Computer Vision

Bogdan Alexe

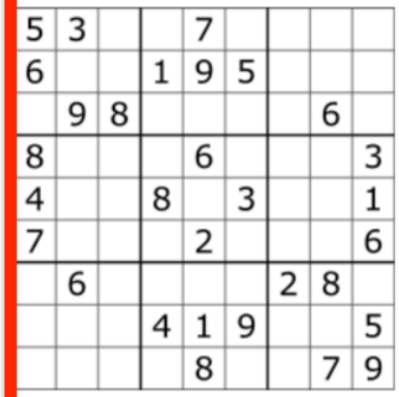
bogdan.alexe@fmi.unibuc.ro

University of Bucharest, 2nd semester, 2020-2021

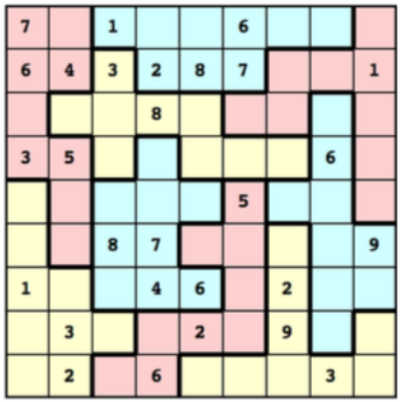
Project 1

Extracting visual information from Sudoku puzzles

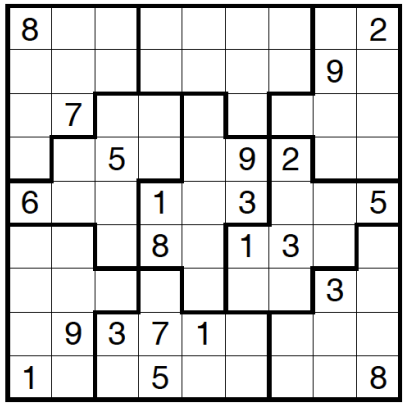
Many variants of Sudoku puzzles



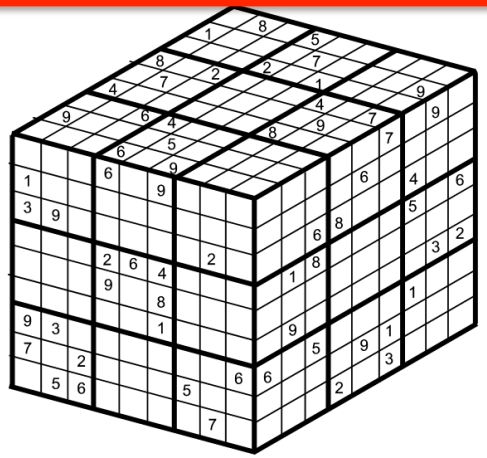
classic



jigsaw

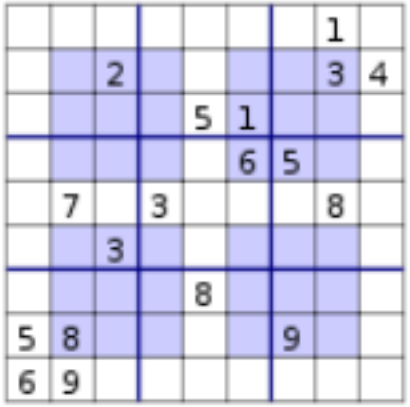


jigsaw

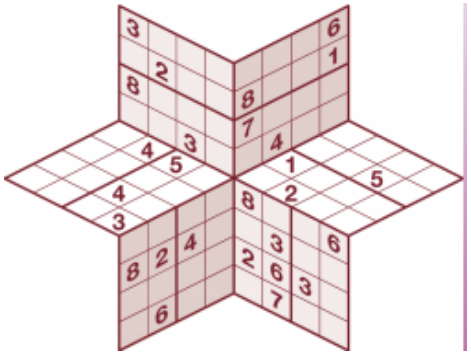


cube

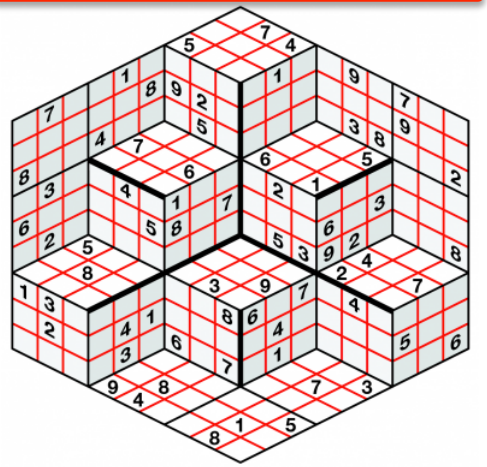
wordoku



hypersudoku



3D star



3D

- many more…

First task - Classic Sudoku

Task 1 - extracting configurations of Classic Sudoku puzzles

In the first task you are asked to write a program that processes an input image containing a Classic Sudoku puzzle and outputs the configuration of the puzzle by determining whether or not a cell contains a digit. We mark empty cells with letter 'o' and the filled in cells with letter 'x'. The training data consists of 50 training examples. Each training example (an image obtained by taking a photo with the mobile phone) contains one Classic Sudoku puzzle, centered, usually axis aligned or with small rotations with respect to the Ox and Oy axis. Figure 4 shows two training examples of Clasic Sudoku puzzles and their corresponding configurations.

```
OOXXOOOOX
OXOOOOOOX
XOOXOOXXO
XOXXOOXOO
OOOXOXOOO
OOXOOXXOX
OXXOOXOOX
XOOOOOOXO
XOOOOXXOO
```

```
OXOOXOOXO
XOOXOXOOX
OOXOOOXOO
OXOOXOOXO
XOXXOXXOX
OXOOXOOXO
OOXOOOXOO
XOOXOXOOX
OXOOXOOXO
```

Figure 4: *Examples of two Classic Sudoku puzzles and their corresponding configuration.*

Second task - Jigsaw Sudoku

Task 2 - extracting configurations of Jigsaw Sudoku puzzles

In the second task you are asked to write a program that processes an input image containing a Jigsaw Sudoku puzzle and outputs the configuration of the puzzle by: (1) determining the irregular shape regions in the puzzle; (2) determining whether or not a cell contains a digit. For this task, we mark all cells with a string of length two: the digit (1 to 9) corresponding to the irregular shape region where the cell is positioned and a letter ('o' or 'x') specifying whether or not the cell is empty. The irregular shape regions from the puzzle are separated by bold borders and sometimes (in the colored puzzles, see Figure 5) contain cells with the same color.

```
1o1o1o2o2o2o2x3x3o
1x1x1o2o3x3o3o3o3o
1o1o2o2o2o2x3o3o4o
1o4o4o4x4o4o4x4x4o
5o5o5o5o6o6o6o6o6x
5o5x7o6o6x8o8o6x6o
5o5x7o8x8x8o8x9o9o
5o7o7o7x8o8x9x9o9o
7o7x7o7o8o9x9o9o9o
```

```
1x1x2o2x2o3x3x3o3x
4o1o1x2o2x2o3x3o3o
4o4x1x1o2o2o2o3o3o
4o4o4o1o1o5x5o5o5o
6o4x4o7o1o8o8o5x5o
6o6o4o7x7o8o8o5o5o
6o6o9x9o7o7o8x8x5o
6o6o9o9o9x7o7x8o8o
6x6o9x9x9o9x7o7x8x
```

Figure 5: *Examples of two Jigsaw Sudoku puzzles and their corresponding configuration.*

Second task - Jigsaw Sudoku

cells with the same color. For determining the digit corresponding to a cell in an irregular shape regions in a Jigsaw puzzle we use the following simple algorithm: (i) we process the cells from left to right and top to bottom; (ii) the top left cell gets digit 1 as it is part of region 1; (iii) we assign the same digit for all cells in the same region; (iv) the first cell in the next region gets the increased digit (we move to the next region). The training data consists of 40 training examples (20 colored and 20 black and white Jigsaw Sudoku puzzles). Each training example (an image obtained by taking a photo with the mobile phone) contains one Jigsaw Sudoku puzzle, either colored or black and white, centered, usually axis aligned or with small rotations with respect to the Ox and Oy axis. A colored Jigsaw Sudoku puzzle will always contain regions of three possible colors: blue, yellow and red.

```
1o1o1o2o2o2o2x3x3o
1x1x1o2o3x3o3o3o3o
1o1o2o2o2o2x3o3o4o
1o4o4o4x4o4o4x4x4o
5o5o5o5o6o6o6o6o6x
5o5x7o6o6x8o8o6x6o
5o5x7o8x8x8o8x9o9o
5o7o7o7x8o8x9x9o9o
7o7x7o7o8o9x9o9o9o
```

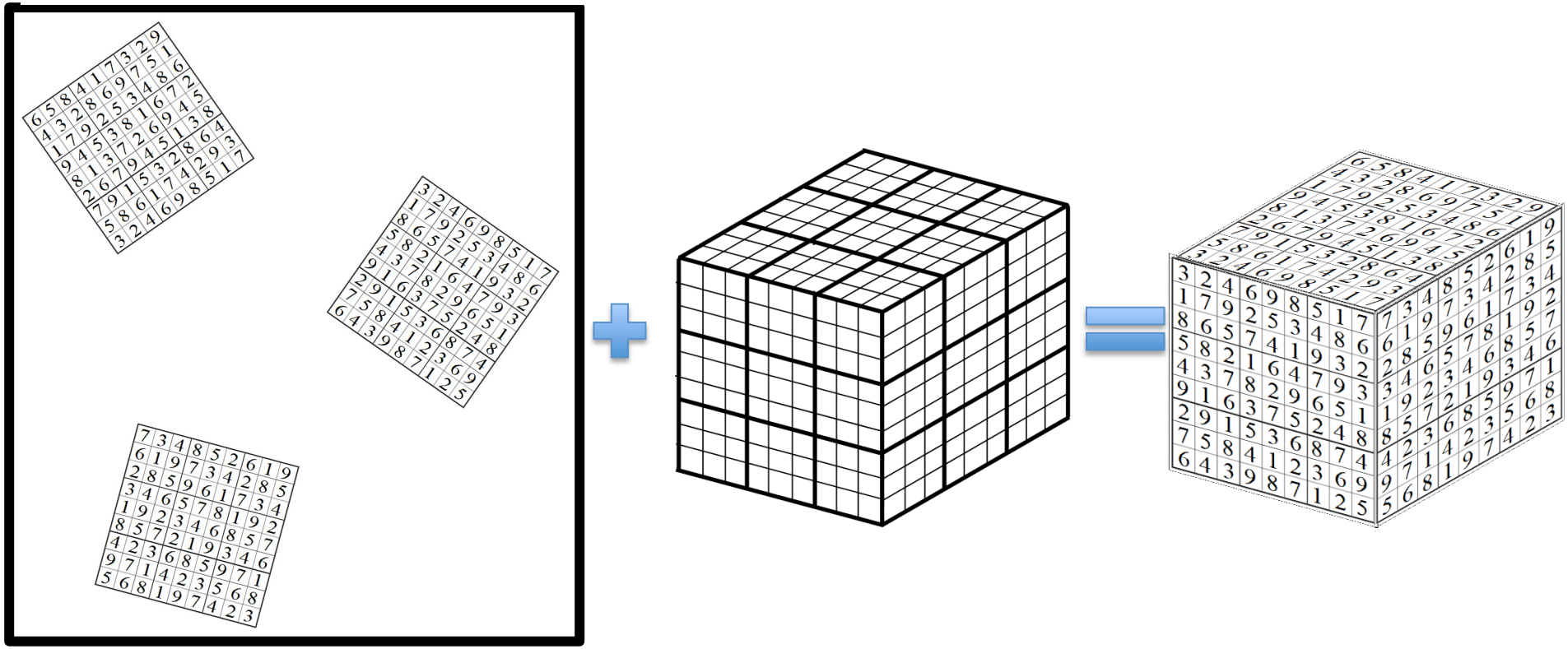
```
1x1x2o2x2o3x3x3o3x
4o1o1x2o2x2o3x3o3o
4o4x1x1o2o2o2o3o3o
4o4o4o1o1o5x5o5o5o
6o4x4o7o1o8o8o5x5o
6o6o4o7x7o8o8o5o5o
6o6o9x9o7o7o8x8x5o
6o6o9o9o9x7o7x8o8o
6x6o9x9x9o9x7o7x8x
```

Figure 5: *Examples of two Jigsaw Sudoku puzzles and their corresponding configuration.*

Third task – Sudoku Cube

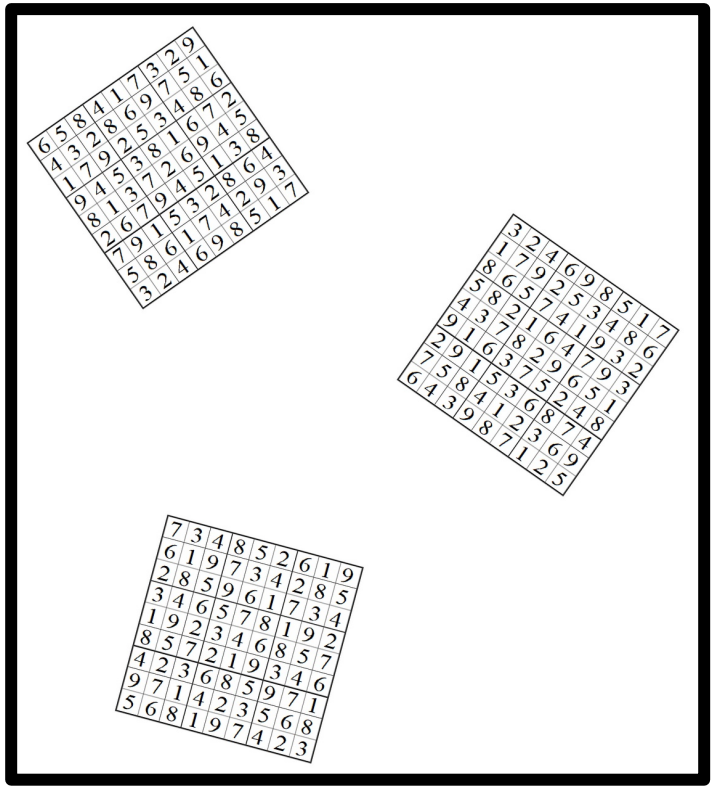
Task 3 - assembling a Sudoku Cube

In the third task you are asked to write a program that processes an input image containing three sides (each side is a sudoku puzzles) of a Sudoku Cube and outputs the coresponding Sudoku Cube by: (1) localizing the three sudoku puzzles in the image that form the sides of the Sudoku Cube; (2) inferring their position in the Sudoku Cube using the constraint that the digits on the common edge of two sides must be the same number; (3) warping each side on the corresponding side of a given template for the Sudoku Cube.



Third task – Sudoku Cube

data consists of 10 training examples. Each training example (an image obtained by taking a photo with the mobile phone) contains three sides of a Sudoku Cube (see Figure 7). They are scattered around the image and usually rotated with respect to the axis. First, you have to find the three sudoku puzzles in the image, recognize the digits in each puzzle and solve the simple problem of matching the sides in the Sudoku cube. Please note that there is one solution. Then you have to warp the puzzles on the template in order to obtain the desired

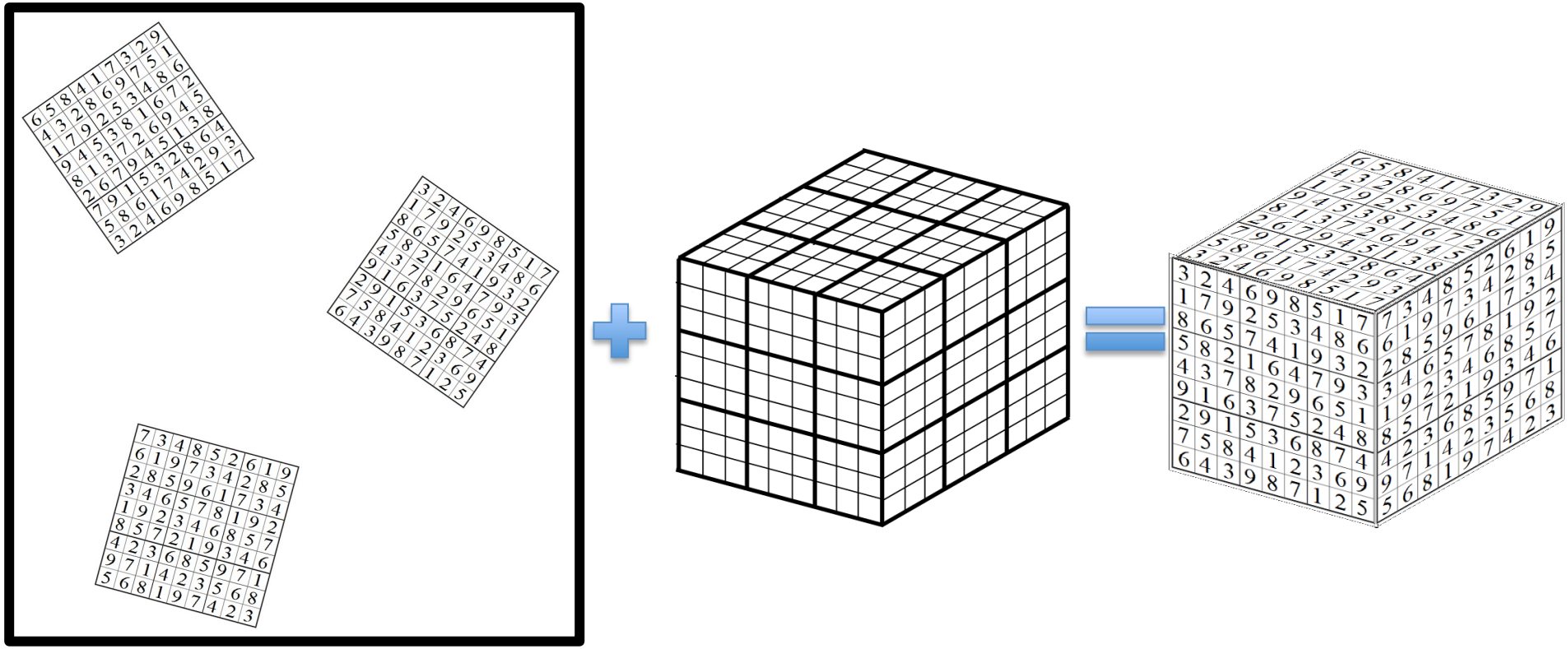


```
658417329
432869751
179253486
945381672
813726945
267945138
791532864
586174293
324498517

324498517 734852619
179253486 619734285
865741932 285961734
582164793 346578192
437829651 192346857
916375248 857219346
291536874 324685971
758412369 971423568
643987125 568197423
```

Third task – Sudoku Cube

solution. Then you have to warp the puzzles on the template in order to obtain the desired results. For warping, you are allowed to annotate points on the template for warping such that it is easy to map each puzzle found in the image on the corresponding side of the Cube. Figure 7 shows the input image and the corresponding configuration (listed as side one, two and three; we consider side one of the Cube to be the side seen from above, side two of the Cube to be the side seen from the front and side three of the Cube to be the side seen from right). Figure 6 shows the Sudoku Cube template and the desired output after warping.



Grading

The release data directory (available here https://tinyurl.com/CV-2021-Project1) contains two directories: *train* and *test*. The two directories have the same structure, although the *test* data will be made available after the deadline.

Requirements

Your job is to write a program in Python that automatically solves Task 1, Task 2 and Task 3. This project worth 5 points but you can gain a bonus for a total of 6 points. We will grade your project based on the performance achieved by your algorithms on each of the three tasks. The grading scheme is as follows:

- **Task 1** - you receive a test set containing 50 testing examples with Classic Sudoku puzzles. For each test image you have to output the corresponding configuration. Each correctly labeled configuration will worth 0.05 points for a total of **2.5 points**;
- **Task 2** - you receive a test set containing 40 testing examples with Jigsaw Sudoku puzzles (20 colored and 20 black and white). For each test image you have to output the corresponding configuration. Each correctly labeled configuration will worth 0.05 points for a total of **2 points**;
- **Task 3** - you receive a test set containing 10 testing examples with Sudoku Cube puzzles. For each test image you have to output the corresponding configuration (see Figure 7right) and the desired output (see Figure 6right). Each correctly labeled configuration will worth 0.05 points and each correct warped image will worth 0.05 points for a total of **1 point**;
- ex officio **0.5 points**;

Deadlines & Submissions

Deadlines: Submit a zip archive containing your code and a pdf file describing your approach until Saturday, 8^{th} of May using the following link https://tinyurl.com/CV-2021-PROJECT1-SUBMISSIONS. 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 for each task and where to look for the output file. On Sunday 9^{th} of May we will make available the test images. You will have to run your system 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-2021-PROJECT1-RESULTS.

Lecture 9

Course structure

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

   Linear filters, color, texture, edge detection

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**

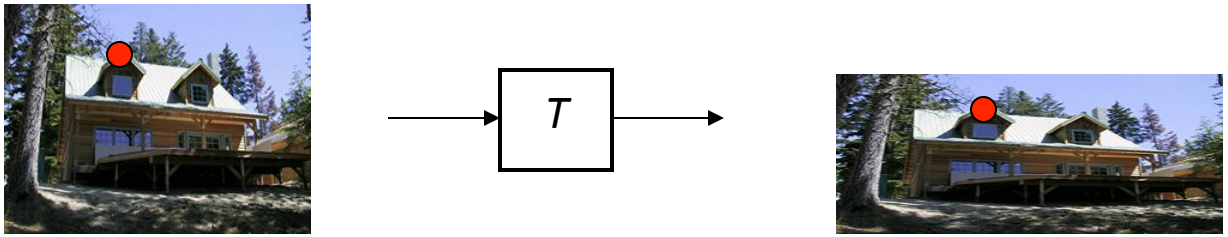
   Object classification, object detection, part based models, bovw models

5. **Video understanding**

   Object tracking, background subtraction, motion descriptors, optical flow

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

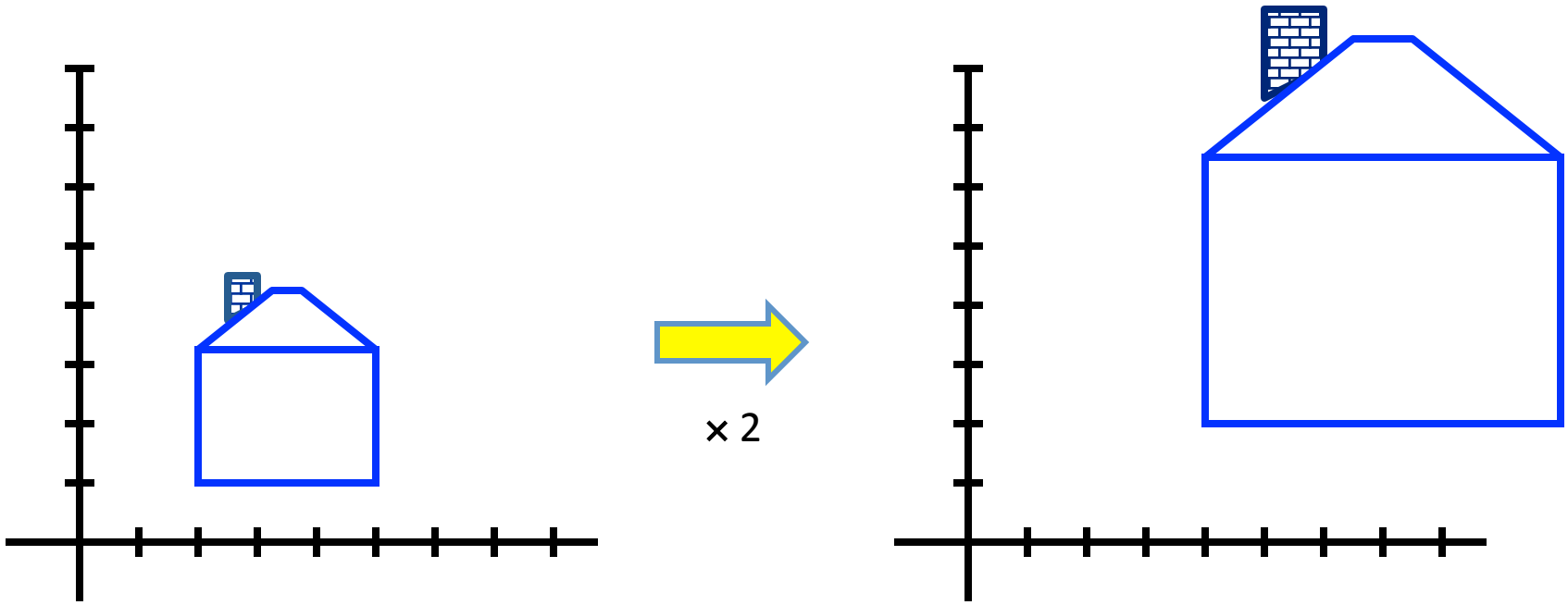
affine

perspective

Slide credit A. Efros and/or S. Seitz

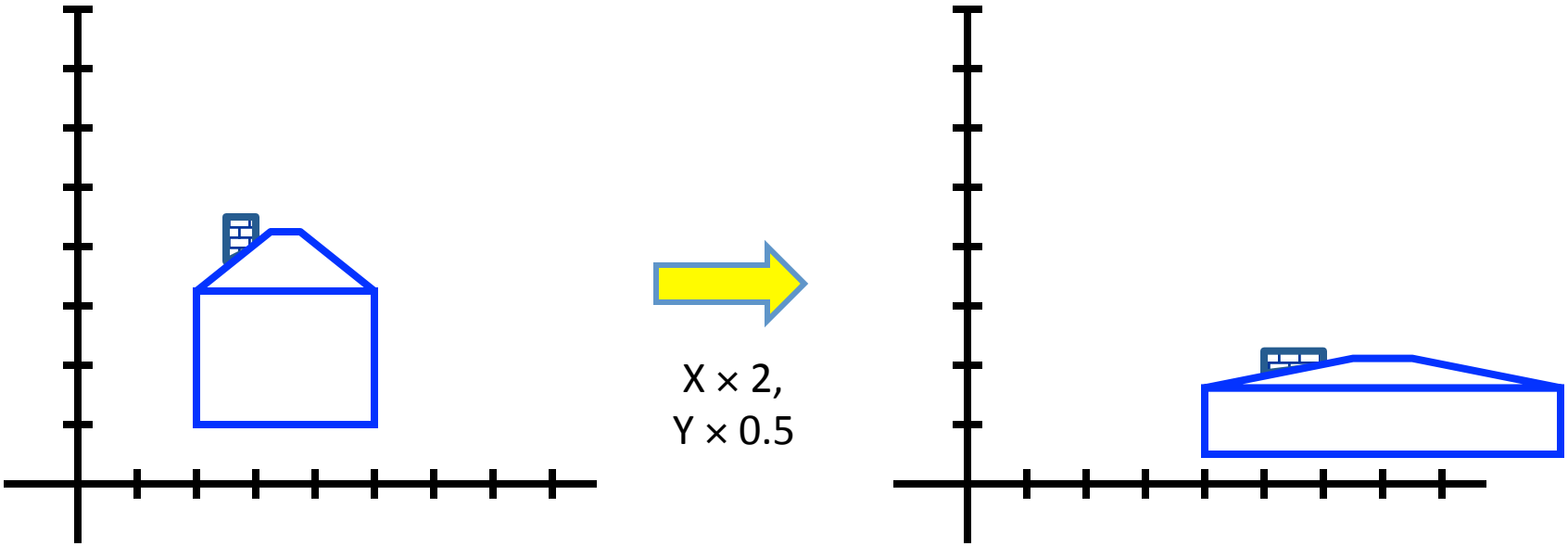
Scaling

- *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:

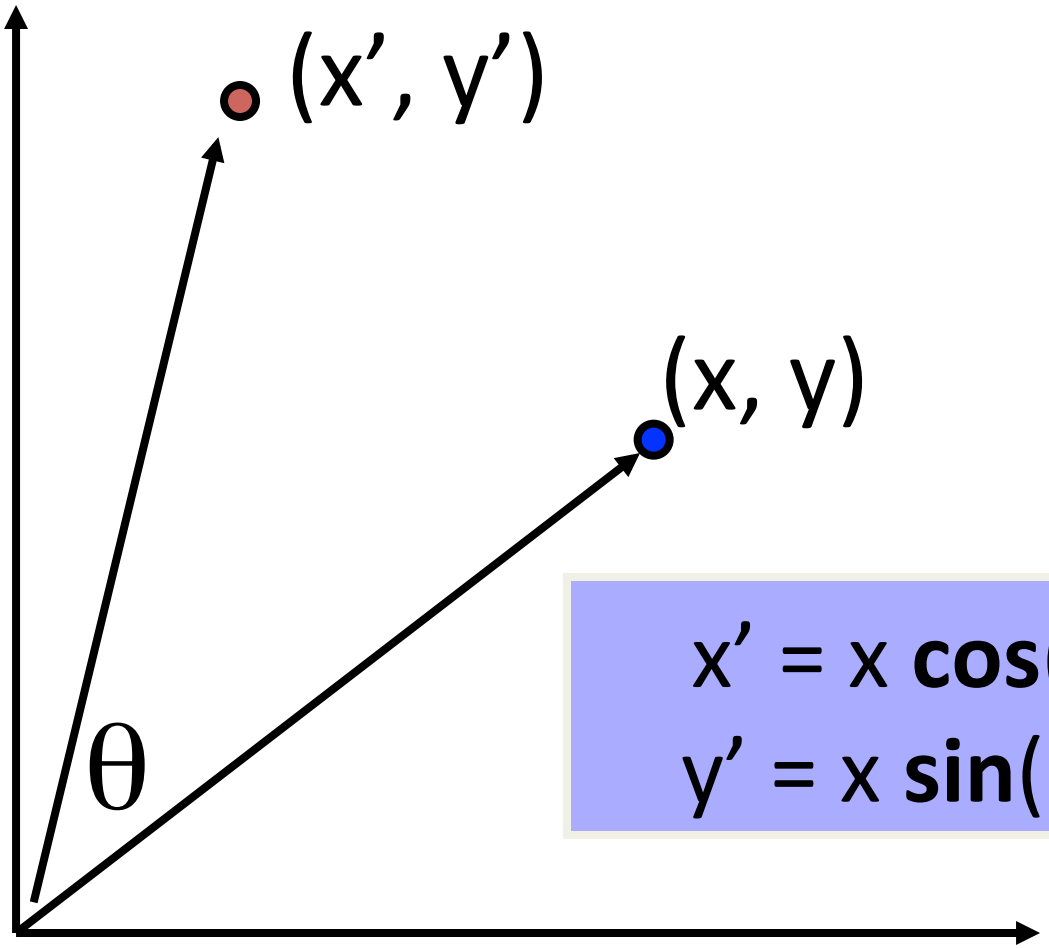
$$x' = ax$$

$$y' = by$$

- Or, in matrix form:

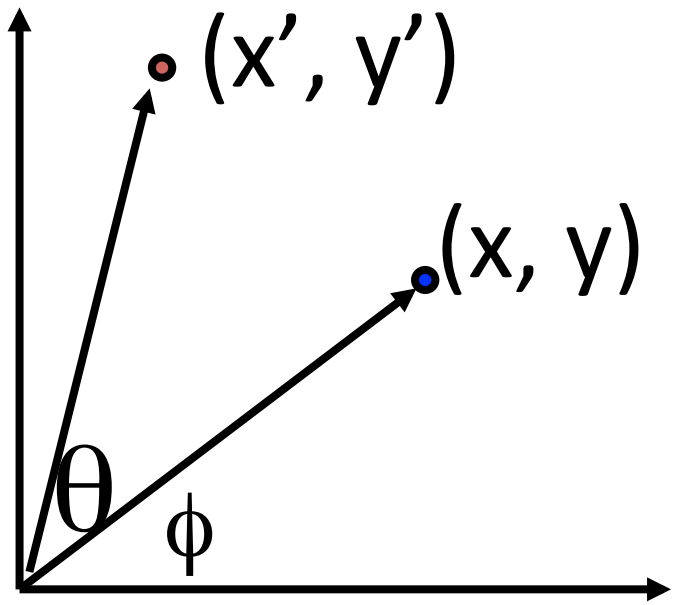
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

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:

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

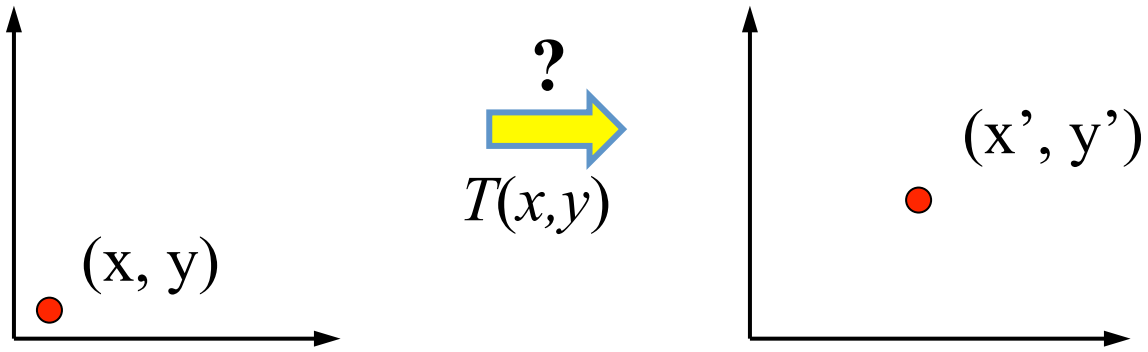
Even though sin(θ) and cos(θ) are nonlinear functions of θ,

- ***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



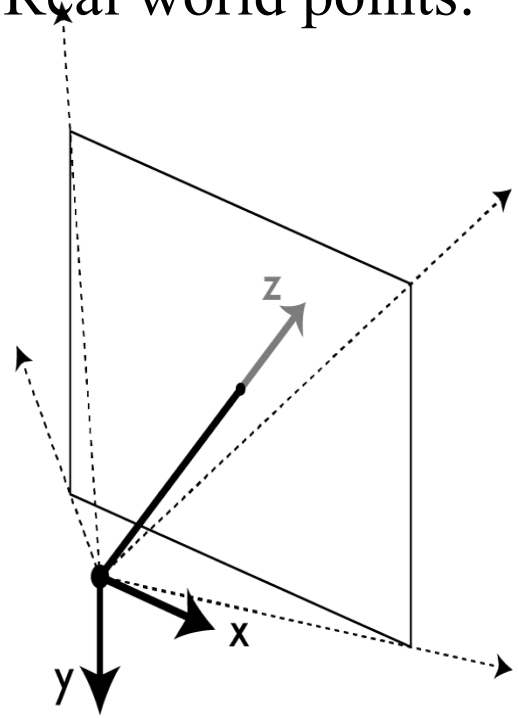
Homogenous coordinates

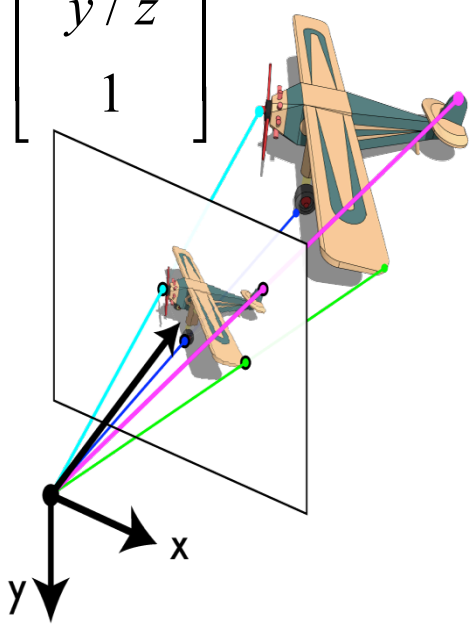
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

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

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

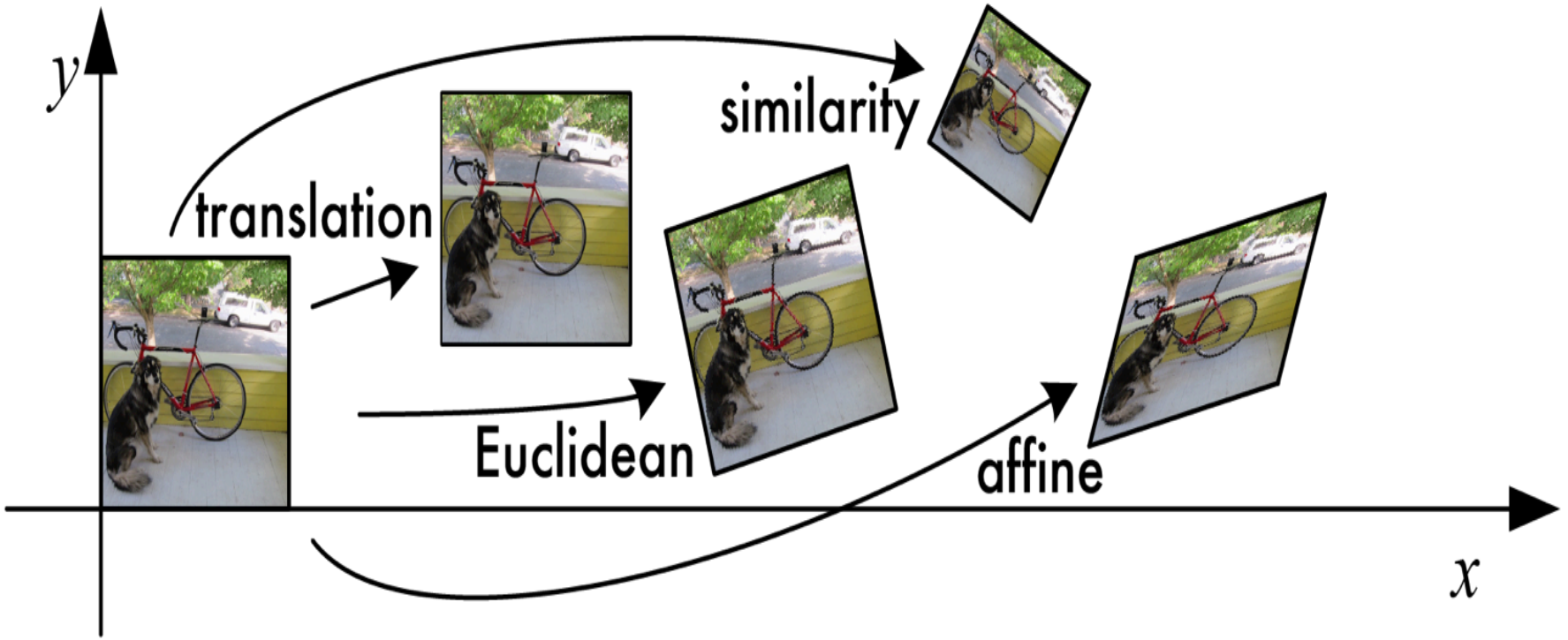
Translate

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

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

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

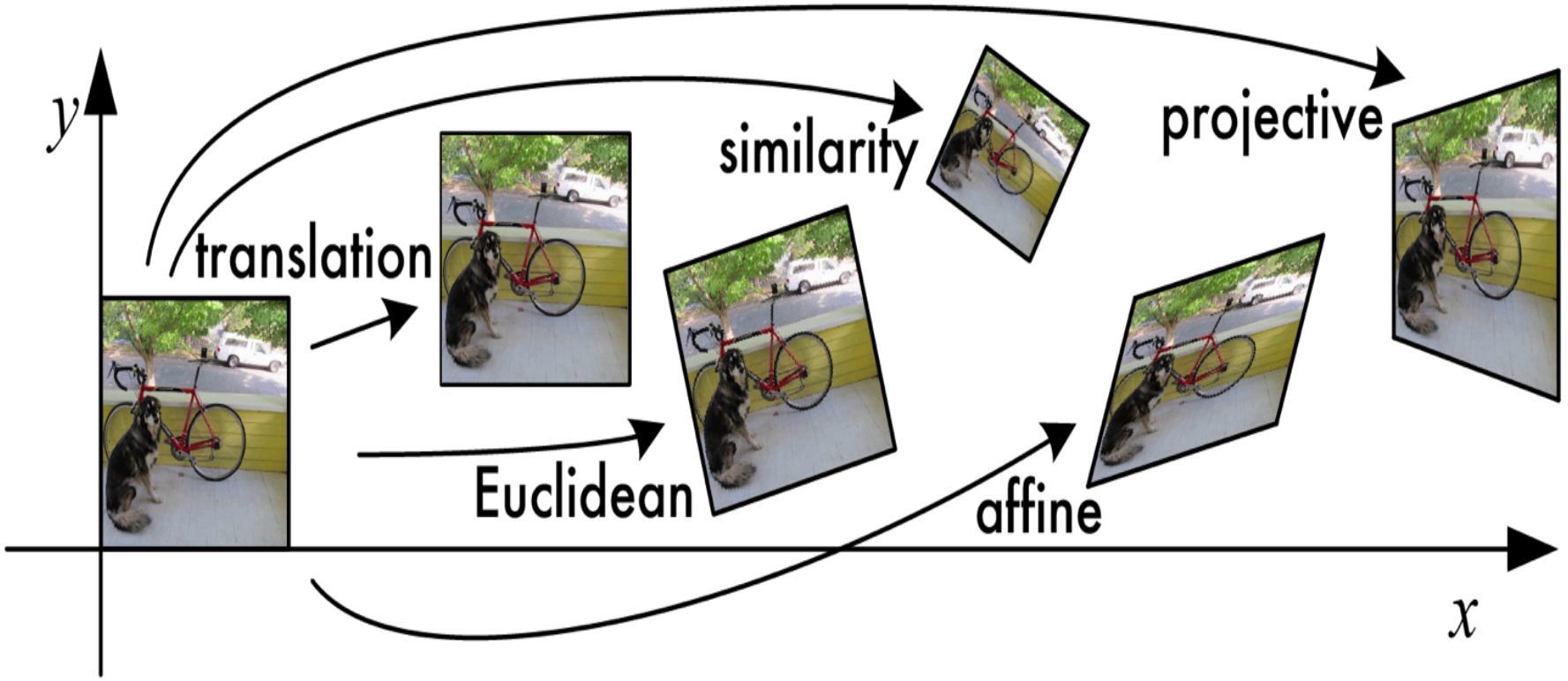
Properties of affine transformations:

- 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

Projective transformations are combos of

- Affine transformations, and
- Projective warps

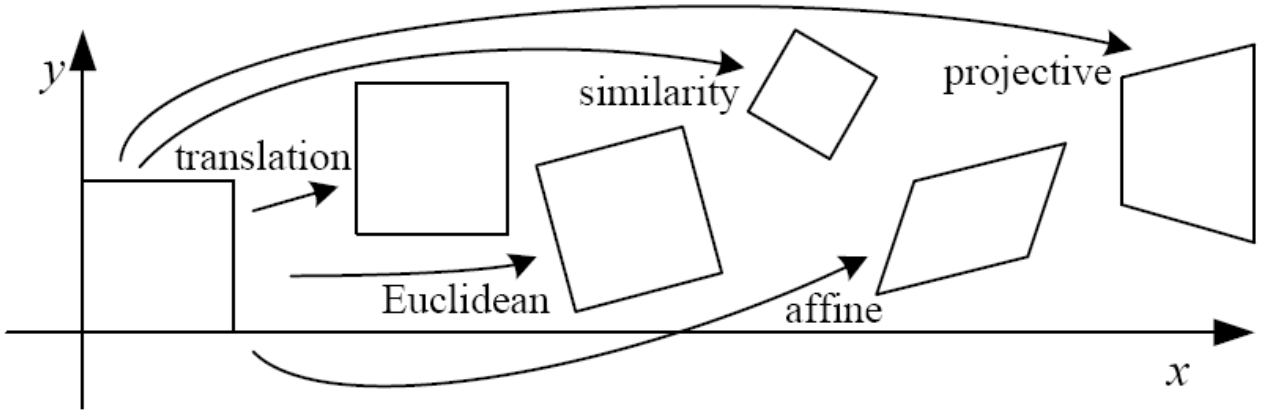
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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)

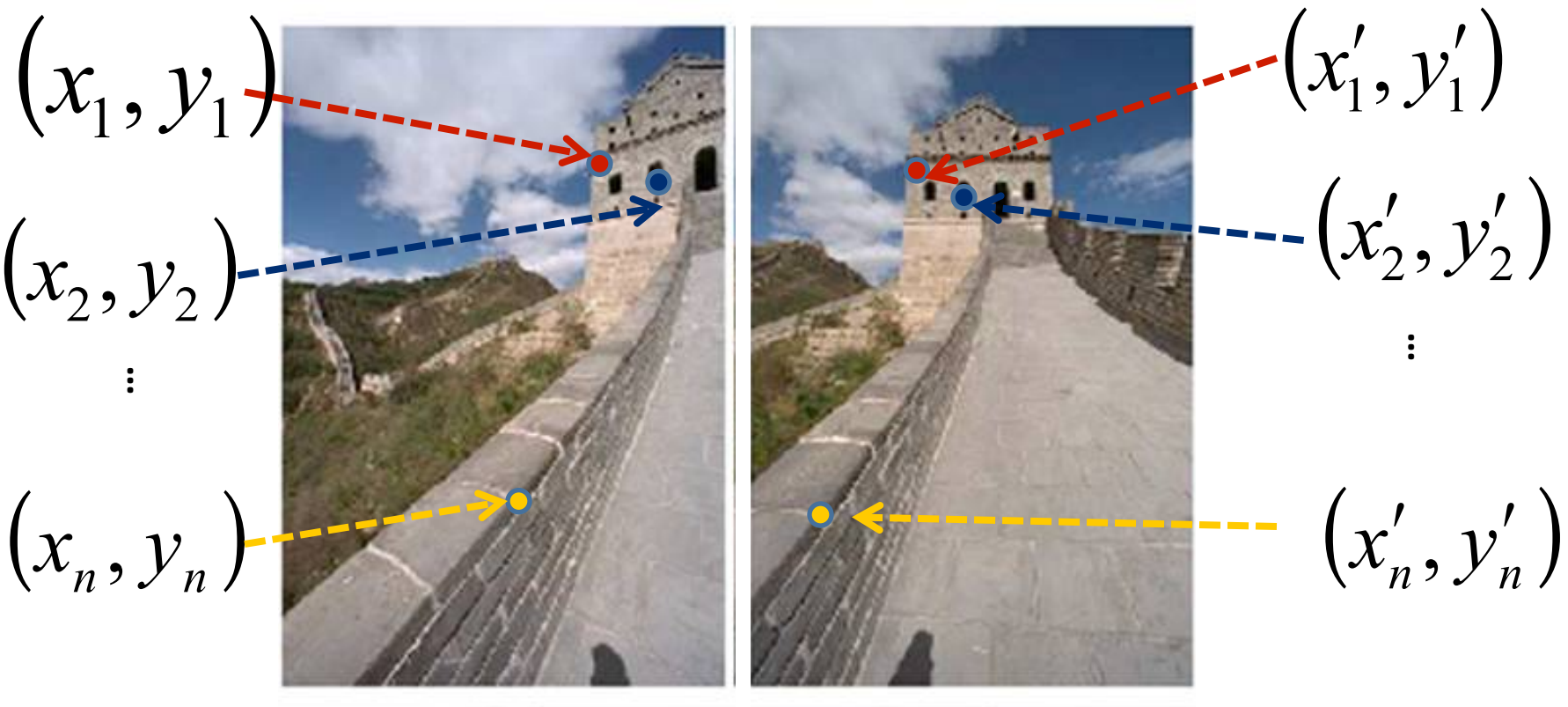
Also known as perspective transformations or homographies.

2D image transformations (reference table)



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	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

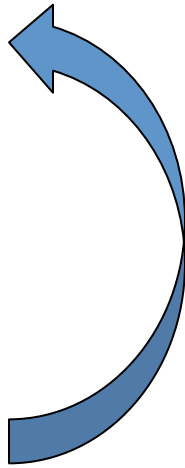
- Need at least 8 eqs, but the more the better…
- Solve for h. If overconstrained, solve using least-squares (lecture 4):

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

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

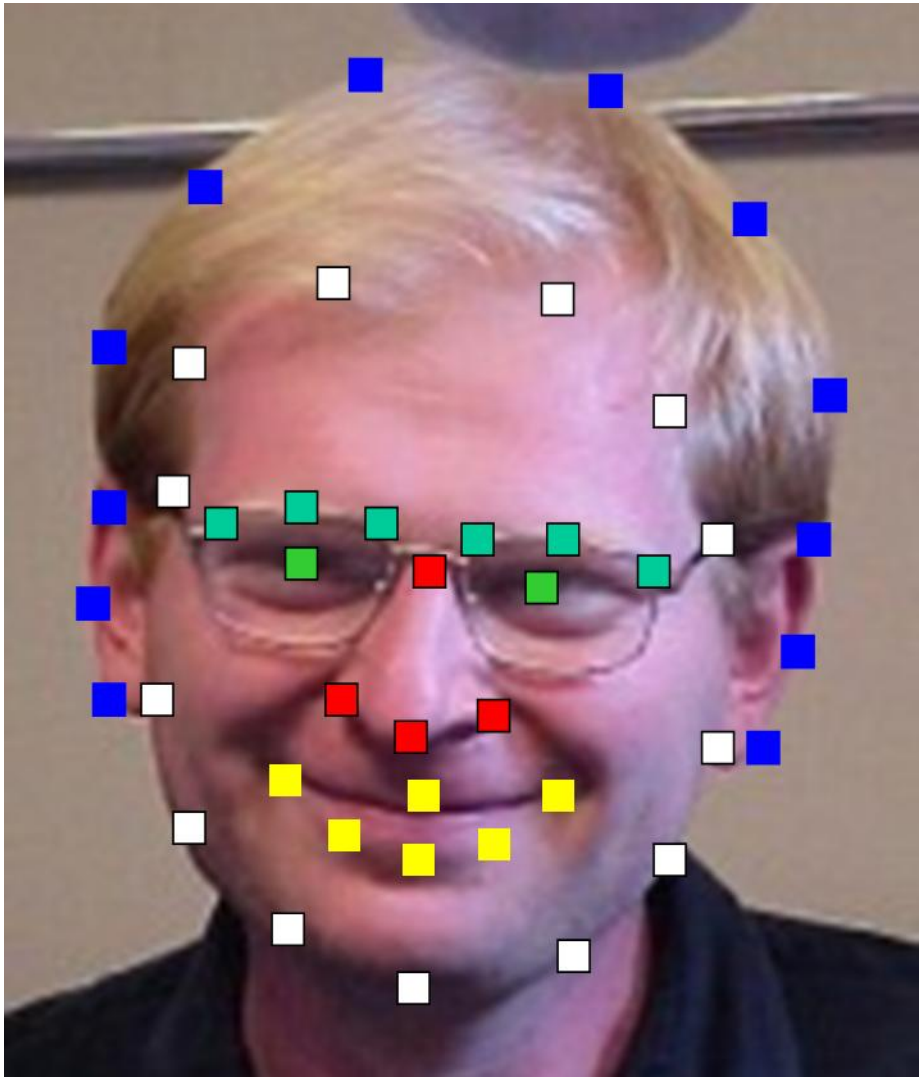
$$\underbrace{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}}_{\mathbf{p'}} = \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p}}$$

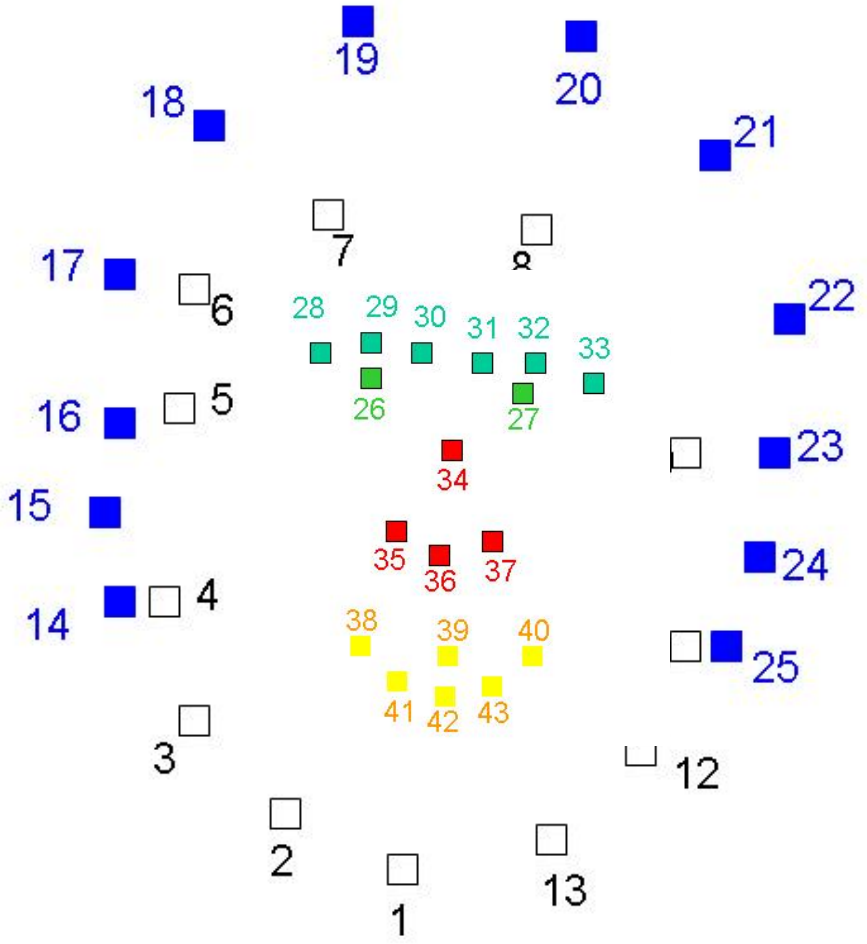
Application to affine transformation: image morphing

Application to affine transformation: image morphing

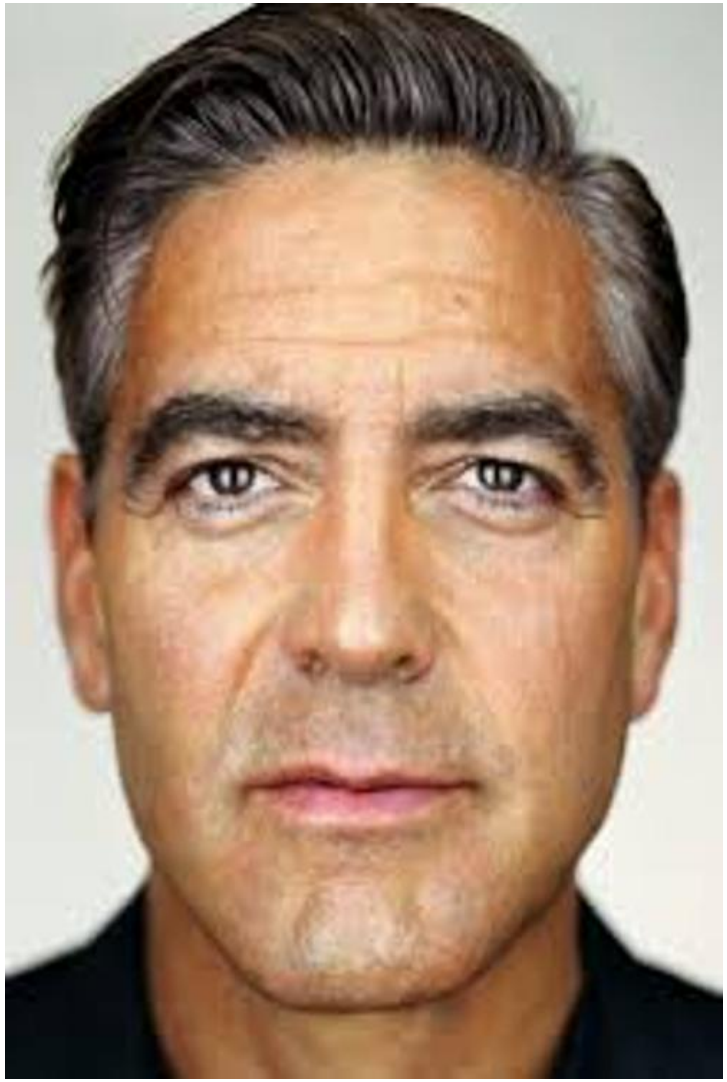
https://www.youtube.com/watch?v=XHb7Ig3yPgI

Facial landmarks: manual or automatic

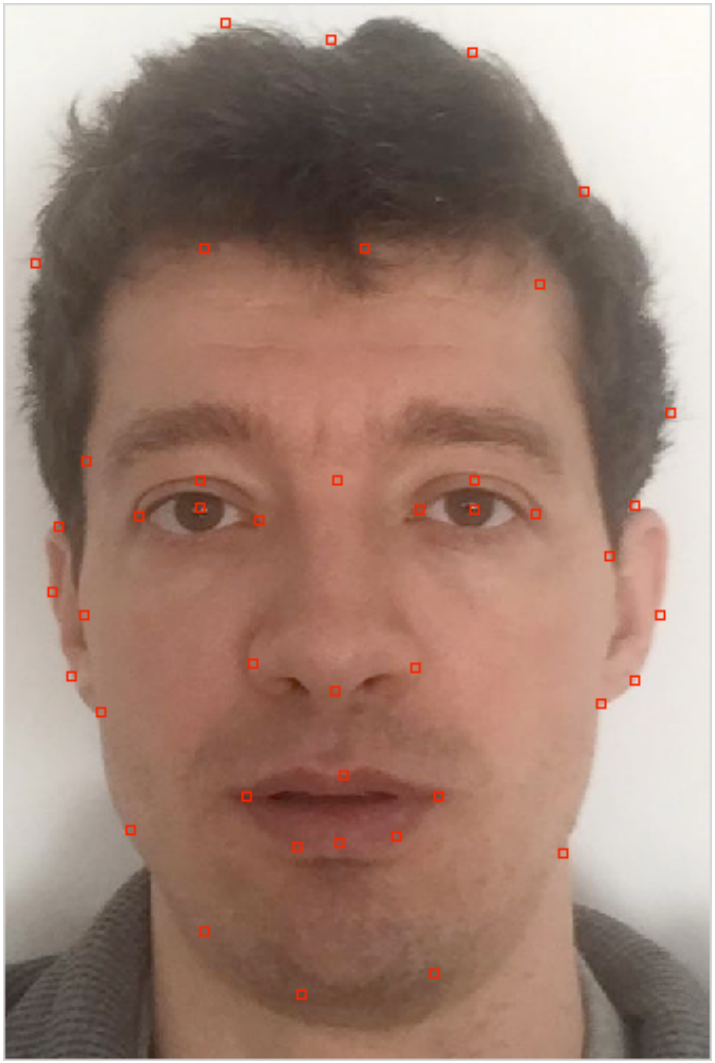




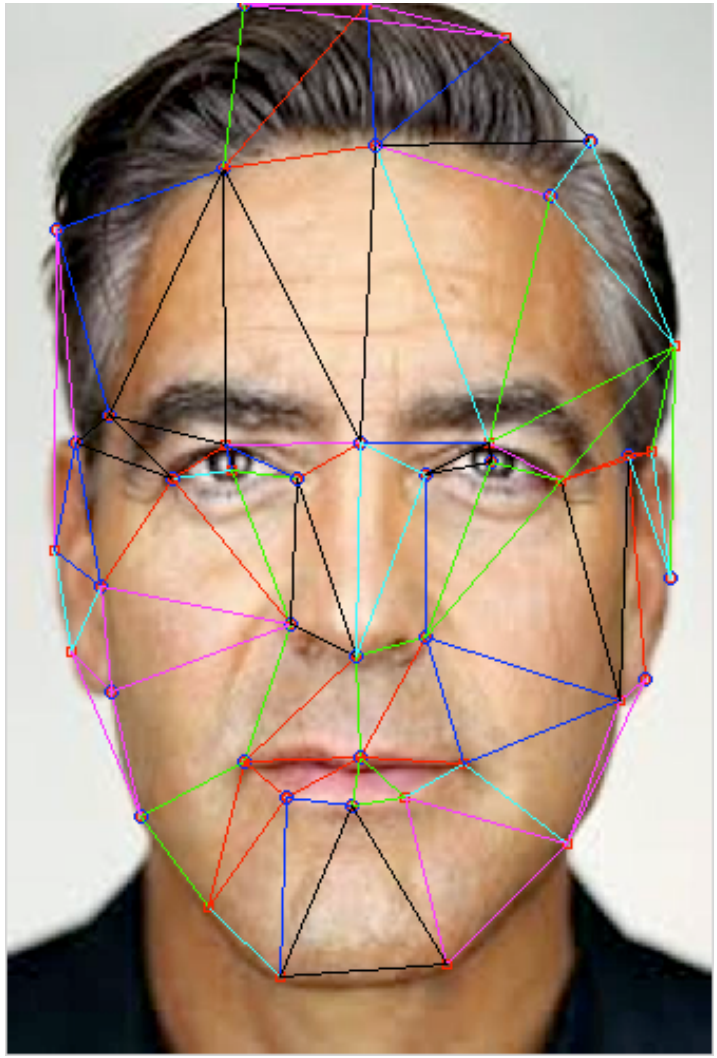
Facial landmarks: manual or automatic



Facial landmarks: manual or automatic



Delauney triangulation



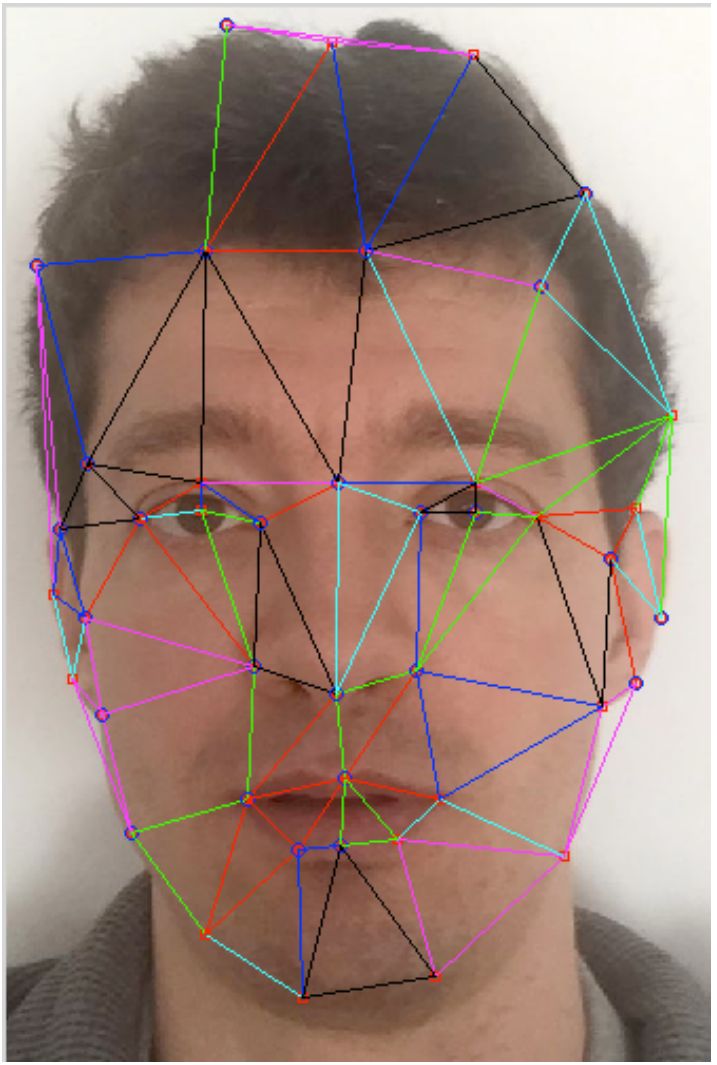
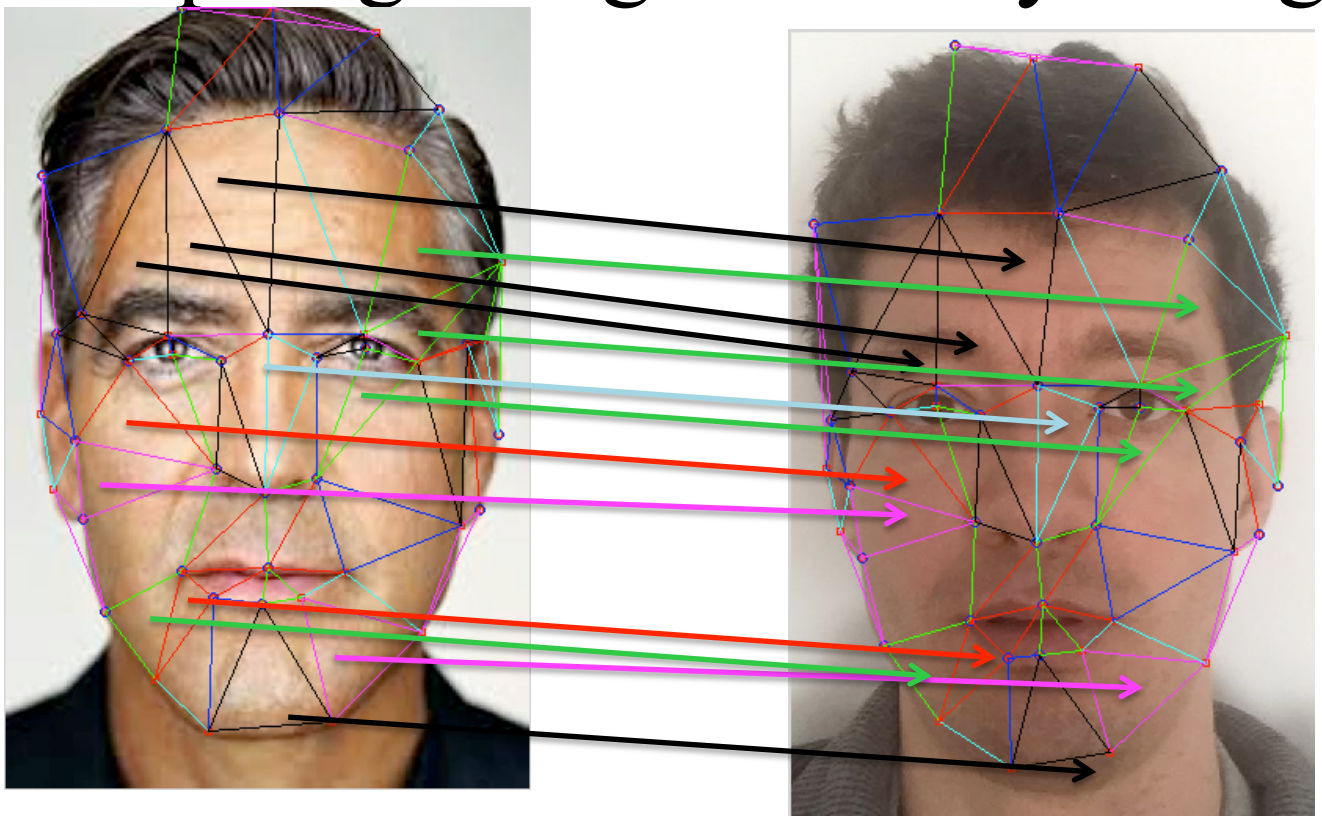


Image morphing using Delauney triangulation



- 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

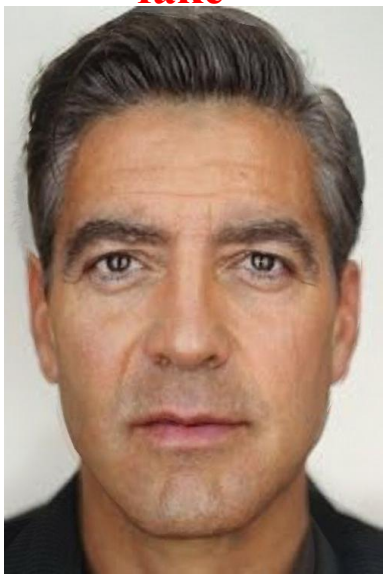
real

fake

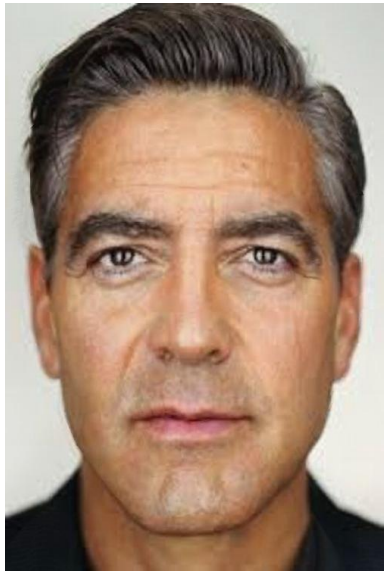
fake

fake

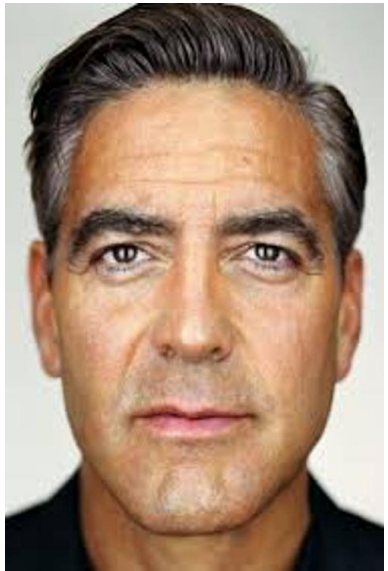
fake



fake

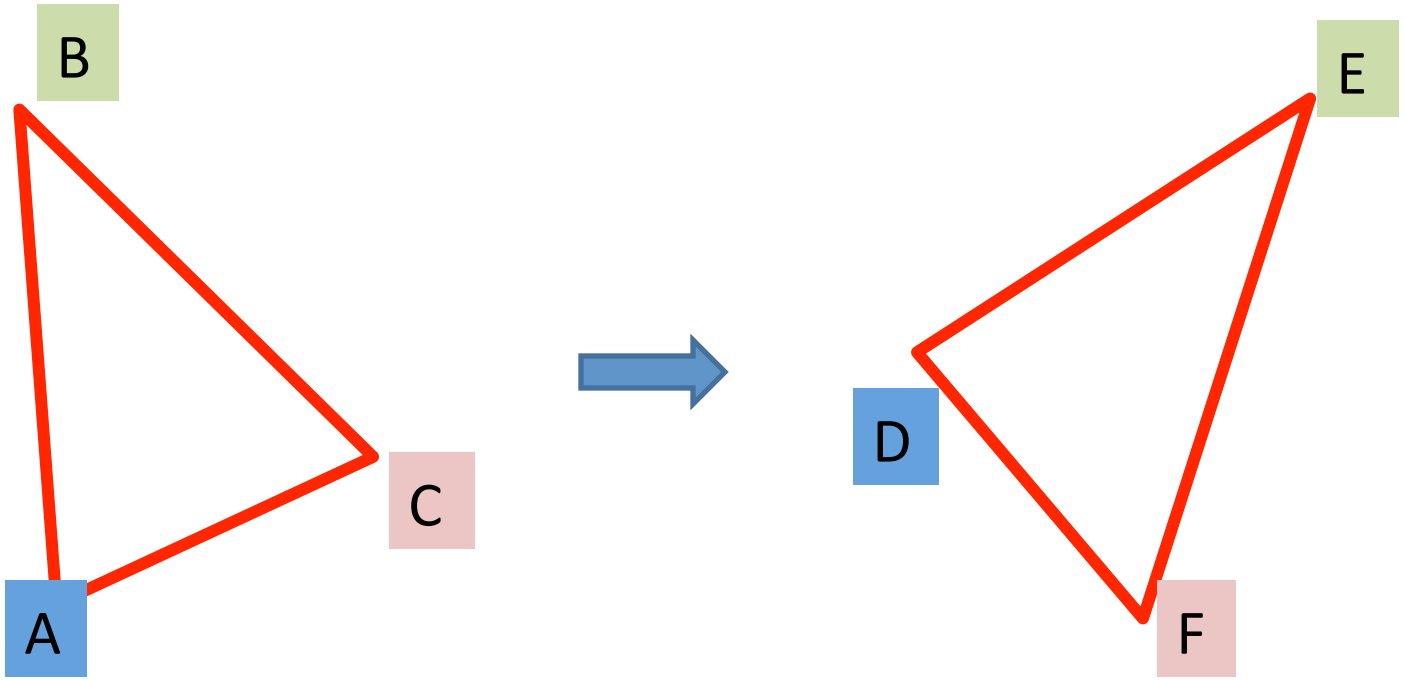


real



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

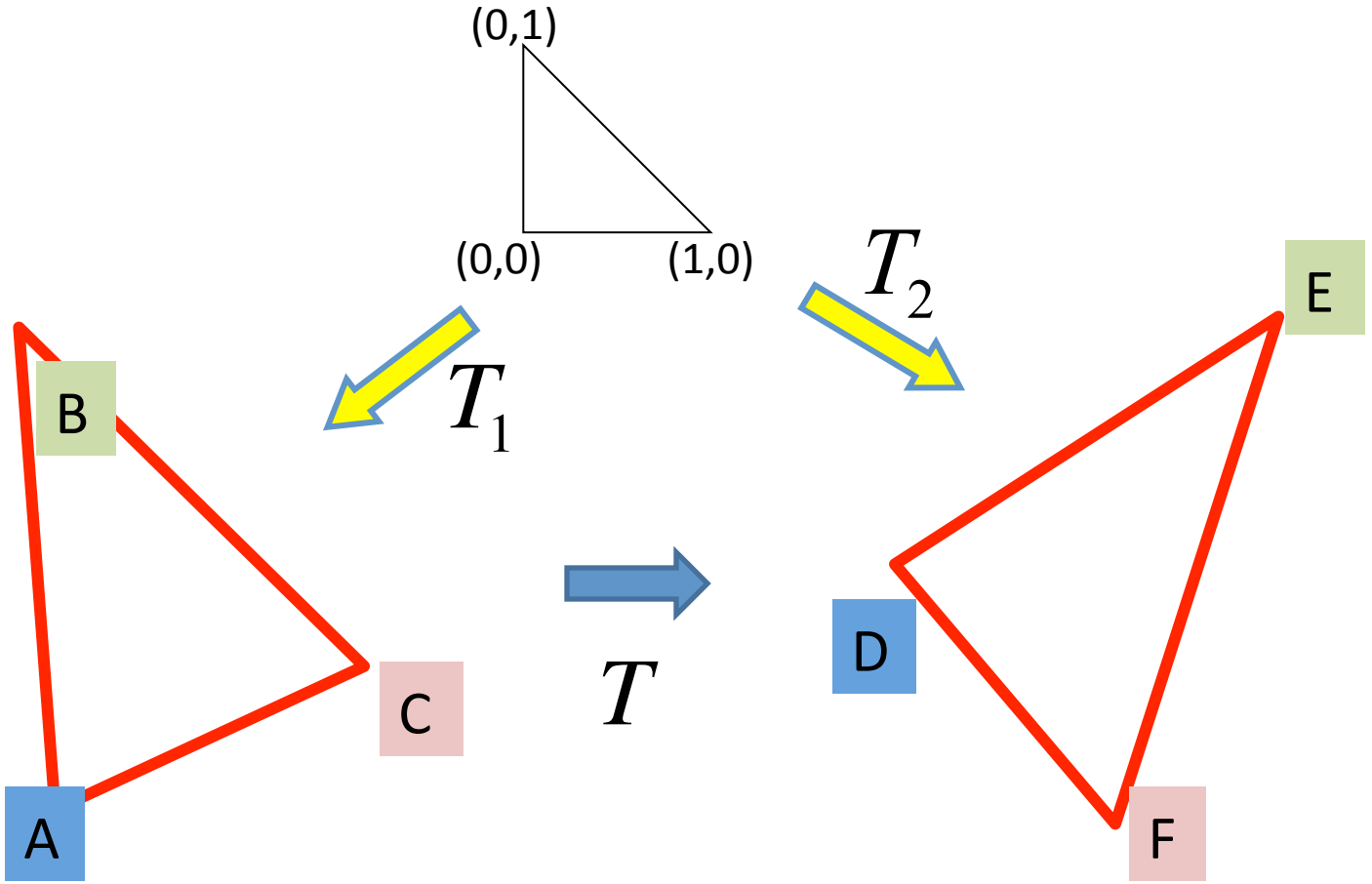
For two triangles ABC and DEF I can find the affine transformation that it will map A in D, B in E and C in F by solving a system with 6 equations and 6 unknowns

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

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

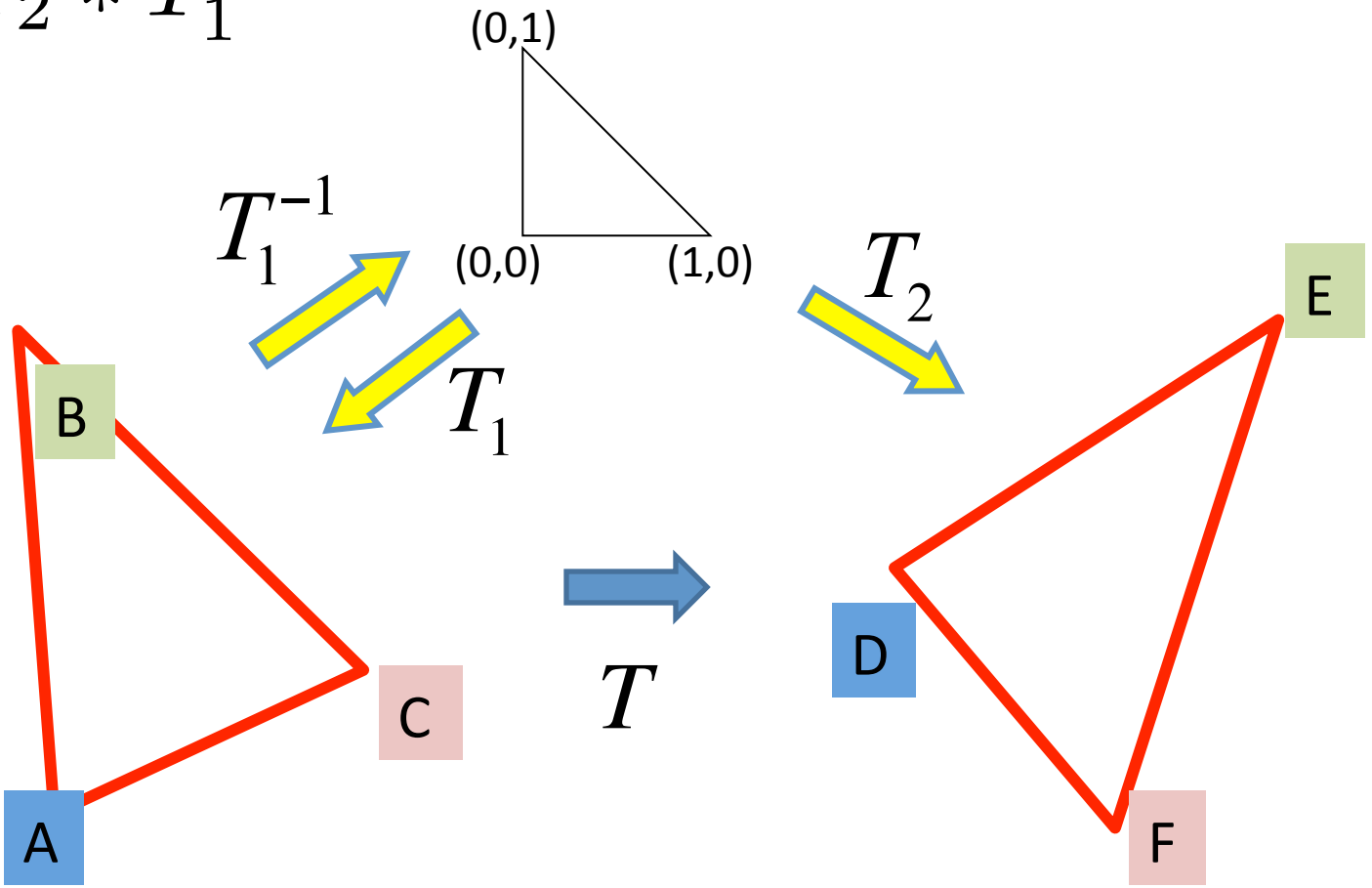
Equivalent approach: affine transformation composition

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:

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

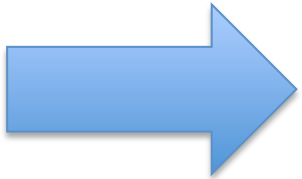
$$\begin{bmatrix} x_E \\ y_E \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Example: Computing T_2

Much nicer system with 6 equations and 6 unknowns:

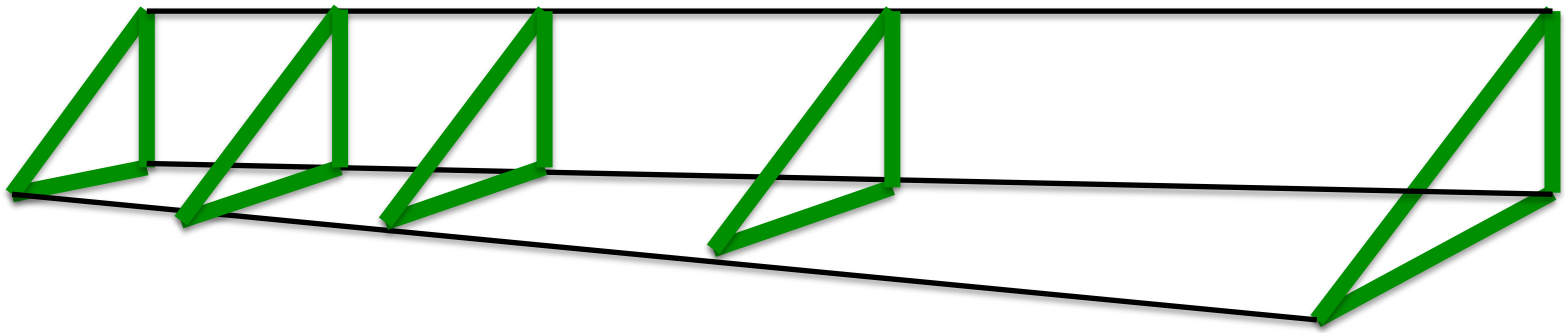
$$\begin{bmatrix} x_D \\ y_D \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x_F \\ y_F \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0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

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



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

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:

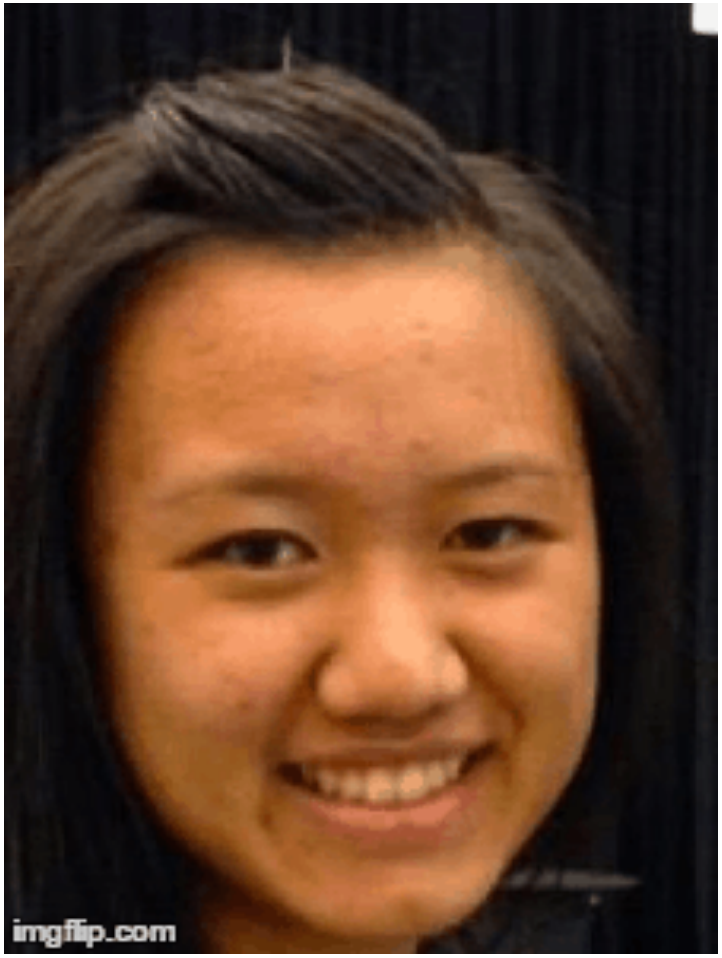
- geometrically: $T_t = (1-t)*T_0 + t*T_1$ ("average weighting" of T_0 and T_1)
- photometrically: for each pixel (x,y) from triangle T_t, find the coresponding pixel (x_0,y_0) from T_0 and (x_1,y_1) from T_1 (using the affine transformations). Then compute:

  $RGB_{(x,y)} = (1-t)*RGB_{(x0,y0)} + t*RGB_{(x1,y1)}$ ("average weighting" a culorilor RGB din T_0 și 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



Course structure

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

   Linear filters, color, texture, edge detection

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

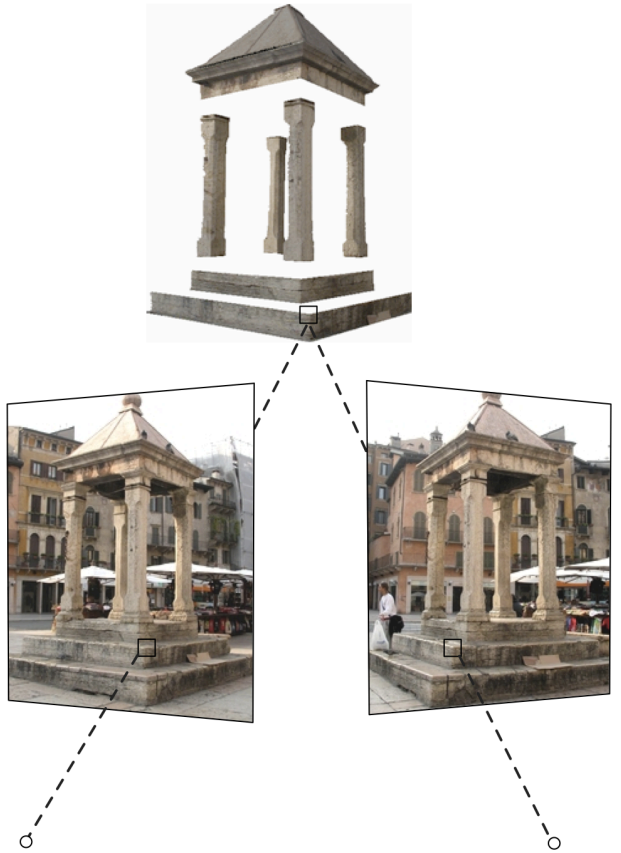
5. **Video understanding**

   Object tracking, background subtraction, motion descriptors, optical flow

Goal: Recovery of 3D structure

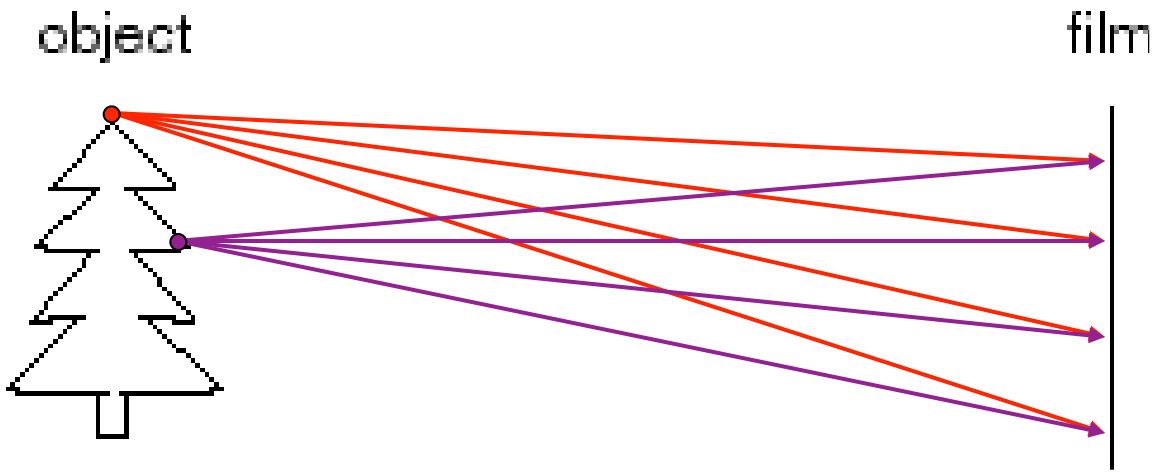
- When certain assumptions hold, we can recover structure from a single view

- In general, we need *multi-view geometry*



- But first, we need to understand the geometry of a single camera…

Image formation

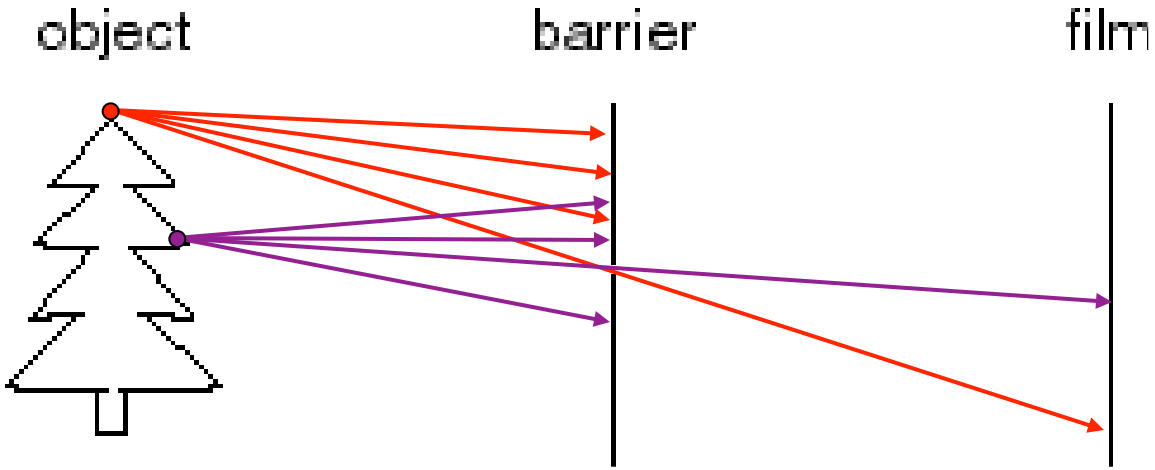


Let's design a camera

- Idea 1: put a piece of film in front of an object
- Do we get a reasonable image?

Slide credit Steve Seitz

Pinhole camera

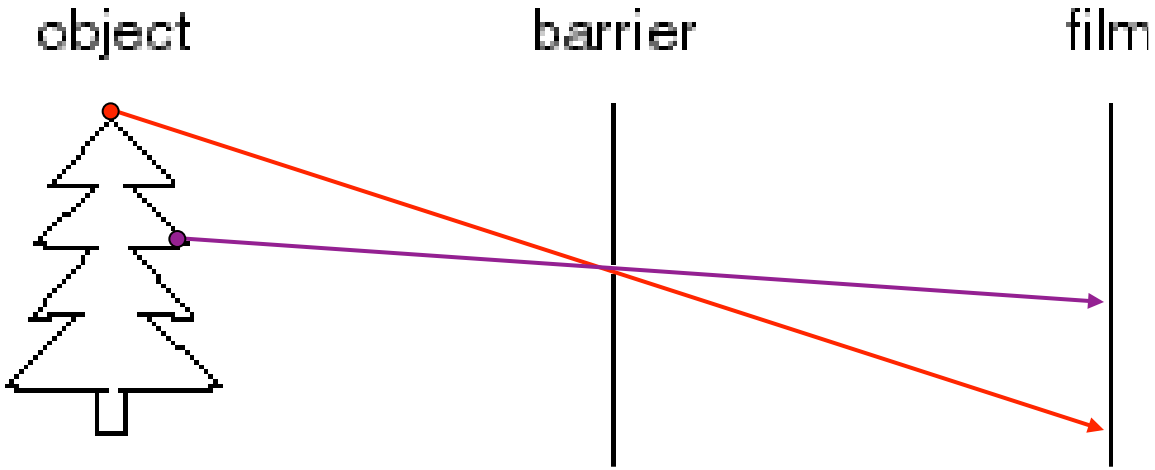


Idea 2: add a barrier to block off most of the rays

- This reduces blurring
- The opening known as the **aperture**

Slide credit Steve Seitz

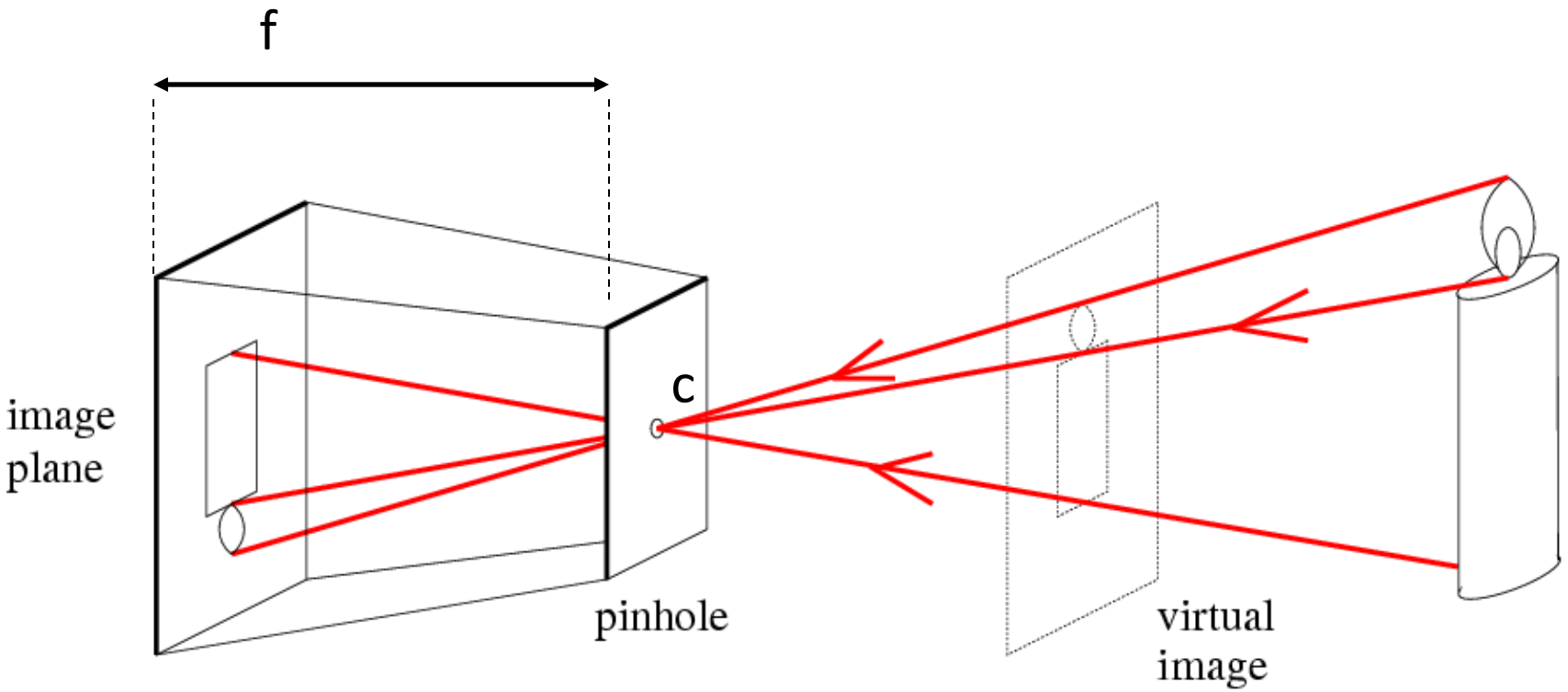
Pinhole camera



- Captures **pencil of rays** – all rays through a single point: **aperture, center of projection, optical center, focal point, camera center**
- The image is formed on the **image plane**

Slide credit Steve Seitz

Pinhole camera

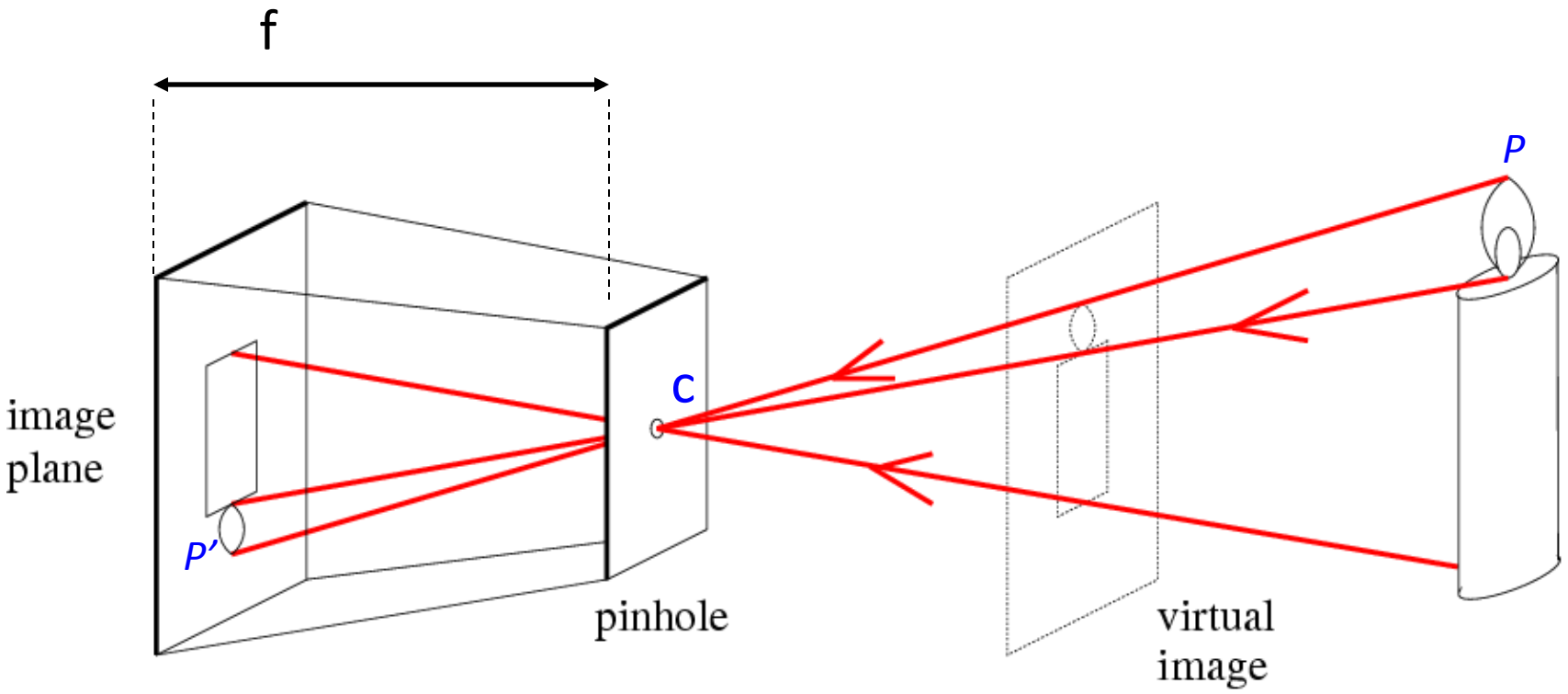


f = focal length
C = center of the camera

Note: instead of dealing with an image that is upside down, most of the time we will pretend that the image plane is in front of the camera center.

Slide credit D. Forsyth

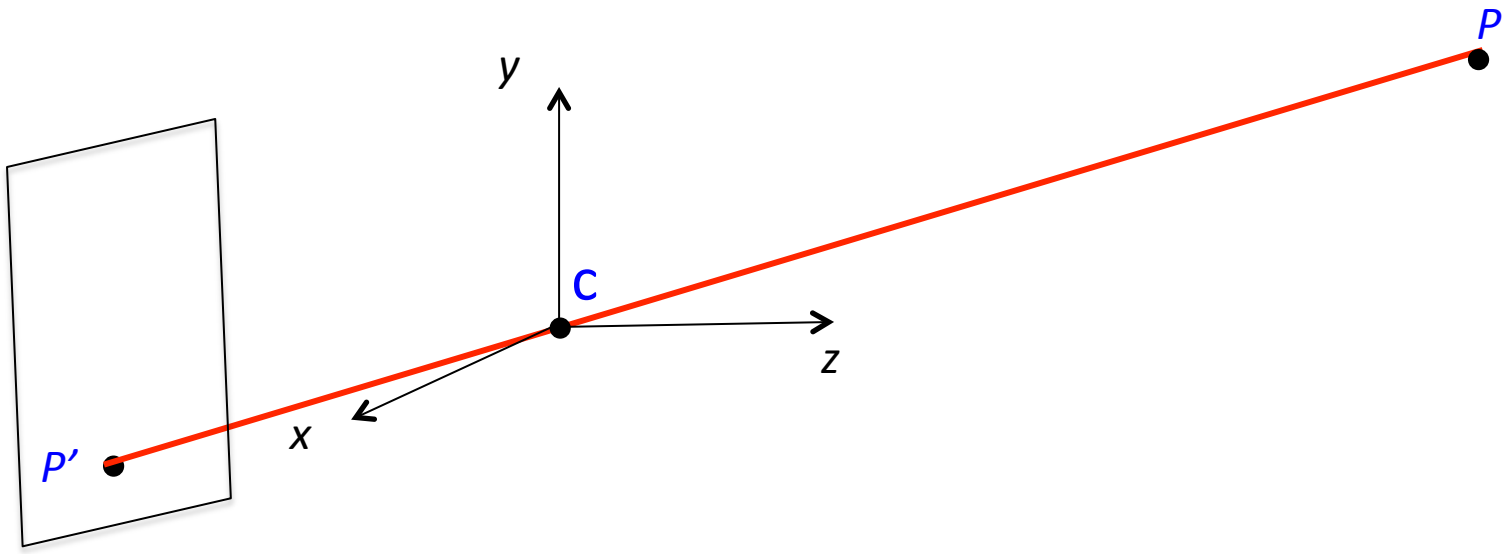
Pinhole projection model



- To compute the projection *P'* of a scene point *P*, form the **visual ray** connecting *P* to the **camera center** *C* and find where it intersects the **image plane**

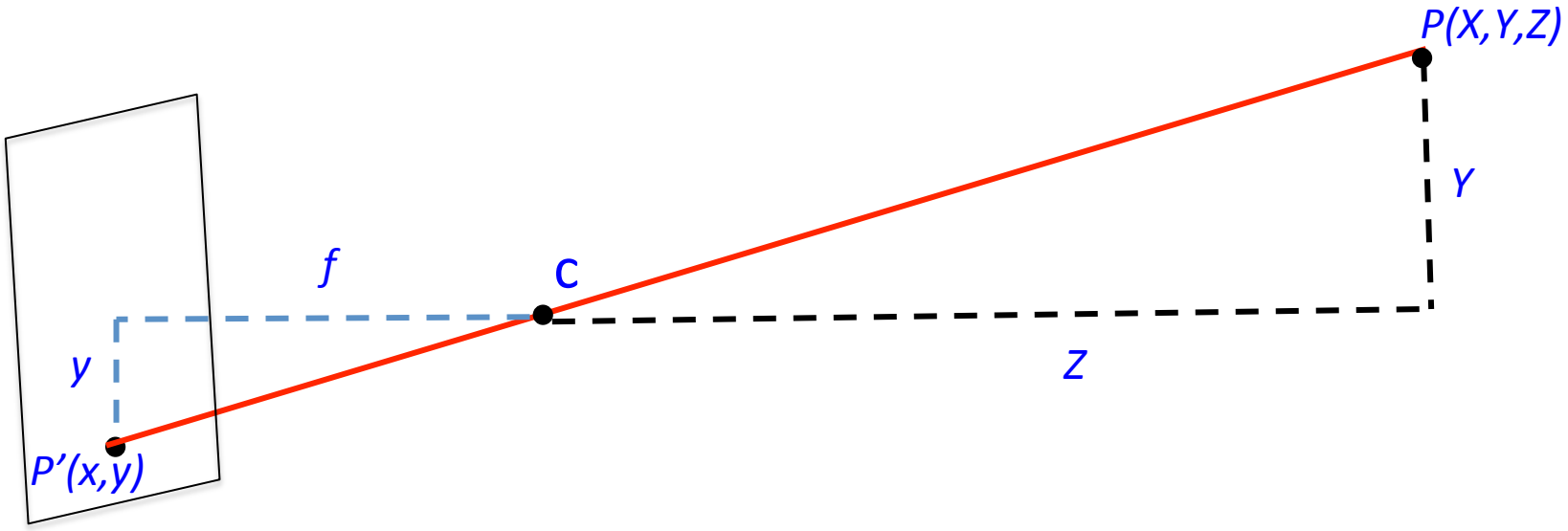
Slide credit D. Forsyth

Pinhole projection model



- The coordinate system
  - The optical center (***C***) is at the origin
  - The image plane is parallel to xy-plane and perpendicular to the z-axis, which is the *optical axis*

Pinhole projection model



- Projection equations – derived using similar triangles

$$\frac{f}{Z}=\frac{y}{Y},\frac{f}{Z}=\frac{x}{X}$$

$$(X,Y,Z)\rightarrow(f\frac{X}{Z},f\frac{Y}{Z})$$

3D world coordinates

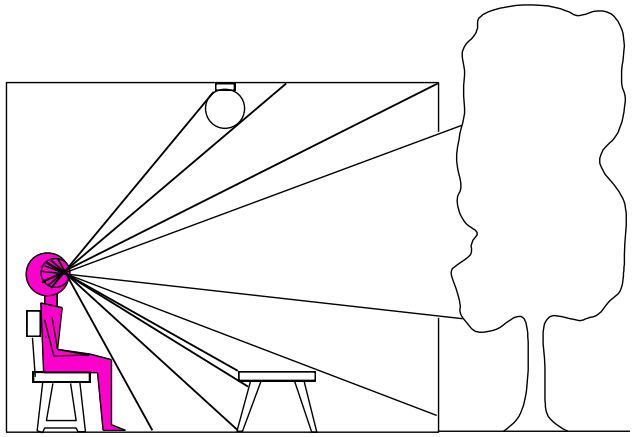
2D image coordinates

$$P(X,Y,Z)\rightarrow P'(f\frac{X}{Z},f\frac{Y}{Z},f)$$

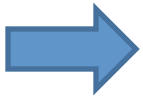
Dimensionality reduction: from 3D to 2D

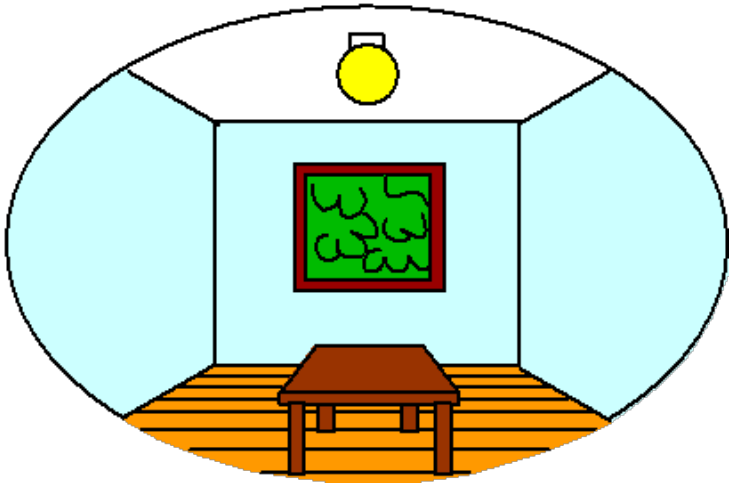
3D world

2D image



Point of observation





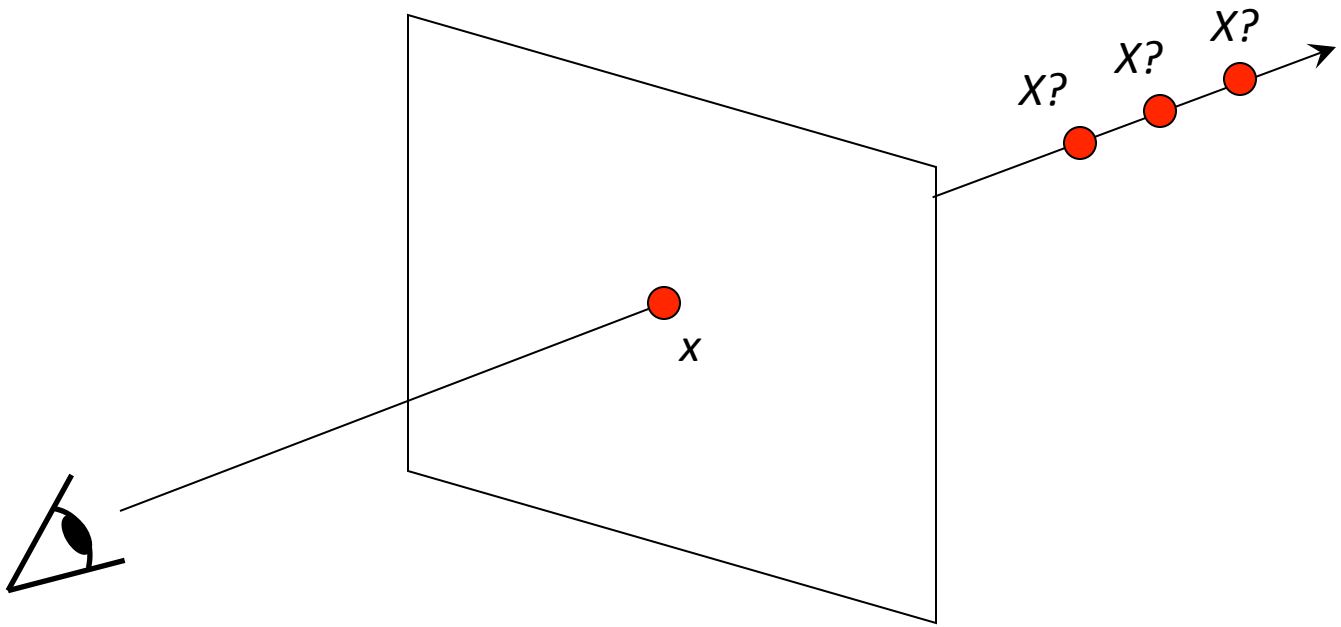
Slide credit D. Forsyth

Projection can be tricky…

Projection can be tricky…

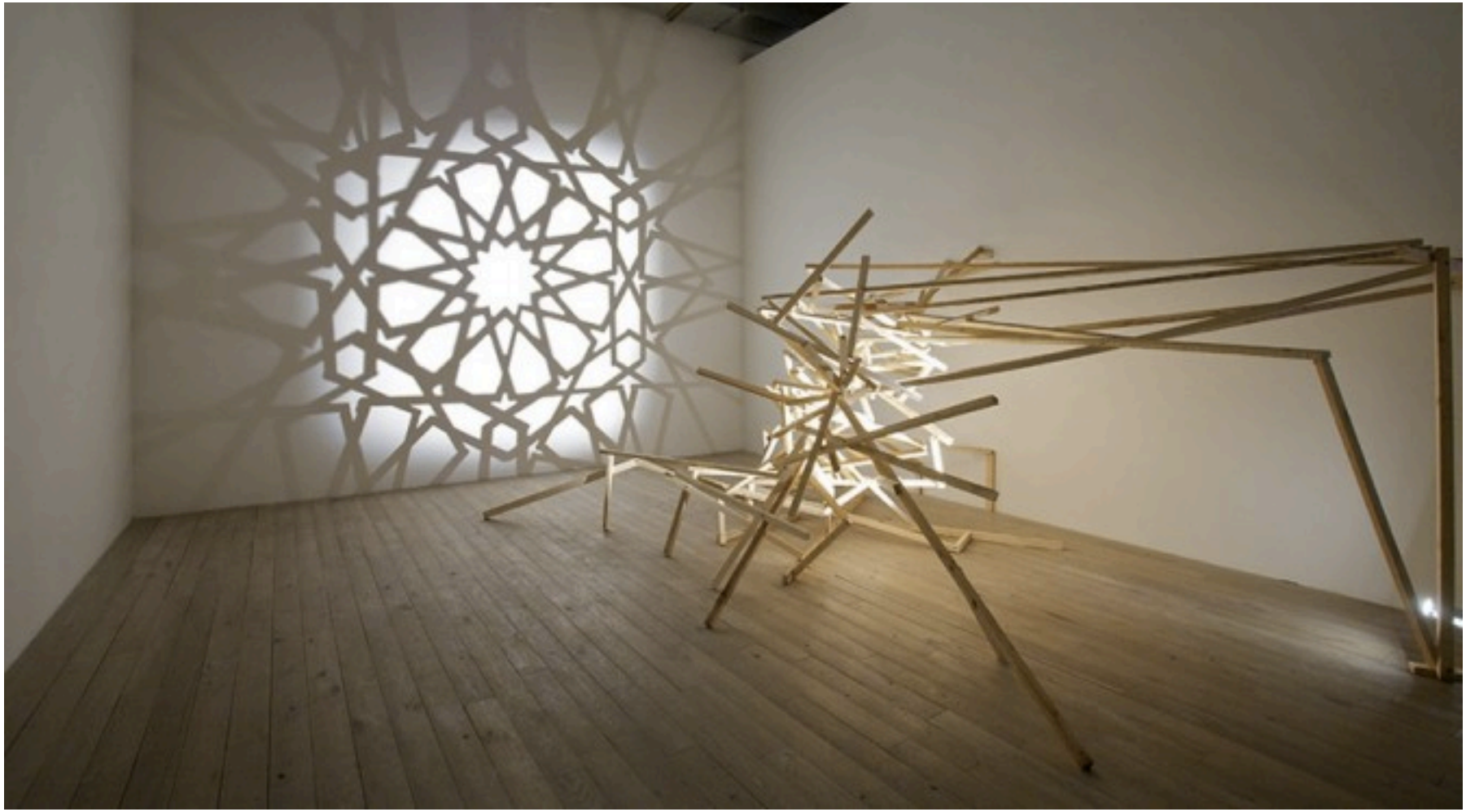
Making of 3D sidewalk art: http://www.youtube.com/watch?v=3SNYtd0Ayt0

Single-view ambiguity



Single-view ambiguity

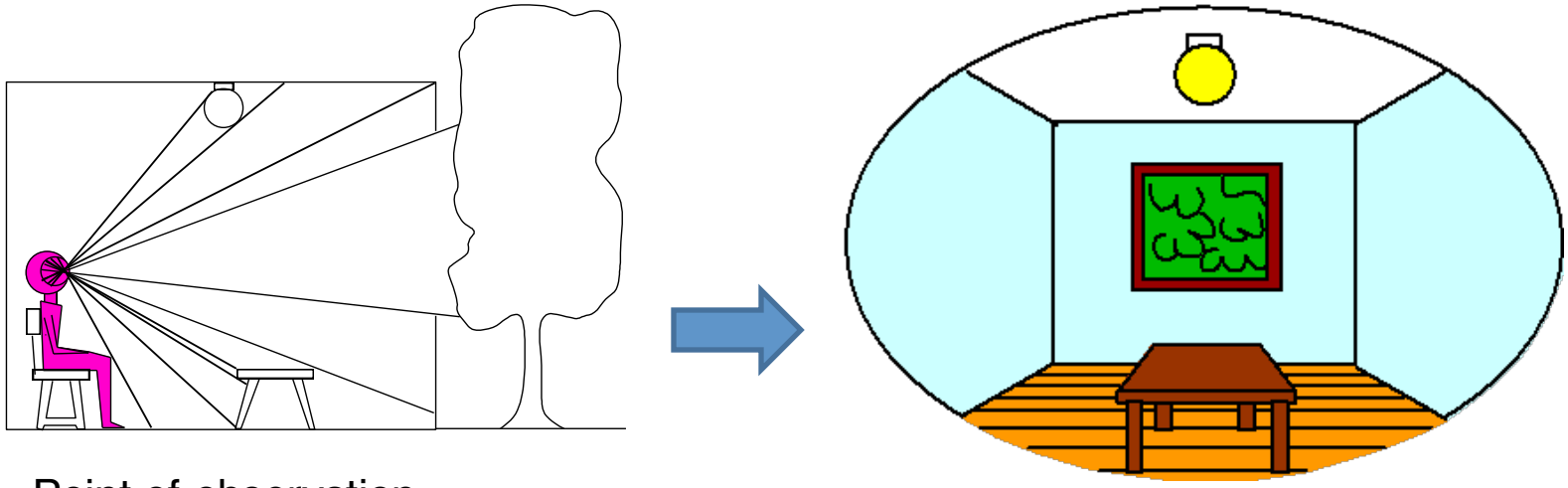
Single-view ambiguity



Rashad Alakbarov shadow sculptures

Dimensionality reduction: from 3D to 2D

3D world *2D image*



Point of observation

What properties of the world are preserved?

- Straight lines, incidence

What properties are not preserved?

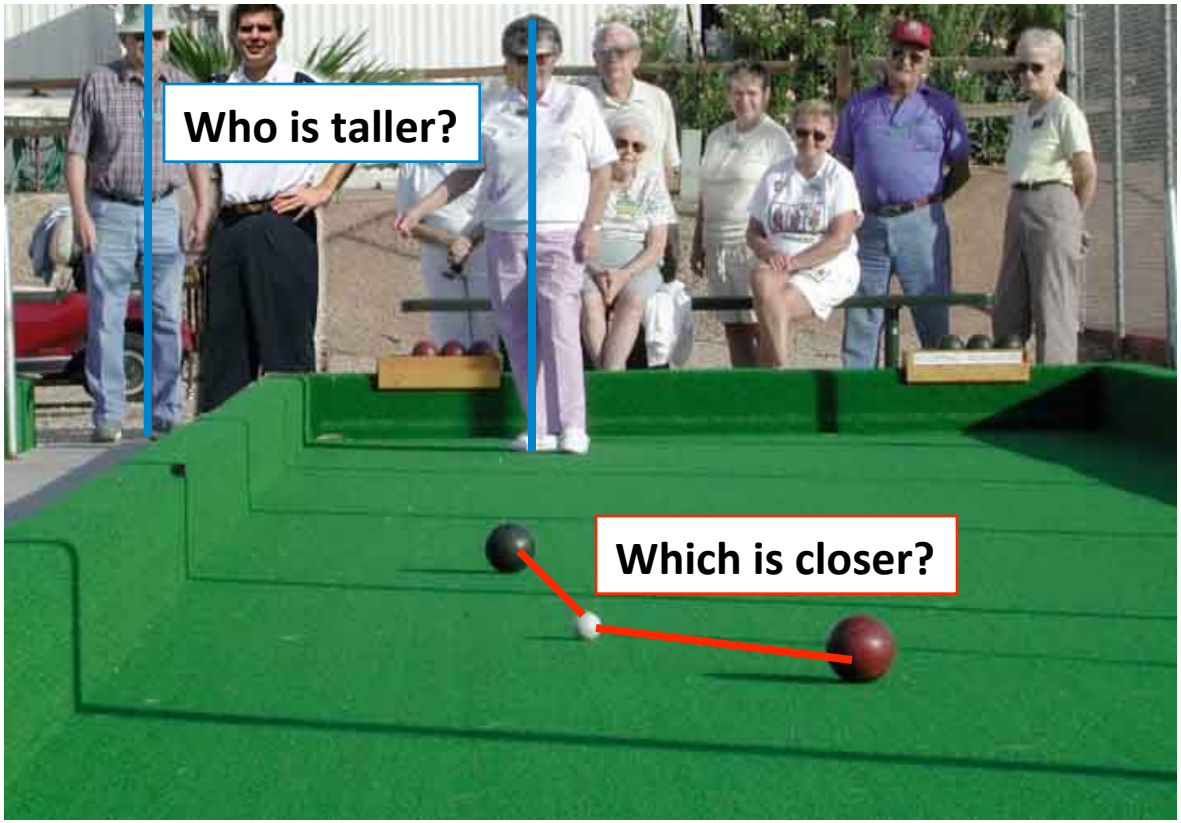
- Angles, lengths

Slide credit D. Forsyth

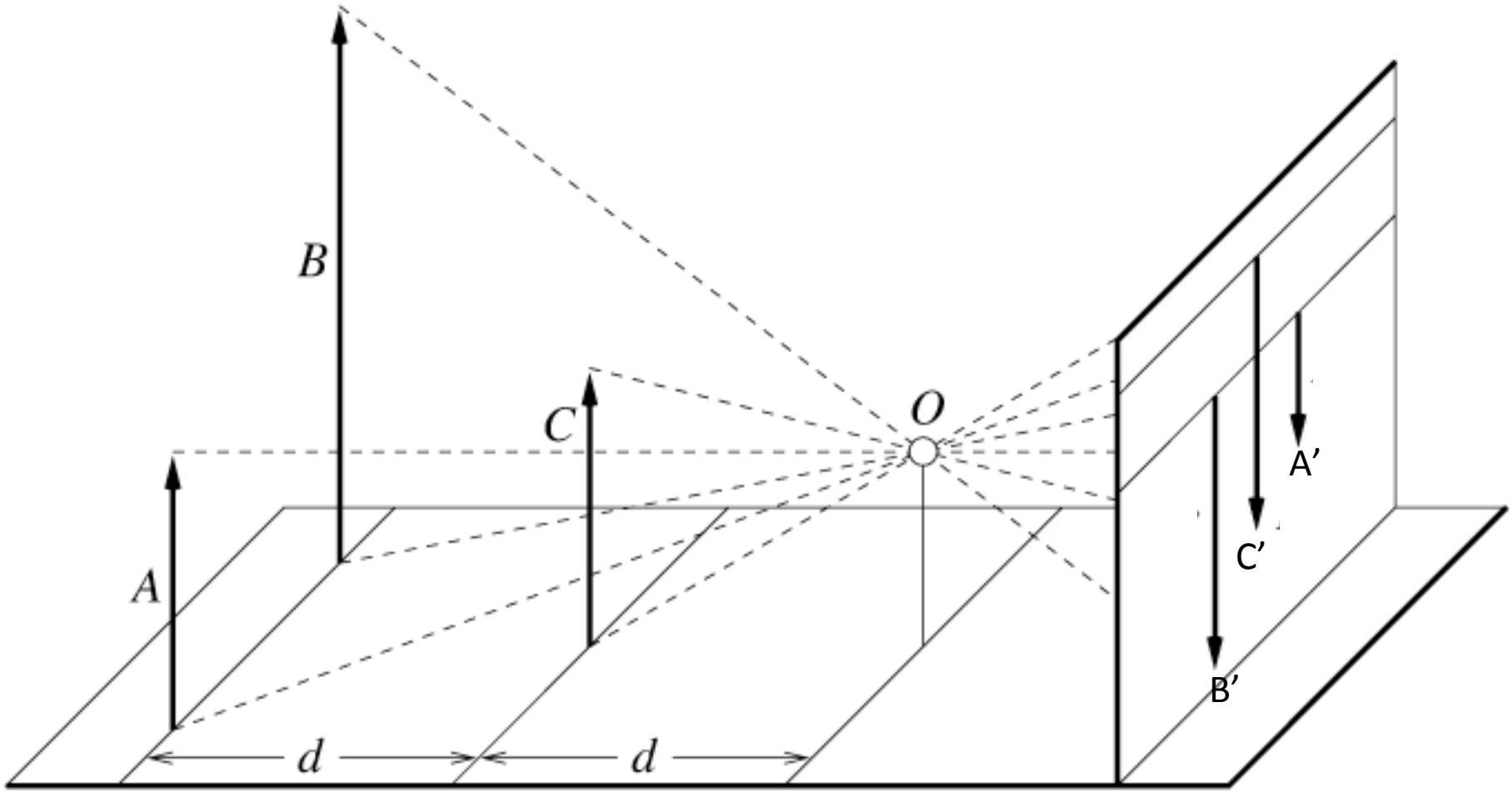
Projective Geometry

What is lost?

- Length



Length is not preserved

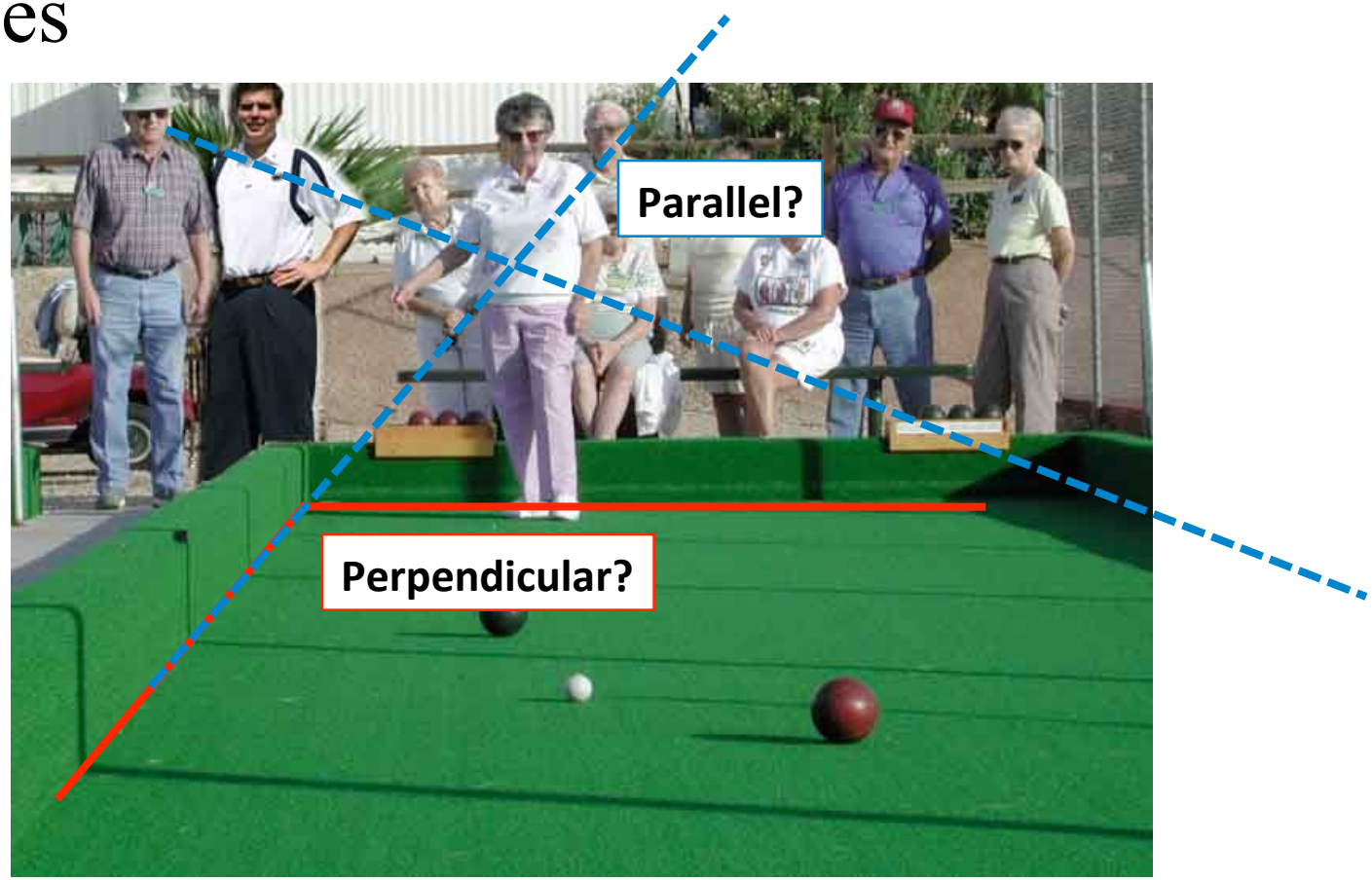


Slide credit D. Forsyth

Projective Geometry

What is lost?

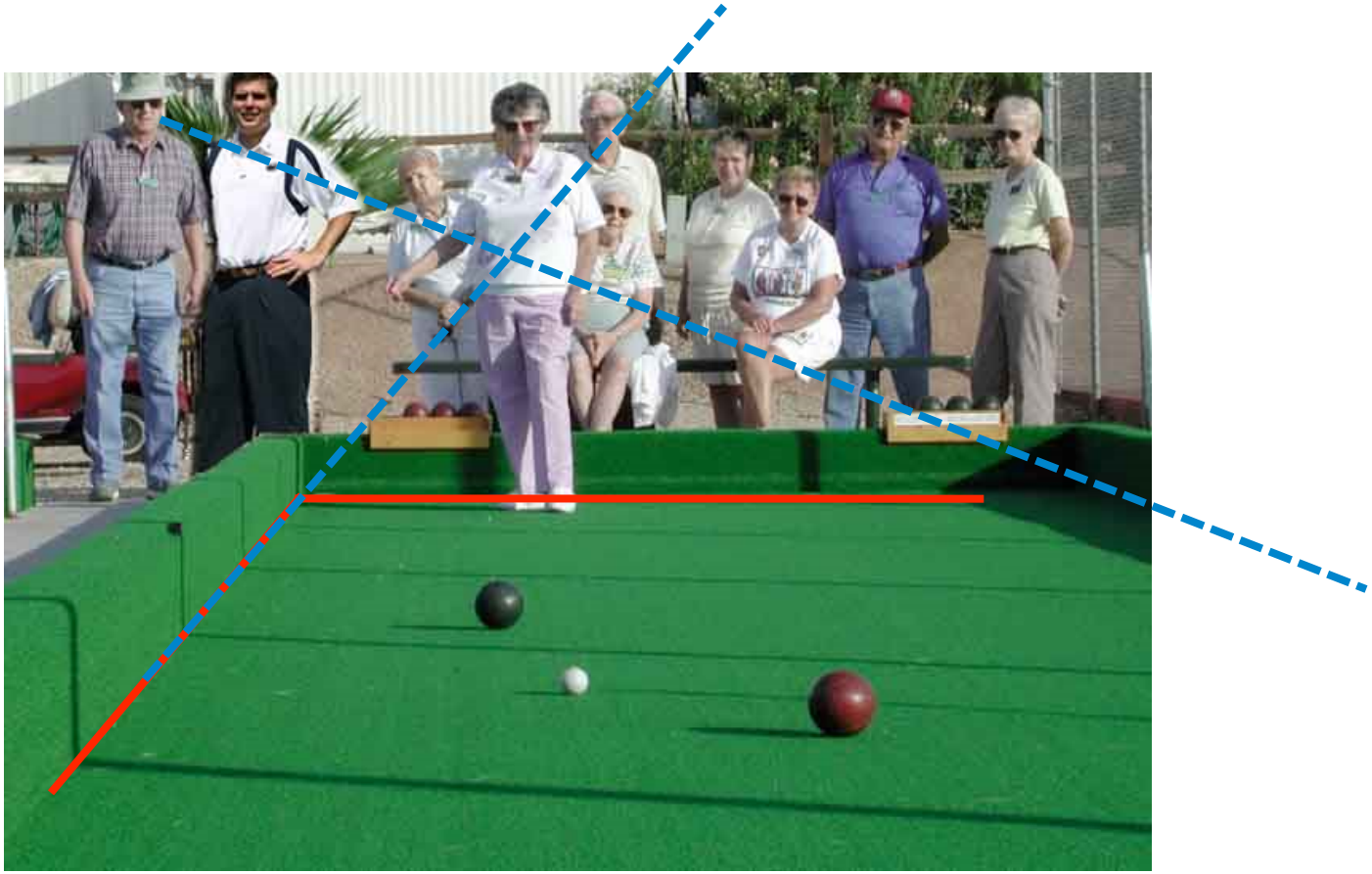
- Length
- Angles



Projective Geometry

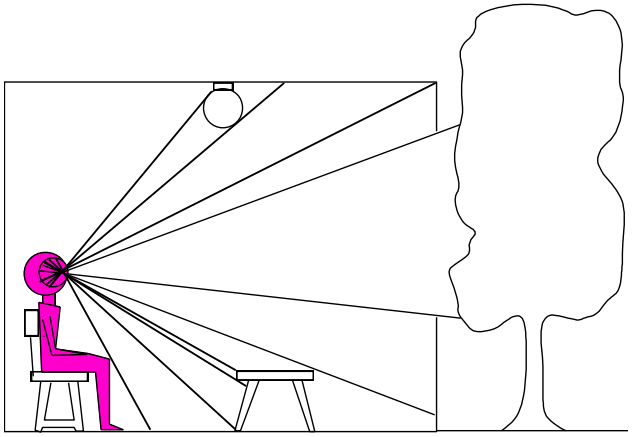
What is preserved?

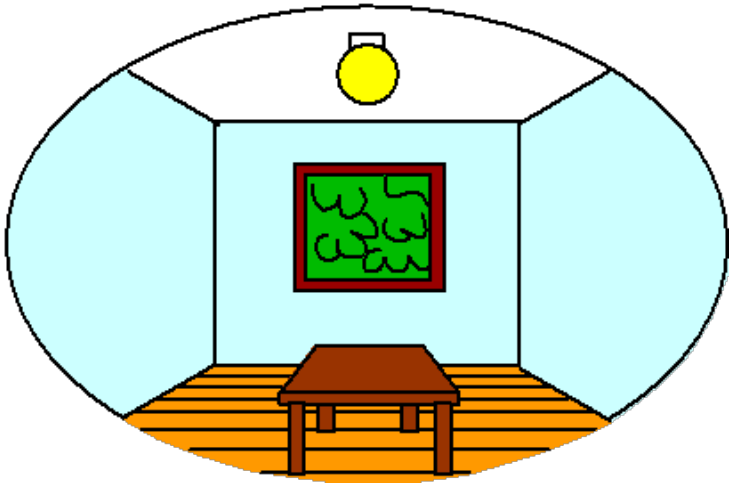
- Straight lines are still straight



Fronto-parallel planes

- What happens to the projection of a pattern on a plane parallel to the image plane?
  - All points on that plane are at a fixed *depth* Z
  - The pattern gets scaled by a factor of f / Z, but angles and ratios of lengths/areas are preserved





$$(X,Y,Z) \rightarrow (f\frac{X}{Z}, f\frac{Y}{Z}) = (X\frac{f}{Z}, Y\frac{f}{Z})$$

Vanishing points and lines

Parallel lines in the world intersect in the image at a "vanishing point"