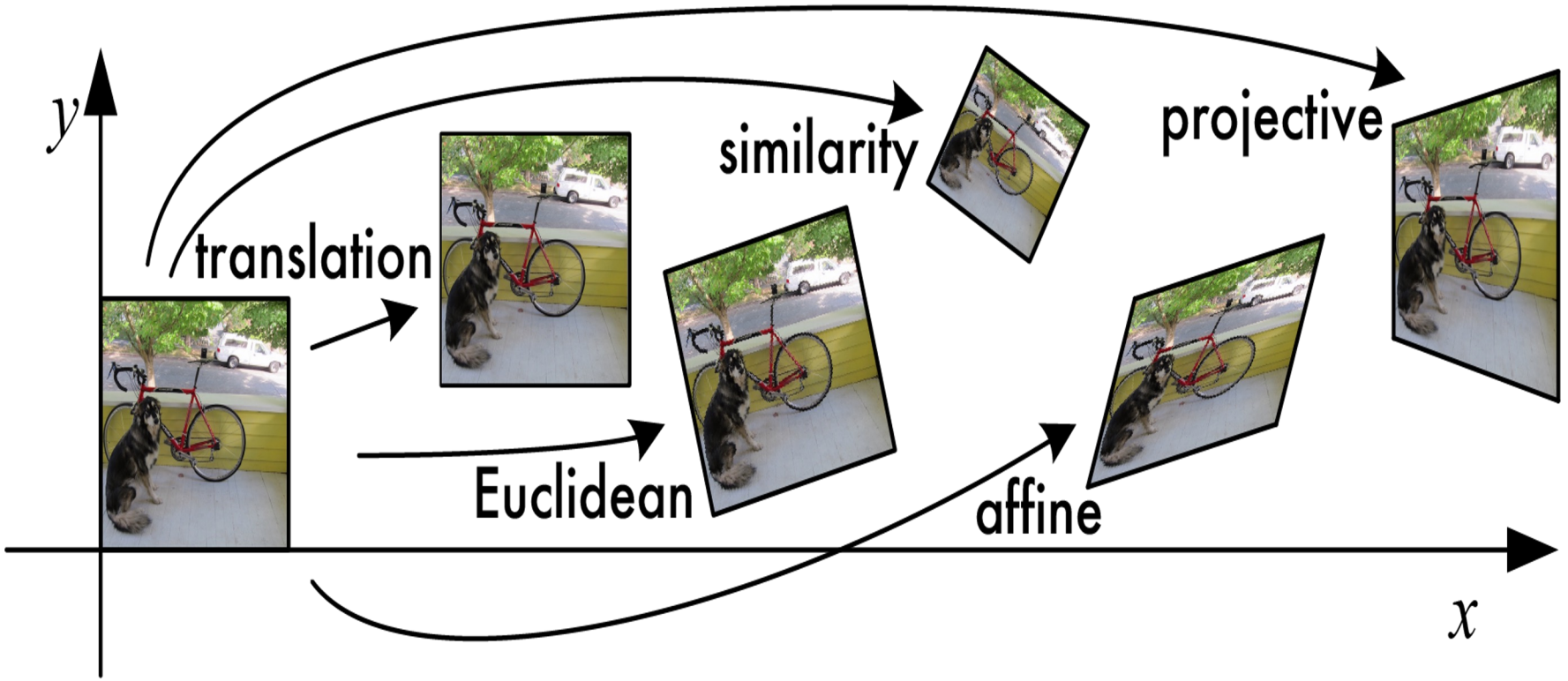
Properties of affine transformations:

- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition

Say you want to translate, then rotate, then translate back, then scale.

p' = S T R T **p** = M **p**, where M = (S T R T) is affine transformation, in order to do the multiplications need to write S, T, R as 3 × 3 matrices (add the row [0 0 1])

Projective Transformations



Projective Transformations

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Projective transformations are combos of

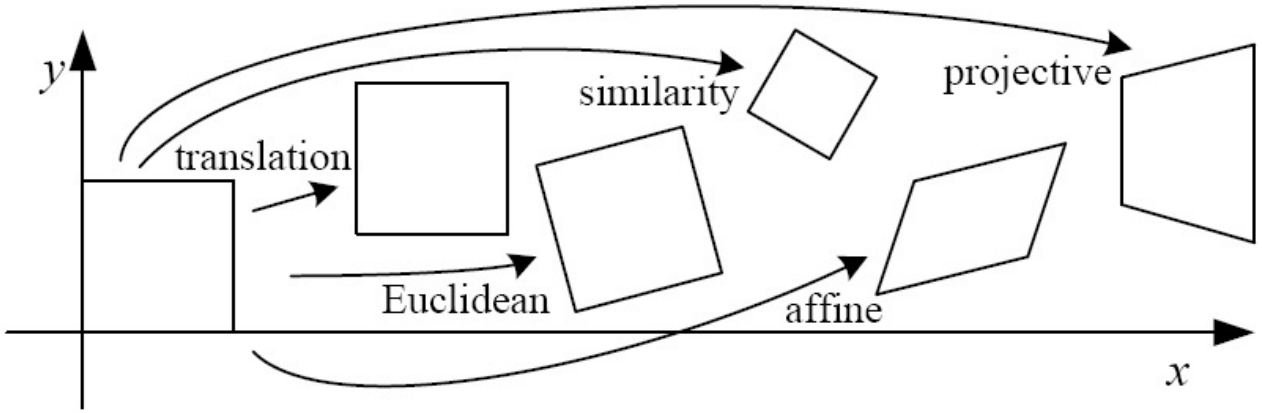
- Affine transformations, and
- Projective warps

Properties of projective transformations:

- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis
- Projective matrix is defined up to a scale (8 DOF)

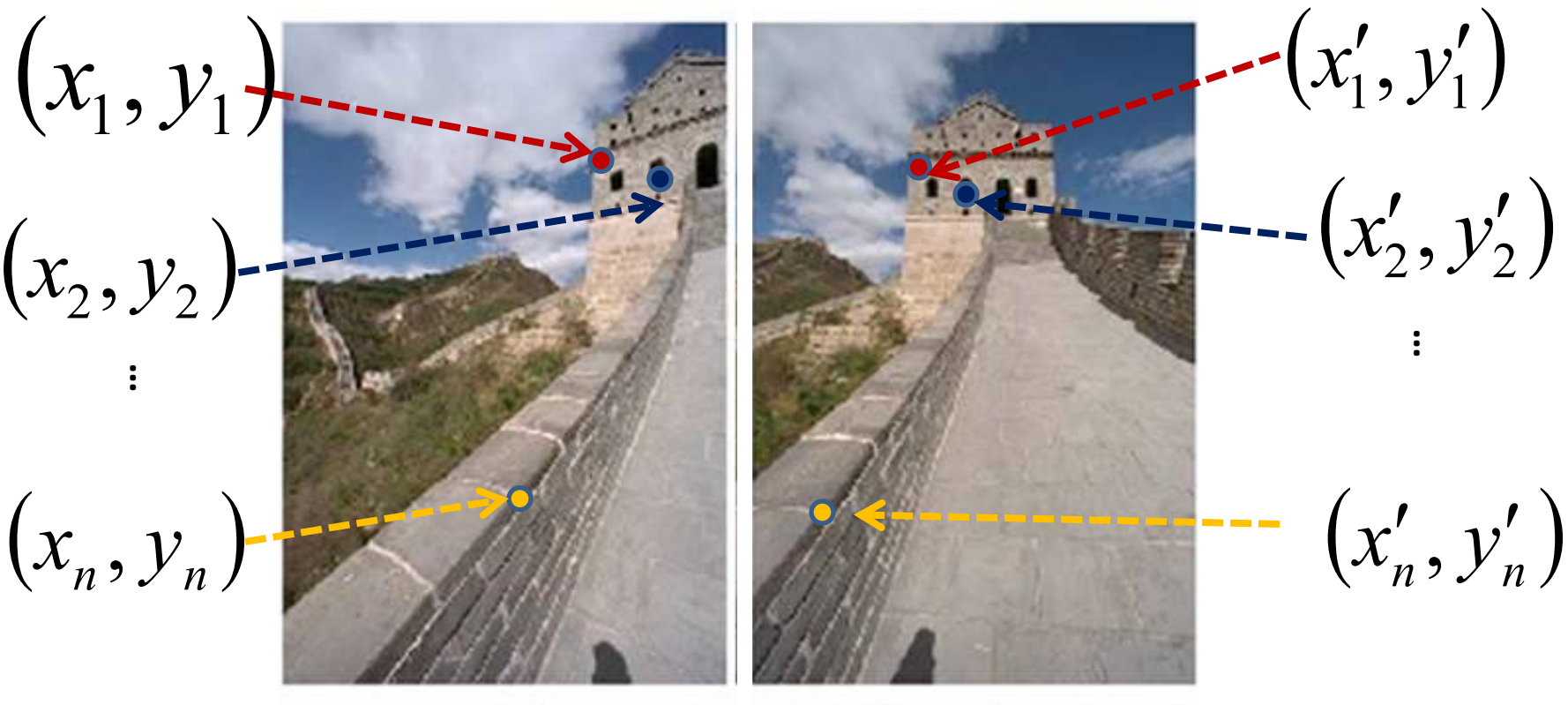
Also known as perspective transformations or homographies.

2D image transformations (reference table)



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	4	angles $+\cdots$	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times 3}$	8	straight lines	

Image matching using homographies



To **compute** the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of **H** are the unknowns…

Solving for homographies

$$\mathbf{p'} = \mathbf{Hp}$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Can set scale factor i=1. So, there are 8 unknowns.
- Set up a system of linear equations:

$$\mathbf{Ah} = \mathbf{b}$$

where vector of unknowns h = [a, b, c, d, e, f, g, h]^T

- Need at least 8 eqs, but the more the better…
- Solve for h. If overconstrained, solve using least-squares:

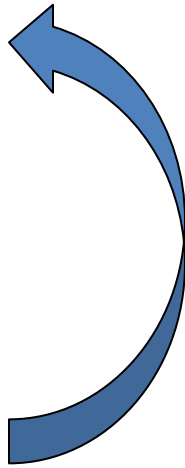
$$\min \|Ah - b\|^2$$

$$h = (A^T A)^{-1} A^T b$$

RANSAC for estimating homography

RANSAC loop:

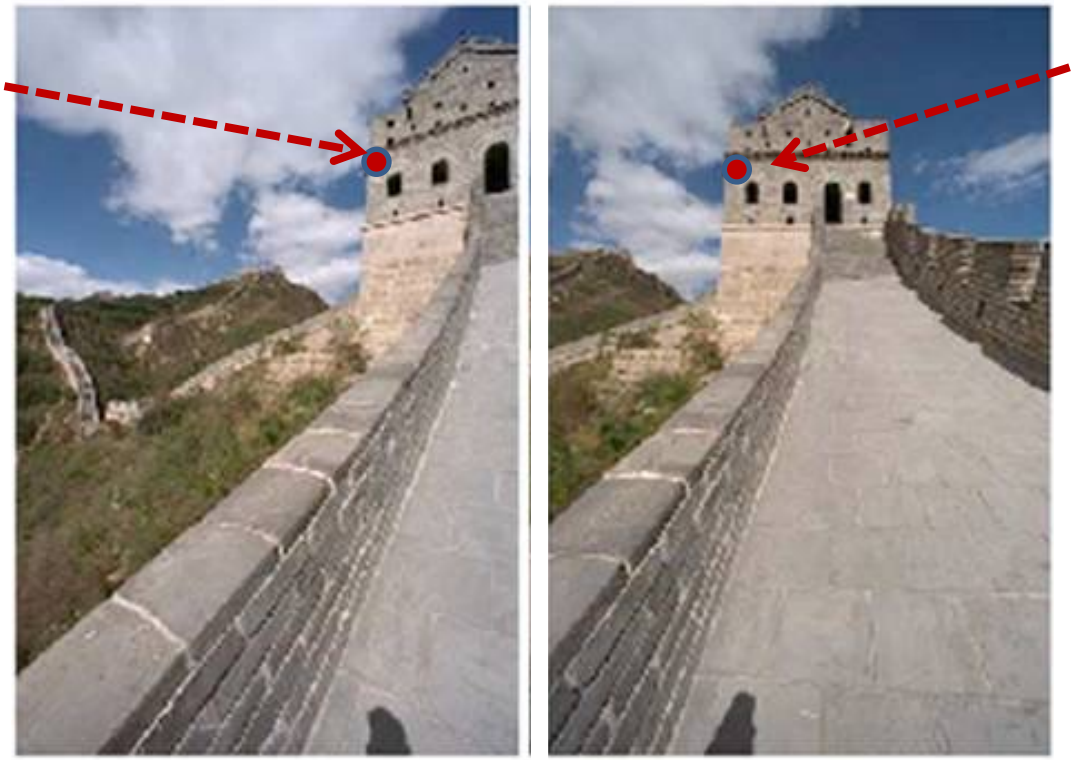
1. Select four feature pairs (at random)
2. Compute homography H
3. Compute *inliers* where $SSD(p_i', \mathbf{H}p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers



Slide credit: Steve Seitz

Homography

(x, y)



$\left(wx'/w, wy'/w\right) = (x', y')$

To **apply** a given homography **H**

- Compute **p'** = **Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

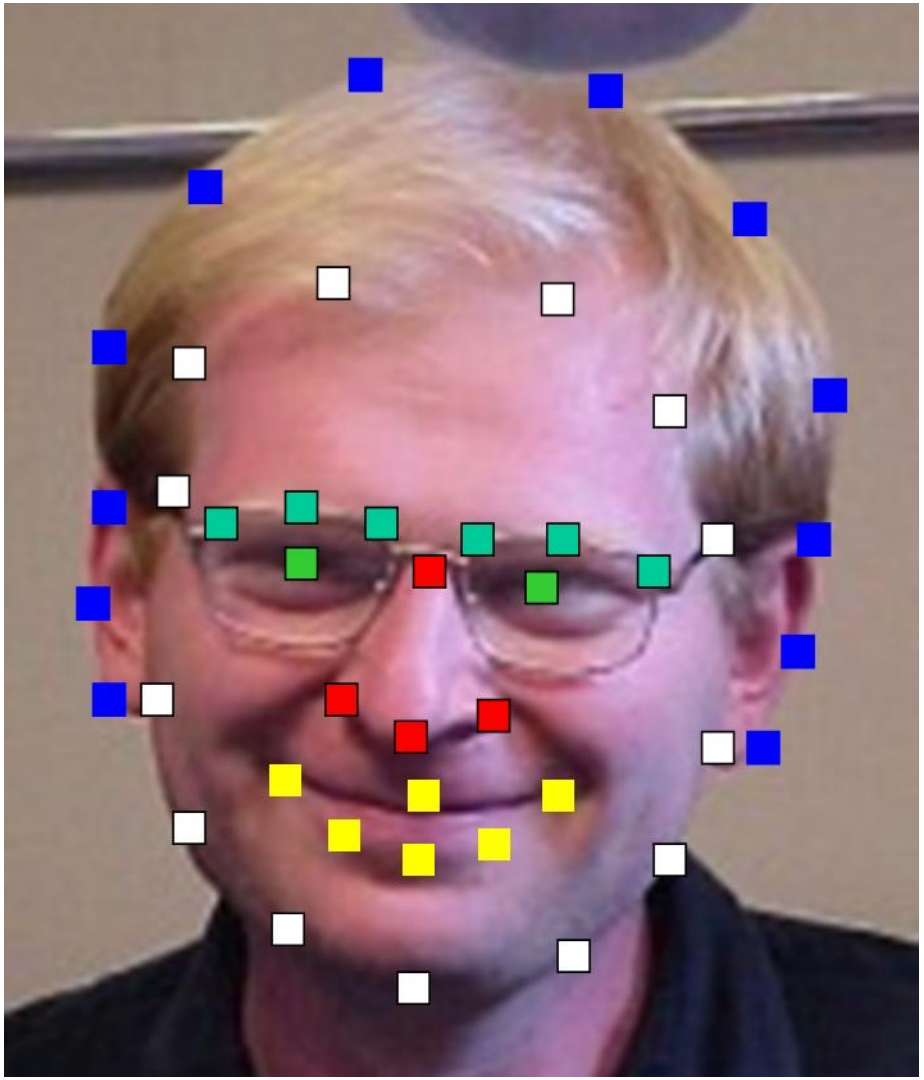
p' **H** **p**

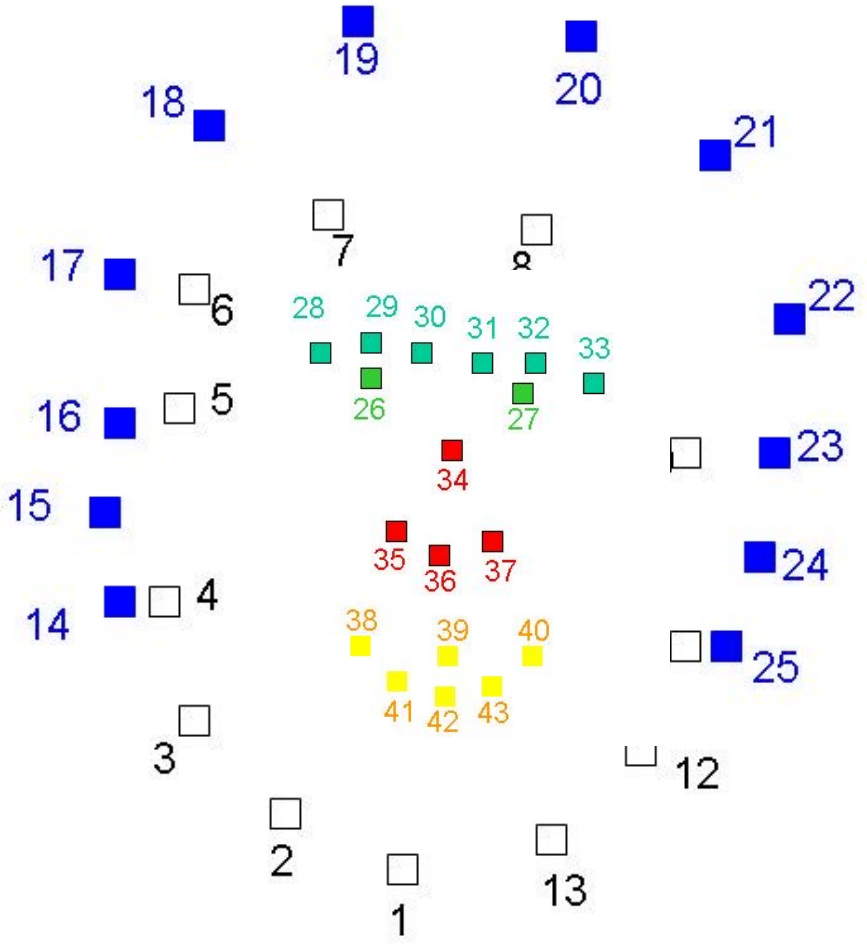
Application to affine transformation: image morphing

Application to affine transformation: image morphing

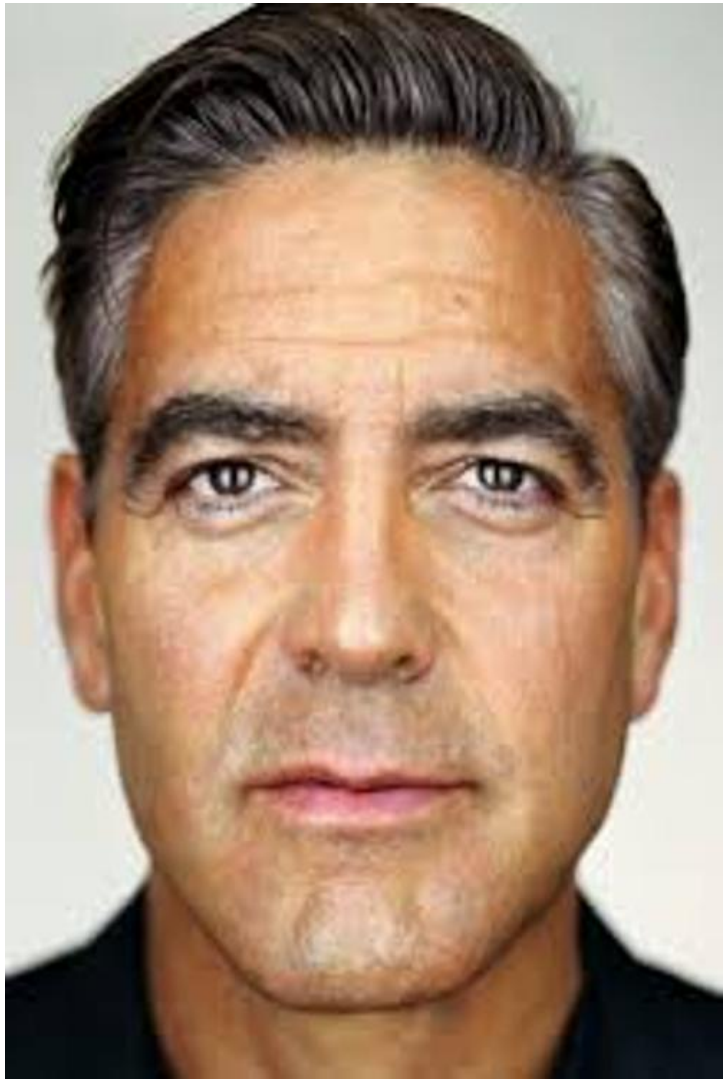
https://www.youtube.com/watch?v=XHb7Ig3yPgI

Facial landmarks: manual or automatic

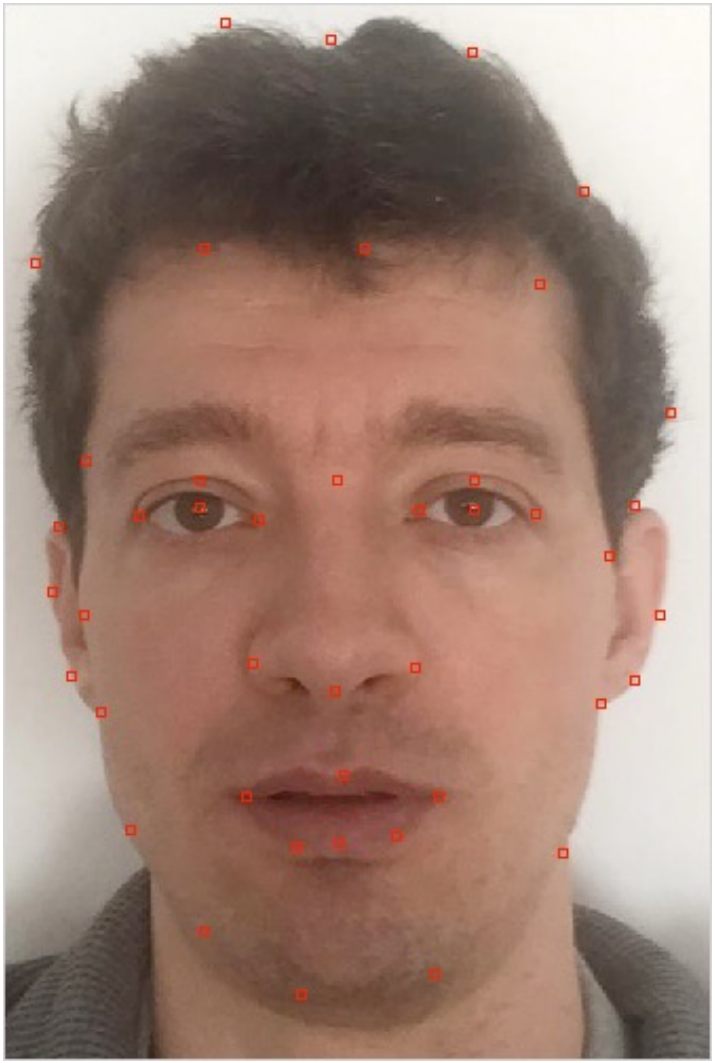




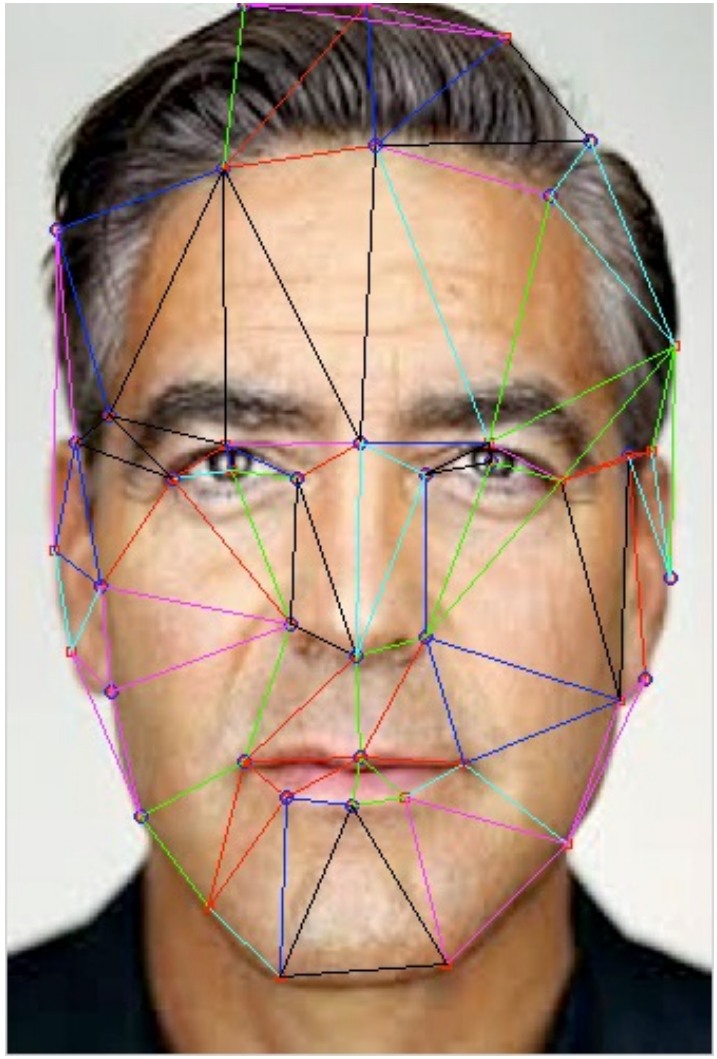
Facial landmarks: manual or automatic



Facial landmarks: manual or automatic



Delauney triangulation



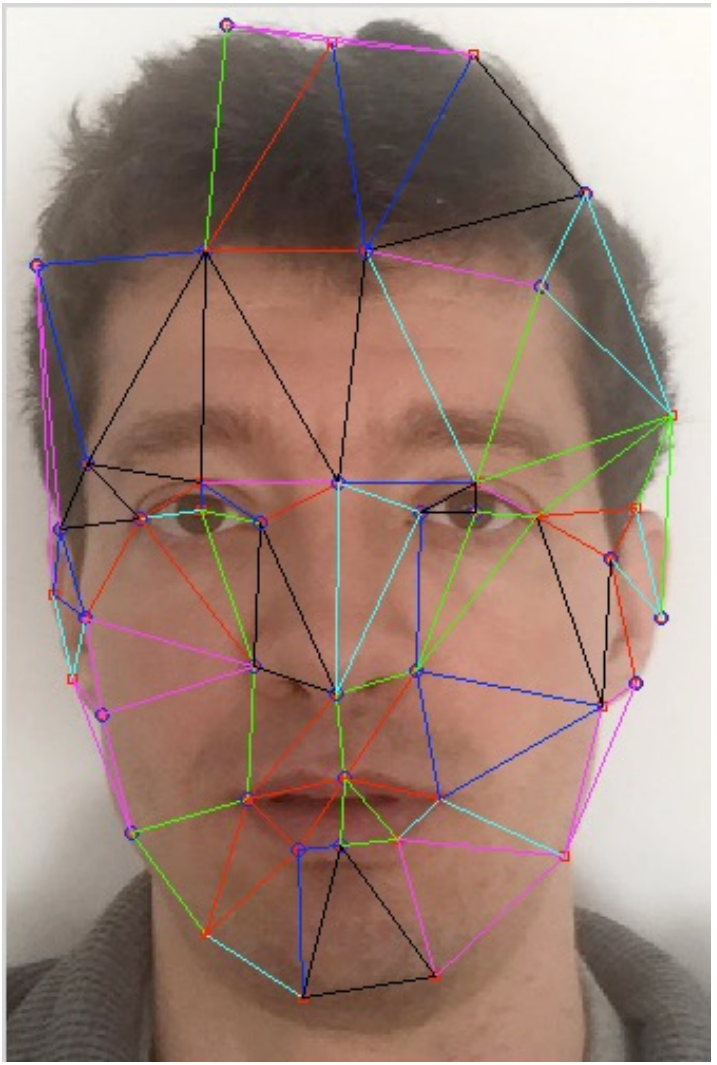
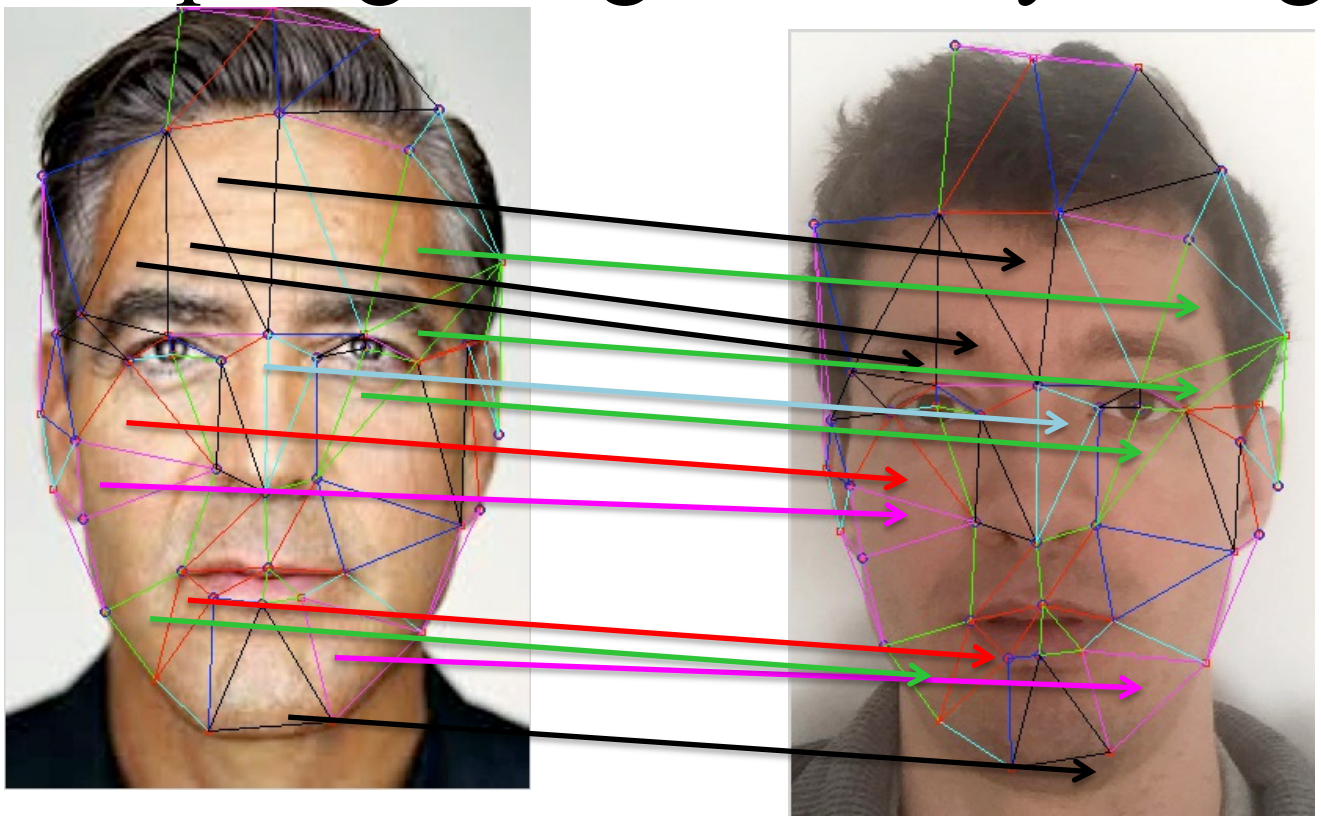


Image morphing using Delauney triangulation



- transform each triangle from the source image to the destination image by applying geometric and photometric transformation
- smooth transition by creating fake images: first image =source, last image = destination, in between = fake images

Transition between corresponding triangles

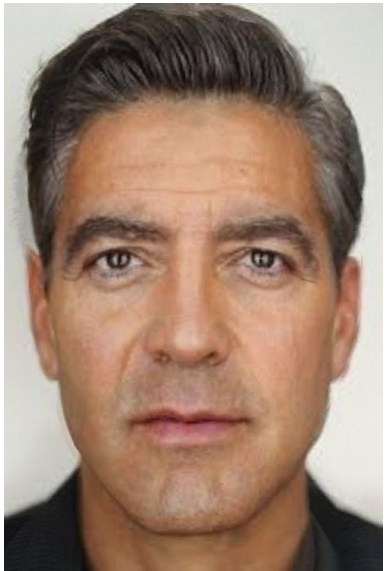
real

fake

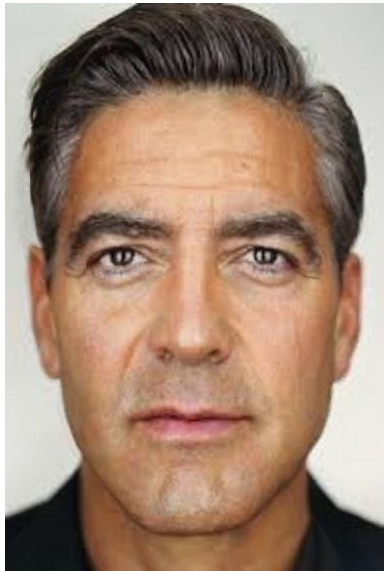
fake

fake

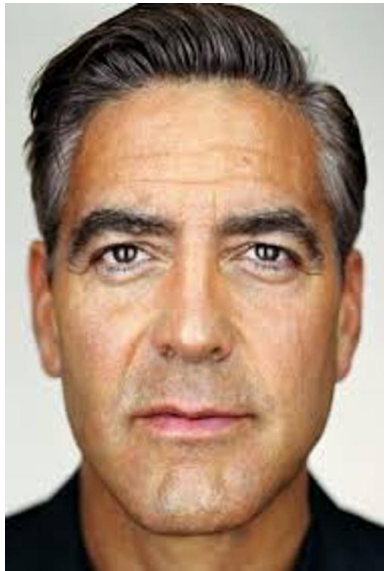
fake



fake

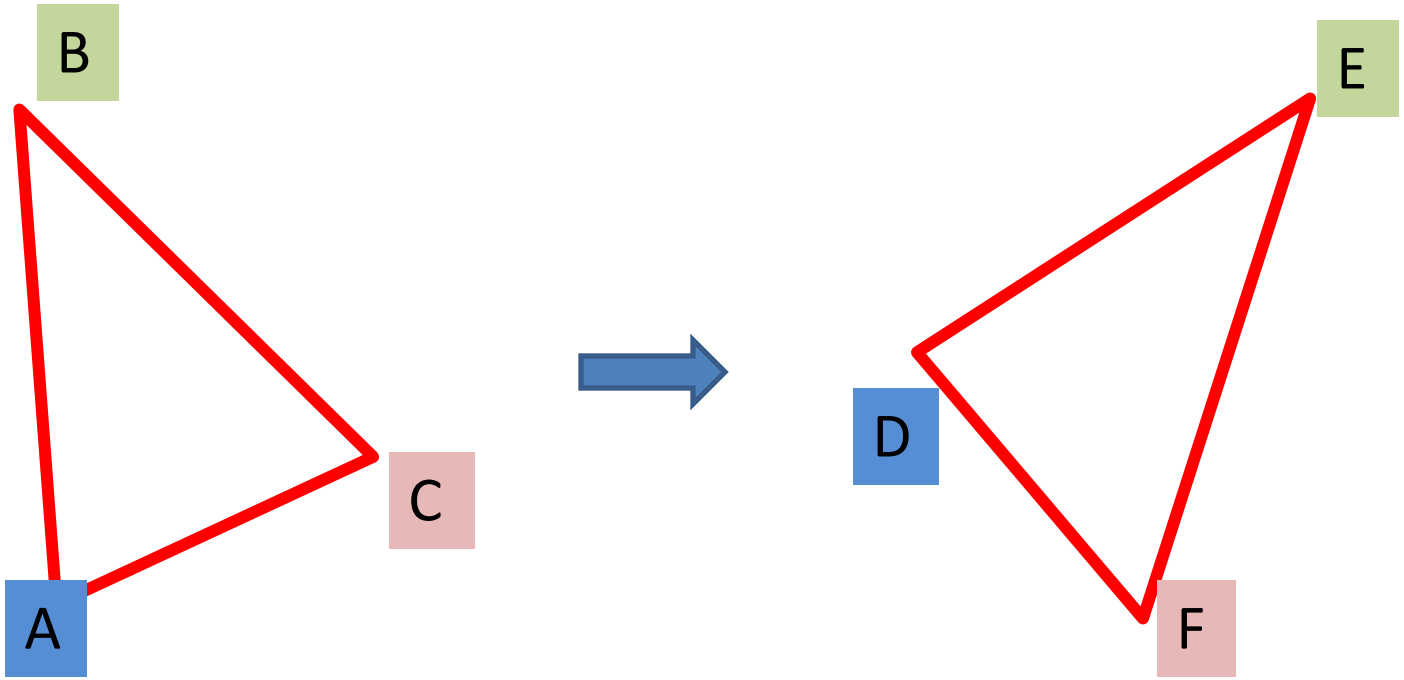


real



Affine transformation for triangles

For two triangles ABC and DEF I can find the affine transformation that it will map A in D, B in E and C in F by solving a system with 6 equations and 6 unknowns



Affine transformation for triangles

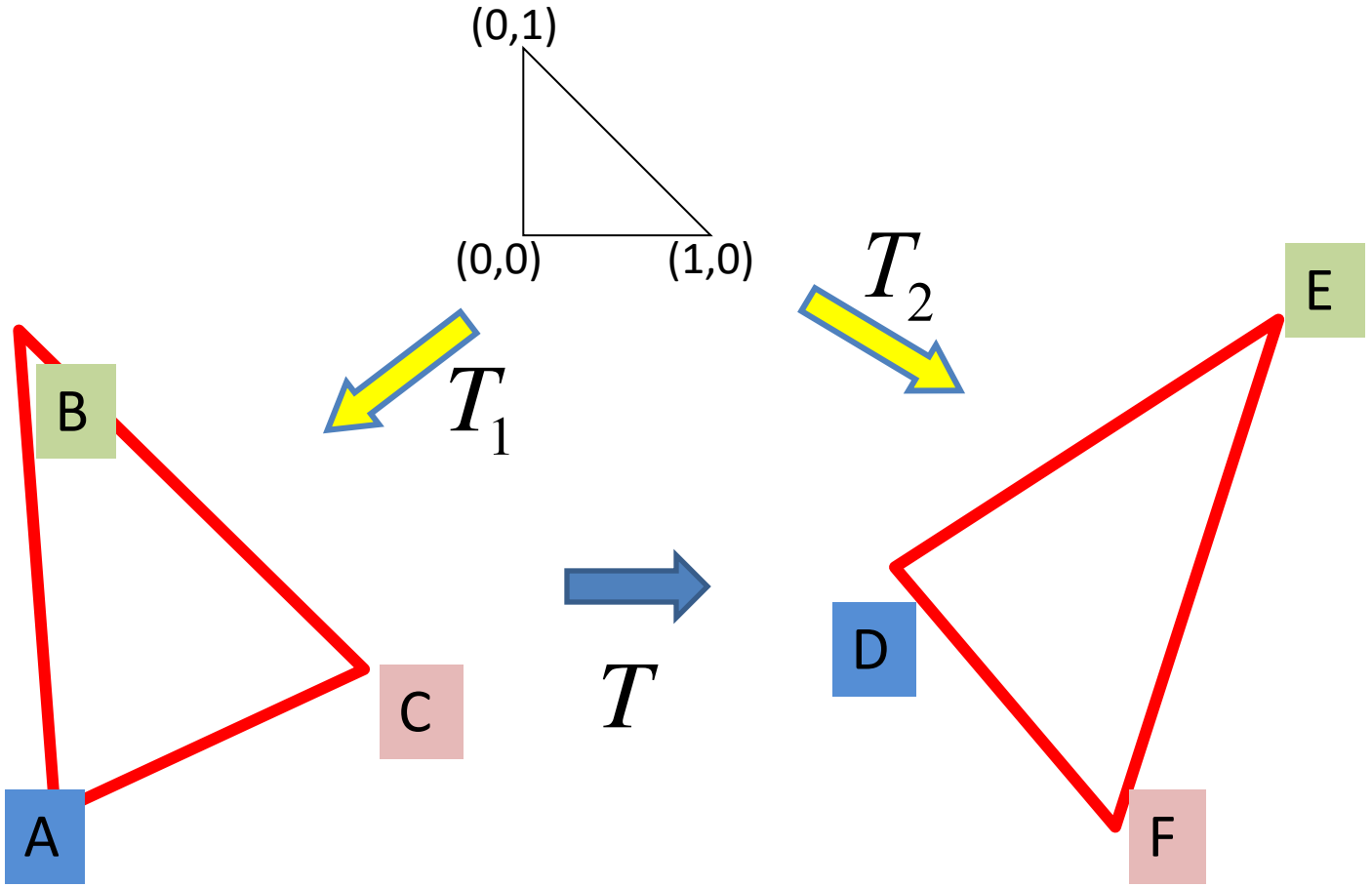
For two triangles ABC and DEF I can find the affine transformation that it will map A in D, B in E and C in F by solving a system with 6 equations and 6 unknowns

$$\begin{bmatrix} x_D \\ y_D \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x_F \\ y_F \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_E \\ y_E \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_B \\ y_B \\ 1 \end{bmatrix}$$

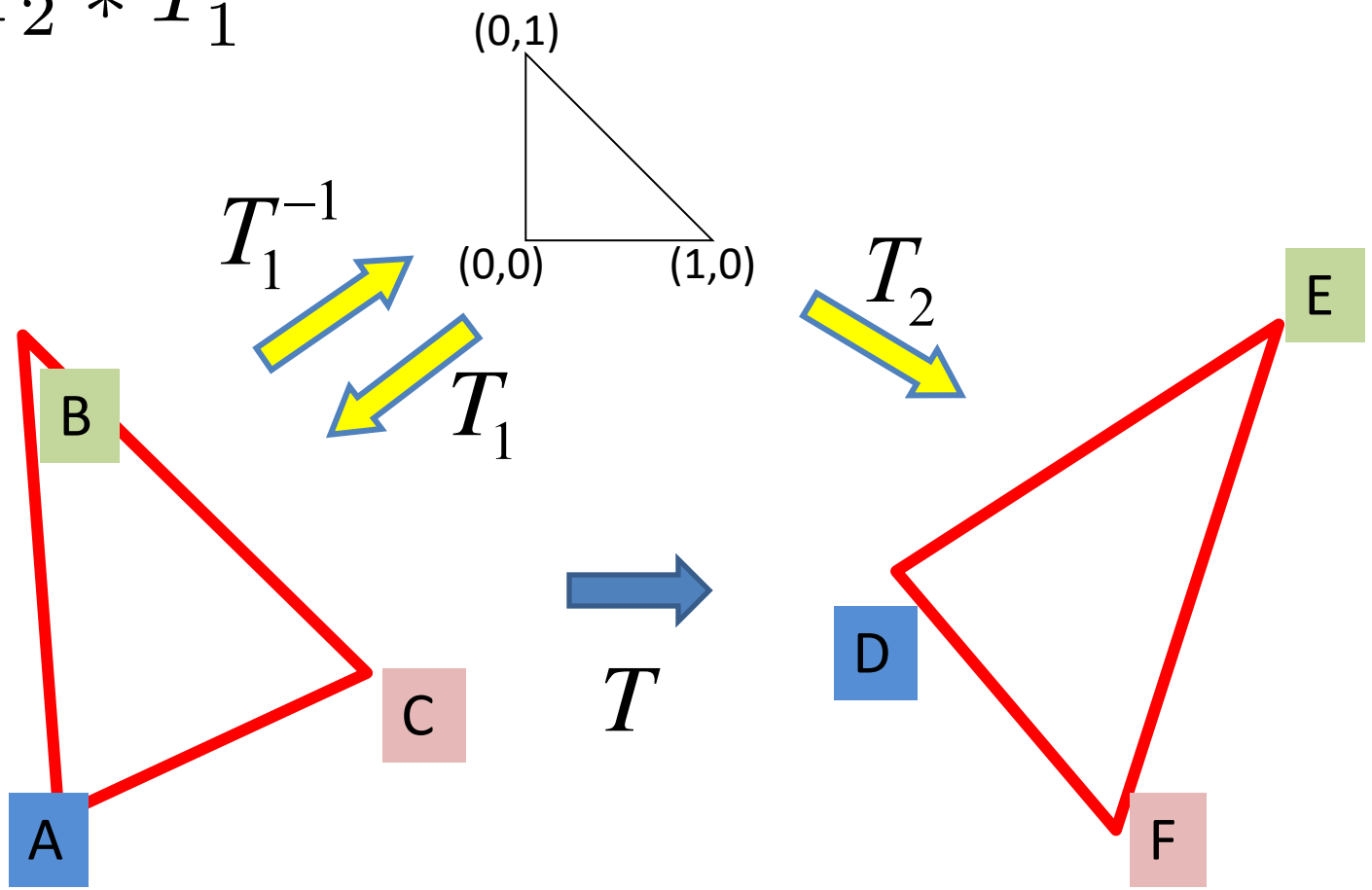
Equivalent approach: affine transformation composition

Use e “proxy” triangle with “nice” coordinates (very easy to solve the system with 6 equations and 6 unknowns) to get affine transformations T_1, T_2. Then find T.



Equivalent approach: affine transformation composition

$$T = T_2 * T_1^{-1}$$



Example: Computing T_2

Much nicer system with 6 equations and 6 unknowns:

$$\begin{bmatrix} x_D \\ y_D \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x_F \\ y_F \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

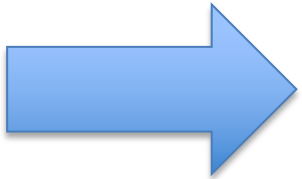
$$\begin{bmatrix} x_E \\ y_E \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Example: Computing T_2

Much nicer system with 6 equations and 6 unknowns:

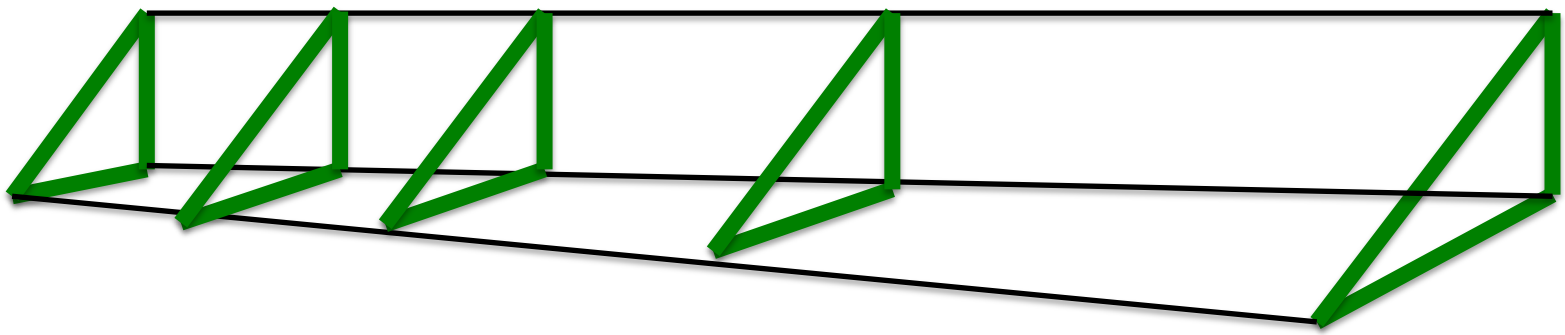
$$\begin{bmatrix} x_D \\ y_D \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x_F \\ y_F \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_E \\ y_E \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$



$$c = x_D$$
$$f = y_D$$
$$a = x_F - x_D$$
$$d = y_F - y_D$$
$$b = x_E - x_D$$
$$e = y_E - y_D$$

Geometric and photometric transition between triangles



Make a smooth transition, in time, from the source triangle T_0 to target triangle T_1. At step t:

- geometrically: $T_t = (1-t)*T_0 + t*T_1$ ("average weighting" of T_0 and T_1)
- photometrically: for each pixel (x,y) from triangle T_t, find the coresponding pixel (x_0,y_0) from T_0 and (x_1,y_1) from T_1 (using the affine transformations). Then compute:

  $RGB_{(x,y)} = (1-t)*RGB_{(x0,y0)} + t*RGB_{(x1,y1)}$ ("average weighting" of RGB colors from T_0 and T_1)

  -this is very slow ☹

  -do some trick here: warp T_0 to T, warp T_1 to T and than compute the average weighting by using a mask of T (work at the level of bounding boxes)

GIF animations