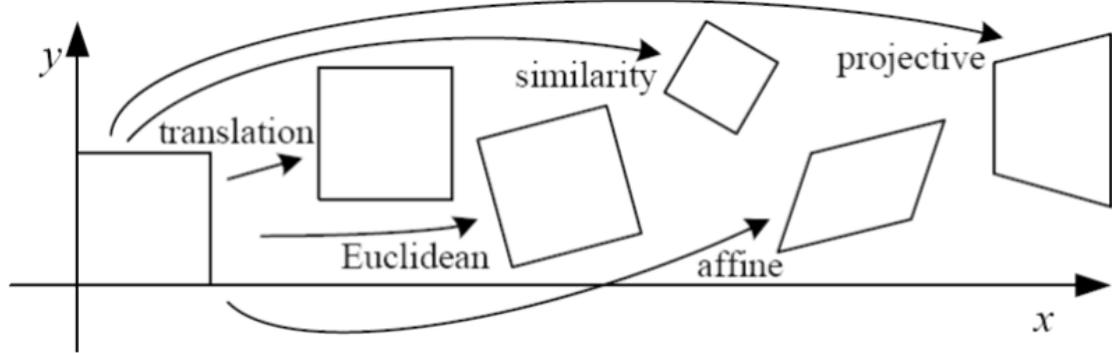




## Tutorial 09 - Homography, Alignment & Panoramas



## Agenda

- [Matching Local Features](#)
- [Parametric Transformations](#)
- [Computing Parametric Transformations](#)
  - [Affine](#)
  - [Projective](#)
  - [Excercise 1 \(Spring 2020 Q1A\)](#)
- [RANSAC](#)
  - [Excercise 2 \(Spring 2020 Q1D\) - Ransac](#)
- [Panorama](#)
  - [Warping](#)
    - [Excercise 3 \(Spring 2020 Q1B\)](#)
  - [Image Blending \(Feathering\)](#)
- [Kornia & Transformations in Deep Learning](#)
- [Recommended Videos](#)
- [Credits](#)

The largest panorama in the world (2014): Mont Blanc

In2WHITE Video ([https://youtu.be/Dwyx0h9h8zg\\_](https://youtu.be/Dwyx0h9h8zg_))

In2WHITE Full Image (<http://www.in2white.com/>)

Homographic usage examples:

<http://blog.flickr.net/en/2010/01/27/a-look-into-the-past/> (<http://blog.flickr.net/en/2010/01/27/a-look-into-the-past/>)

<https://www.instagram.com/albumplusart/> (<https://www.instagram.com/albumplusart/>)

In [1]:

```
%html  
<iframe src="http://www.in2white.com/" width="700" height="600"></iframe>
```



Filippo B. (<http://www.filippobengini.com>)

▲ Sponsored by: ▲



In [2]:

```
# imports for the tutorial  
import numpy as np,sys  
import matplotlib.pyplot as plt  
from PIL import Image  
import cv2  
from scipy import signal  
%matplotlib inline
```

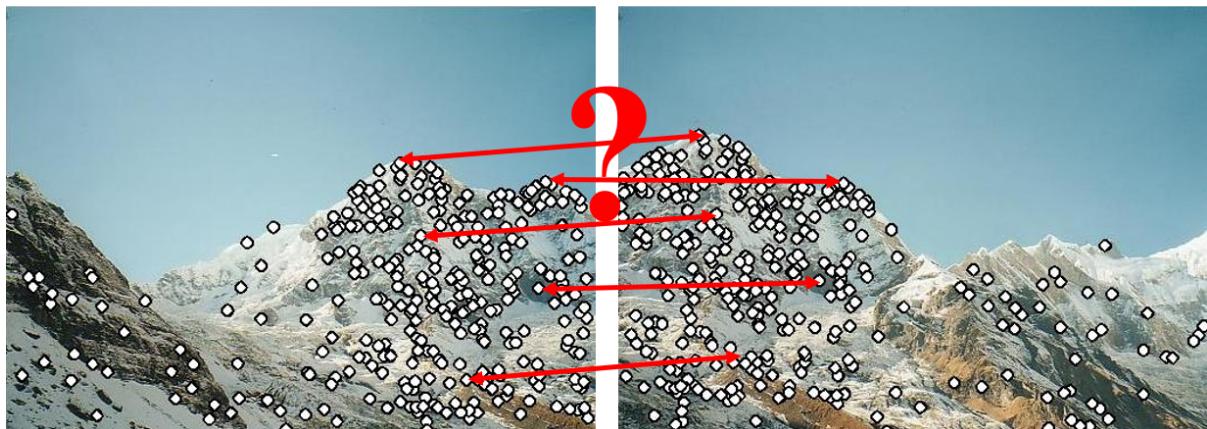
In [3]:

```
# plot images function
def plot_images(image_list, title_list, subplot_shape=(1,1), axis='off', fontsize=30, figsize=(4,4), cmap=['gray']):
    plt.figure(figsize=figsize)
    for ii, im in enumerate(image_list):
        c_title = title_list[ii]
        if len(cmap) > 1:
            c_cmap = cmap[ii]
        else:
            c_cmap = cmap[0]
        plt.subplot(subplot_shape[0], subplot_shape[1], ii+1)
        plt.imshow(im, cmap=c_cmap)
        plt.title(c_title, fontsize=fontsize)
        plt.axis(axis)
```

## Matching Local Features

### Feature matching

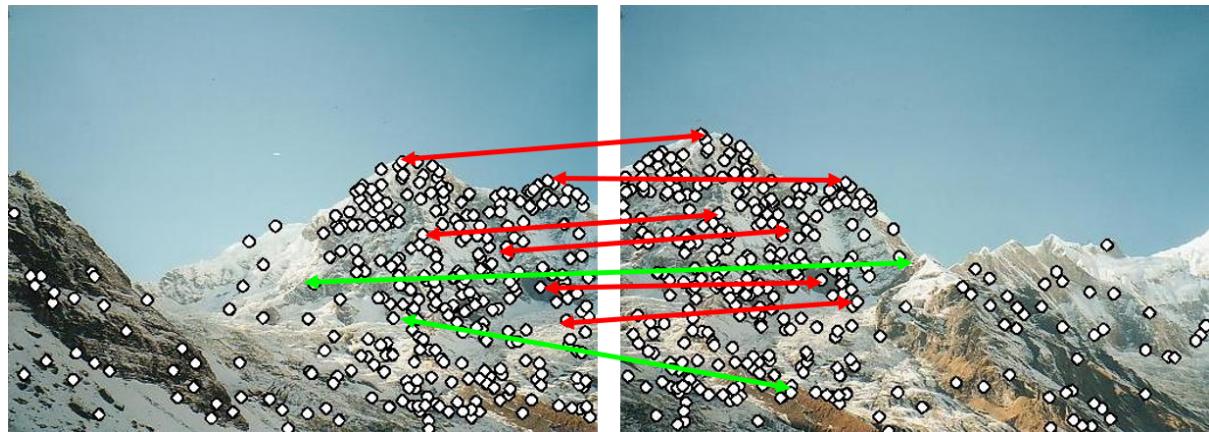
- We know how to detect and describe good points
- Next question: How to match them?



## Typical feature matching results

---

- Some matches are correct
- Some matches are incorrect



- Solution: search for a set of geometrically consistent matches



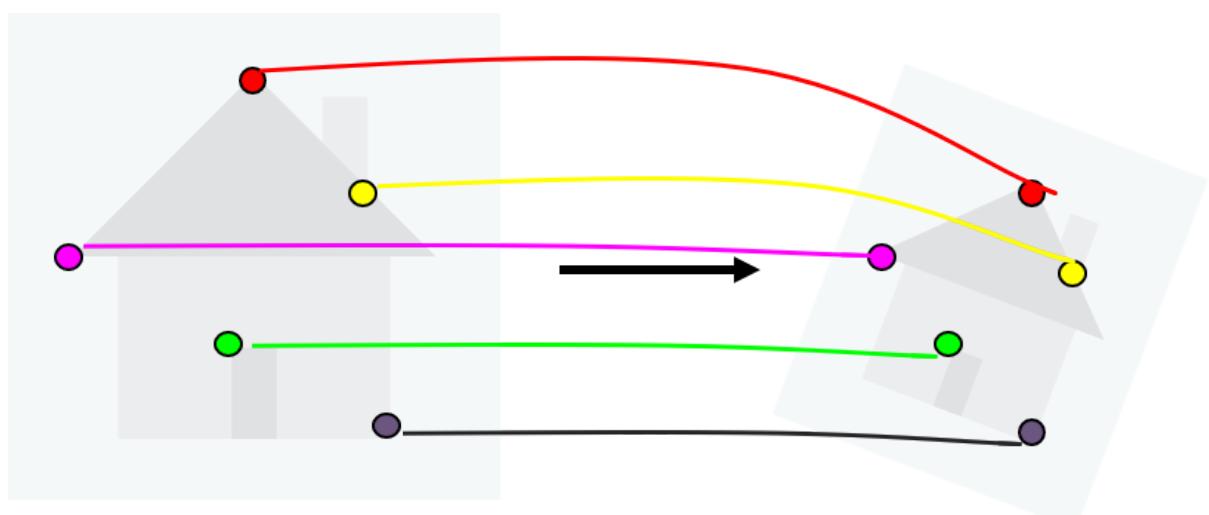
## Parametric Transformations

---

### Image Alignment

---

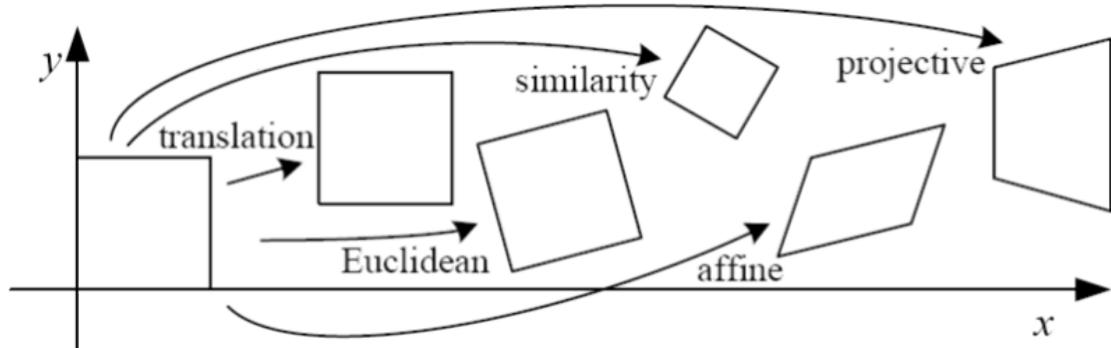
Given a set of matches, what parametric model describes a geometrically consistent transformation?



## Basic 2D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} I & t \end{bmatrix}_{2 \times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} R & t \end{bmatrix}_{2 \times 3}$	3	lengths	
similarity	$\begin{bmatrix} sR & t \end{bmatrix}_{2 \times 3}$	4	angles	
affine	$\begin{bmatrix} A \end{bmatrix}_{2 \times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3 \times 3}$	8	straight lines	

( $x$  is in homogeneous coordinates)



## Parametric (Global) Warping

Examples of parametric warps:

In [4]:

```
im = cv2.imread('./assets/tut_8_exm.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
rows, cols, d = im.shape
print(im.shape)

# Translation:
tx,ty = [10,20]
h_T = np.float32([[1,0,tx],[0,1,ty],[0,0,1]])

# Rotation
theta = np.deg2rad(20)
h_R = np.float32([[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0, 0, 1]])

# Affine
h_AFF = np.array([[ 1.26666667, -0.5, -60], [ -0.33333333, 1, 66.66666667], [0, 0, 1]])

H = [h_T,h_R,h_AFF]

img = [im]
for h in H:
    img.append(cv2.warpPerspective(im, h,(cols,rows)))
Titles = ['Original','Translation','Rotation','Affine']

# Perspective
pts1 = np.float32([[100,77],[320,105],[100,150],[385,170]])
pts2 = np.float32([[0,0],[300,0],[0,100],[300,50]])
M = cv2.getPerspectiveTransform(pts1,pts2)
im_perspective = cv2.warpPerspective(im,M,(400,300))

# Cylindrical (non Linear)
def cylindricalWarp(img, K):
    """This function returns the cylindrical warp for a given image and intrinsics matrix K"""
    h_,w_ = img.shape[:2]
    # pixel coordinates
    y_i, x_i = np.indices((h_,w_))
    X = np.stack([x_i,y_i,np.ones_like(x_i)],axis=-1).reshape(h_*w_,3) # to homog
    Kinv = np.linalg.inv(K)
    X = Kinv.dot(X.T).T # normalized coords
    # calculate cylindrical coords (sin\theta, h, cos\theta)
    A = np.stack([np.sin(X[:,0]),X[:,1],np.cos(X[:,0])],axis=-1).reshape(w_*h_,3)
    B = K.dot(A.T).T # project back to image-pixels plane
    # back from homog coords
    B = B[:, :-1] / B[:, [-1]]
    # make sure warp coords only within image bounds
    B[(B[:,0] < 0) | (B[:,0] >= w_) | (B[:,1] < 0) | (B[:,1] >= h_)] = -1
    B = B.reshape(h_,w_,-1)

    img_rgba = cv2.cvtColor(img, cv2.COLOR_RGB2RGBA) #BGR2BGRA for transparent border
    ...
    # warp the image according to cylindrical coords
    return cv2.remap(img_rgba, B[:, :, 0].astype(np.float32), B[:, :, 1].astype(np.float32),
    ), cv2.INTER_AREA, borderMode=cv2.BORDER_TRANSPARENT)

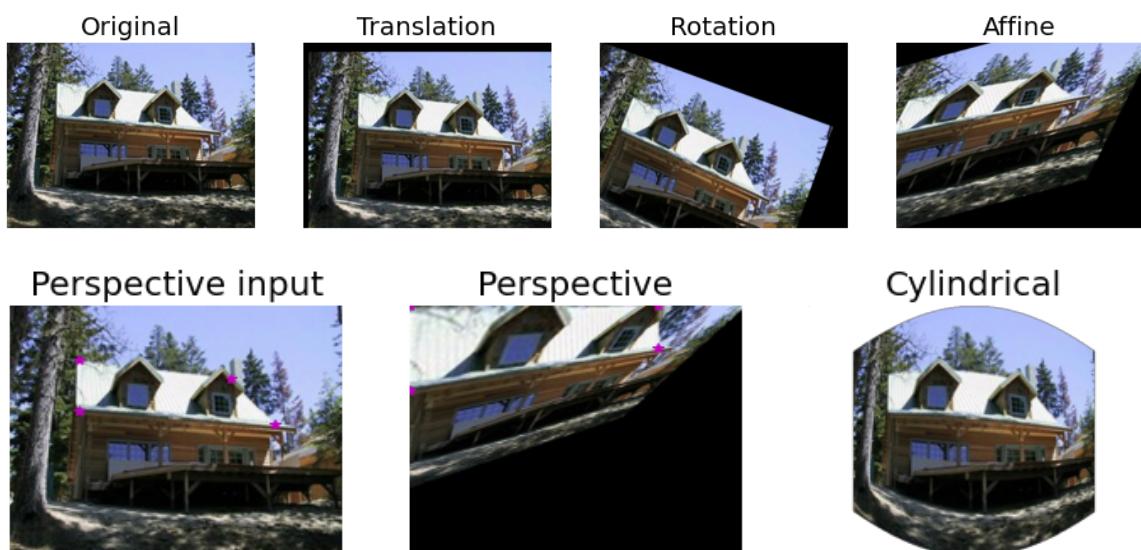
# Camera parameters:
```

```
K = np.array([[200,0,cols/2],[0,400,rows/2],[0,0,1]]) # mock intrinsics  
img_cyl = cylindricalWarp(im, K)
```

(362, 484, 3)

In [5]:

```
plot_images(img, Titles, (1,4), figsize=(16,4), fontsize=20)  
  
plt.figure(figsize=(12,4))  
plt.subplot(131)  
plt.imshow(im), plt.title('Perspective input', fontsize=20)  
plt.plot(pts1[:,0],pts1[:,1], 'm*')  
plt.axis('off')  
plt.subplot(132), plt.imshow(im_perspective), plt.title('Perspective', fontsize=20)  
plt.plot(pts2[:,0],pts2[:,1], 'm*')  
plt.axis('off')  
plt.subplot(133), plt.imshow(img_cyl)  
plt.title('Cylindrical', fontsize=20)  
_ = plt.axis('off')
```



- [Code source \(\[https://opencv-python-tutorials.readthedocs.io/en/latest/py\\\_tutorials/py\\\_imgproc/py\\\_geometric\\\_transformations/py\\\_geometric\\\_transformations.html\]\(https://opencv-python-tutorials.readthedocs.io/en/latest/py\_tutorials/py\_imgproc/py\_geometric\_transformations/py\_geometric\_transformations.html\)\)](https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html) - OpenCV
- [Cylinder code source \(<https://www.morethantechical.com/2018/10/30/cylindrical-image-warping-for-panorama-stitching/>\)](https://www.morethantechical.com/2018/10/30/cylindrical-image-warping-for-panorama-stitching/) - More Than Technical
- [Cylinder coordinates \(\[https://en.wikipedia.org/wiki/Cylindrical\\\_coordinate\\\_system\]\(https://en.wikipedia.org/wiki/Cylindrical\_coordinate\_system\)\)](https://en.wikipedia.org/wiki/Cylindrical_coordinate_system).



## Basic 2D Transformations

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

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

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

Affine

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

Projective  
(Homography)

When can we use a homography matrix to describe the transformation of the scene?

Homography maps between:

- Points that lay on a plane in the real world and their positions in an image.
- Points in two different images that lay on the **same** plane.
- Two images of a 3D object where the camera has been rotated but has not been translated.

Homographies are used to approximate the transformations of far away objects:

- Works fine for small viewpoint changes.



### Excercise 1 (Spring 2020 Q1A)

Write down the homography  $H$  ( $3 \times 3$ ), that maps from image-1 to image-2 in the following cases:

1. Image-2 is a translated image 1 by 100 pixels to the left.
2. Image-2 is a  $3 \times$  zoom of image 1, so that the pixel indexed  $(0, 0)$  stays in its original position.

**Answer:**

1. Image-2 is a translated image 1 by 100 pixels to the left.

We know that for a translation the homography matrix becomes:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

The translation of the image is to the left, which means

$$x_2 = x_1 - 100$$

and the position in the  $y$  axis stays the same:

$$y_2 = y_1$$

Therefore, the homography that describes the given translation is given by:

$$\begin{bmatrix} 1 & 0 & -100 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Sanity check:

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -100 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 - 100 \\ y_1 \\ 1 \end{bmatrix}$$

In [6]:

```
im = cv2.imread('./assets/tut_8_exm.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
rows, cols, d = im.shape
h_T = np.float32([[1, 0, -100],
                  [0, 1, 0],
                  [0, 0, 1]])
im_T = cv2.warpPerspective(im, h_T, (cols, rows))
Titles = ['Original', 'Translation']

plot_images([im, im_T], Titles, (1,2), figsize=(16,4), fontsize=20)
```

Original



Translation



1. Image-2 is a  $3 \times$  zoom of image 1, so that the pixel indexed  $(0, 0)$  stays in its original position.

**Answer:**

The scaled image is different by 1 DOF, since we scale the X-axis and the Y-axis the same amount (same zoom). Going back to the homography matrix, this means only the  $s$  parameter changes:

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In [7]:

```
h_S = np.float32([[3,0,0],[0,3,0],[0,0,1]])
im_S = cv2.warpPerspective(im, h_S,(cols,rows))
Titles = ['Original','ZOOM']

plot_images([im,im_S],Titles,(1,2),figsize=(16,4),fontsize=20)
```



## Computing Parametric Transformations

- [Affine](#)
- [Projective](#)

## Computing Affine Transformation

---

- Assuming we know correspondences, how do we get the transformation?

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \dots \\ x'_i \\ y'_i \\ \dots \end{bmatrix} = \begin{bmatrix} \dots & \dots & 0 & 0 & 0 \\ x_i & y_i & 1 & 0 & 0 \\ 0 & 0 & 0 & x_i & y_i \\ \dots & \dots & \dots & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{bmatrix}$$

$$b = Ah$$

- Solve with Least-squares  $\|Ah - b\|^2$

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

In Python:

```
A_inv = pinv(A)
```

```
h = np.linalg.pinv(A)@b
```

- How many matches (correspondence pairs) do we need to solve?
- Once we have solved for the parameters, how do we compute the coordinates of the corresponding point for any pixel  $(x_{new}, y_{new})$ ?

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$x'_{new} = Hx_{new}$$

## Computing Projective Transformation

- Recall working with homogenous coordinates

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x'_i = u_i/w_i$$

$$y'_i = v_i/w_i$$

- We get the following non-linear equation:

$$x'_i = \frac{h_1 x_i + h_2 y_i + h_3}{h_7 x_i + h_8 y_i + h_9}$$

$$y'_i = \frac{h_4 x_i + h_5 y_i + h_6}{h_7 x_i + h_8 y_i + h_9}$$

- We can re-arrange the equation

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & \dots & -x_i x'_i & -y_i x'_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & \dots & -x_i y'_i & -y_i y'_i & -y'_i \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} \dots \\ 0 \\ 0 \\ \dots \end{bmatrix}$$

- We want to find a vector  $h$  satisfying

$$Ah = 0$$

where  $A$  is full rank. We are obviously not interested in the trivial solution  $h = 0$  hence we add the constraint

$$\|h\| = 1$$

- Thus, we get the homogeneous Least square equation:

$$\arg \min_h \|Ah\|_2^2, s.t \|h\|_2^2 = 1$$

Compute Projective transformation using SVD:

$$\arg \min_h \|Ah\|_2^2, s.t \|h\|_2^2 = 1$$

- Let decompose  $A$  using SVD:  $A = UDV^T$ , where  $U$  and  $V$  are orthonormal matrix, and  $D$  is a diagonal matrix.

- Need a reminder on SVD? [Click Here \(\[https://nbviewer.jupyter.org/github/taldatech/cs236756-intro-to-ml/blob/master/cs236756\\\_tutorial\\\_03\\\_linear\\\_algebra.ipynb\]\(https://nbviewer.jupyter.org/github/taldatech/cs236756-intro-to-ml/blob/master/cs236756\_tutorial\_03\_linear\_algebra.ipynb\)\)](https://nbviewer.jupyter.org/github/taldatech/cs236756-intro-to-ml/blob/master/cs236756_tutorial_03_linear_algebra.ipynb)

- From orthonormality of  $U$  and  $V$ :

$$\begin{aligned} \|UDV^T h\| &= \|DV^T h\| \\ \|V^T h\| &= \|h\| \end{aligned}$$

Hence, we get the following minimization problem:

$$\arg \min_h \|DV^T h\| \text{ s.t. } \|V^T h\| = 1$$

- Substitute  $y = V^T h$ :

$$\arg \min_h \|Dy\| \text{ s.t. } \|y\| = 1$$

- $D$  is a diagonal matrix with decreasing values. Then, it is clear that  $y = [0, 0, \dots, 1]^T$ .
- Therefore, choosing  $h$  to be the last column in  $V$  will minimize the equation.

In Python:

```
(U,D,Vh) = np.linalg.svd(A, False)

h = Vh.T[:, -1]
```

**Some more options to find  $h$ :**

- Lagrange multipliers - [Least-squares Solution of Homogeneous Equations](#) ([http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained\\_lsq.pdf](http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained_lsq.pdf))
- Using EVD (eigenvalue decomposition) on  $A^T A$ .
- If we know our transformation is nearly Affine we can get an approximate solution using linear least squares
- As we said, some of the points may be a good match, and some won't.
- If we use a wrong point in the algorithm, we will get the wrong transformation.
- How can we solve that major problem?
- We can use our prior knowledge on the type of the trasformation, and use points that have strong evidence that are correct.
- For example, we know that homographys preserve lines, so we can try to look for keypoints that lay in a line, and use them to find the transformation.
- How would we find keypoints that lay on a line?

## RANSAC

The RANSAC algorithm is extremely simple, **but** it often:

- Does not produce correct model with user-defined probability
- Outputs an inaccurate model
- Does not handle degeneracies
- Can be sped up (by orders of magnitude)
- Does not guarantee minimum running time
- Needs information about scale of the noise
- Does not handle multiple models efficiently

Many improved algorithms:

- [PROSAC \(<https://dspace.cvut.cz/bitstream/handle/10467/9496/2005-Matching-with-PROSAC-progressive-sample-consensus.pdf;jsessionid=AB7FDDCCABE0CA249D253E7E8F9D1DC7?sequence=1>\)](https://dspace.cvut.cz/bitstream/handle/10467/9496/2005-Matching-with-PROSAC-progressive-sample-consensus.pdf;jsessionid=AB7FDDCCABE0CA249D253E7E8F9D1DC7?sequence=1)
  - Key idea is to assume that the similarity measure predicts correctness of a match
- [Randomized RANSAC \(\[http://www.bmva.org/bmvc/2002/papers/50/full\\\_50.pdf\]\(http://www.bmva.org/bmvc/2002/papers/50/full\_50.pdf\)\)](http://www.bmva.org/bmvc/2002/papers/50/full_50.pdf)
  - Each step take a random subset of the query points and perform RANSAC
- [KALMANSAC \(<https://ieeexplore.ieee.org/abstract/document/1541313>\)](https://ieeexplore.ieee.org/abstract/document/1541313)
- [\(https://en.wikipedia.org/wiki/Random\\_sample\\_consensus#Development\\_and\\_improvements\)](https://en.wikipedia.org/wiki/Random_sample_consensus#Development_and_improvements)
- Estimating homography with RANSAC in OpenCV: `cv2.findHomography(src_pts, dst_pts, cv2.RANSAC)`



### Excercise 2 (Spring 2020 Q1D) - Ransac

We'd like to fit an ellipse to a given set of 200 points,  $X_i = (x_i, y_i), i = 1, \dots, 200$ .

As you know, an ellipse can be described by the implicit equation  $a^T(x^2, xy, y^2, x, y, 1) = 0$  where  $a = (a_1, a_2, a_3, a_4, a_5, a_6)$  is a coefficients vector. The data points might be noisy, such that some of the points may not reside on the perimeter of the ellipse (outliers).

- Describe the RANSAC algorithm needed to fit the ellipse (and find  $a$ ).

**Answer:**

- to define an ellipse, we need at least 5 points ([Why? \(http://www.leepc.com/5pointellipse.html#:~:text=Five%20points%20are%20required%20to%20define%20a%20unique\)](http://www.leepc.com/5pointellipse.html#:~:text=Five%20points%20are%20required%20to%20define%20a%20unique)  
This means  $a$  is defined up to a scale factor.

**Step 1:** Sample 5 points from  $X_i$  randomly. **Step 2:** Fit an ellipse to the points with Least Squares.

- $$\begin{bmatrix} & & \dots & & & \\ x_i^2 & x_i y_i & y_i^2 & x_i & y_i & 1 \\ & \dots & & & & \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} = 0$$

- This can be denoted as  $Sa = 0$ , and solved via SVD under the constraint  $\|a\| = 1$

**Step 3:** Find the points that fit to the model (inliers), i.e. points with a fit error below a threshold chosen in advanced.

**Step 4:** Refine our model according to all the inliers, and count them. The number of inliers is the "score" of the current model (from this iteration).

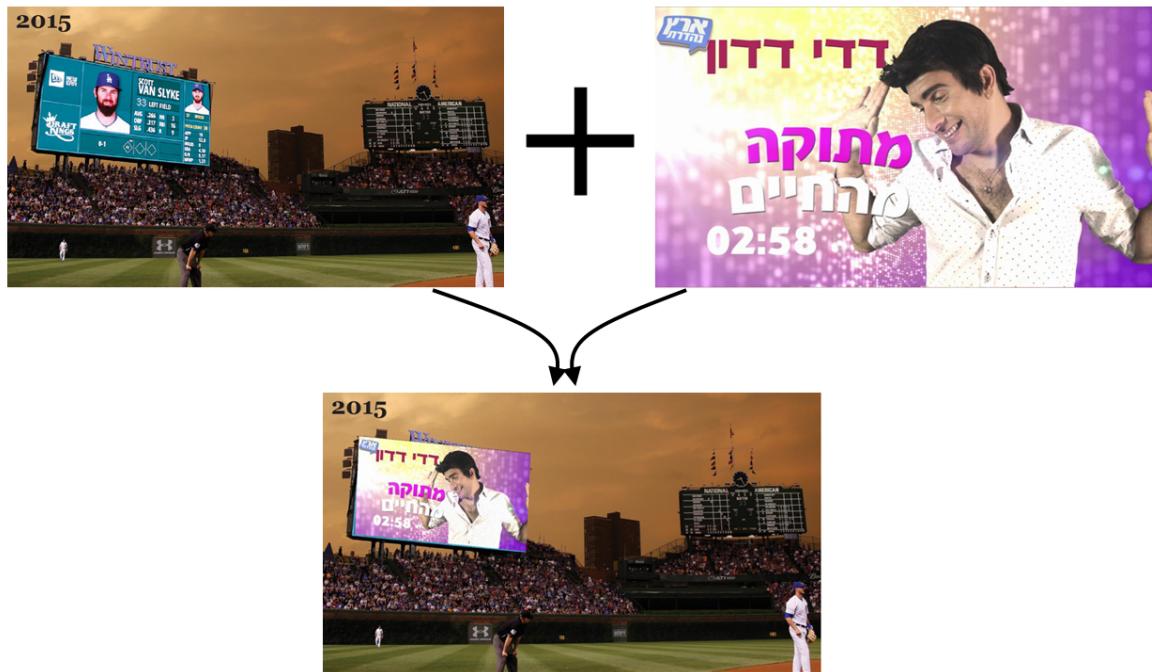
**Step 5:** Repeat 1-4, until the stop criteria has been met.

When do we stop?

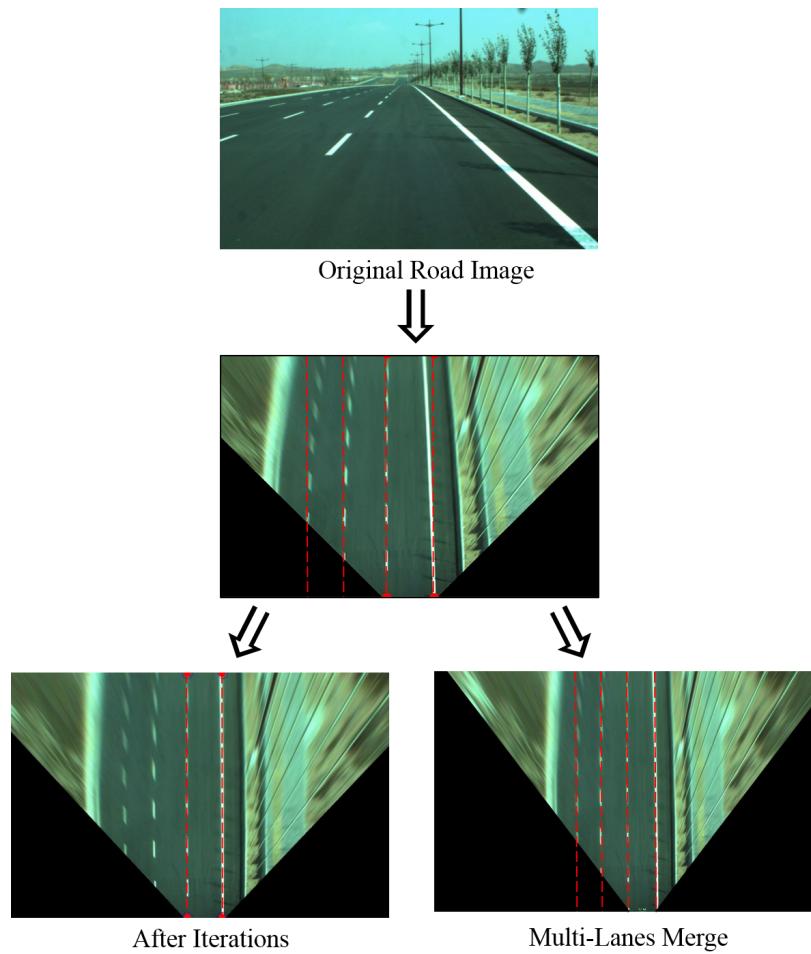
- The percentage of inliers out of all the points is above a certain percentage.
- The amount of inliers is above a certain amount.
- An error threshold (for the LS) is reached.
- A predefined amount of iterations is reached.



## application 1: Planting images into other images

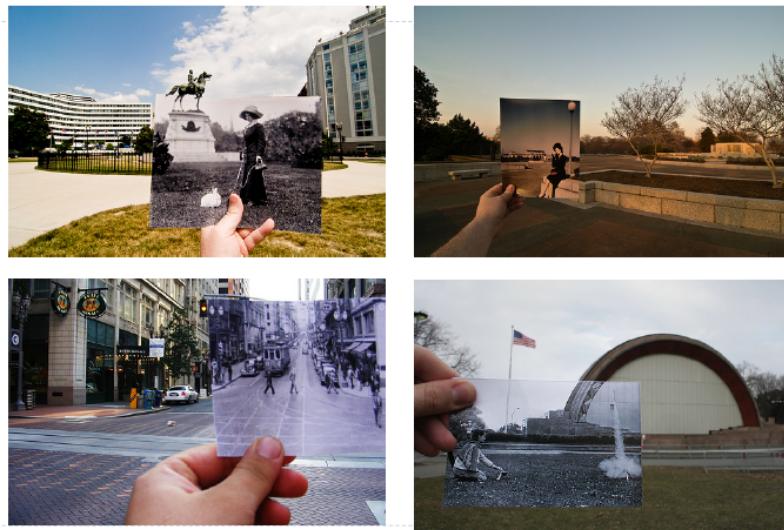


## application 2: BirdEye



- [Image Source \(<https://csyhhu.github.io/2015/07/09/IPM/>\)](https://csyhhu.github.io/2015/07/09/IPM/)

### application 3: Looking into the past



- [Image Source from Flickr \(<http://blog.flickr.net/en/2010/01/27/a-look-into-the-past/>\)](http://blog.flickr.net/en/2010/01/27/a-look-into-the-past/)

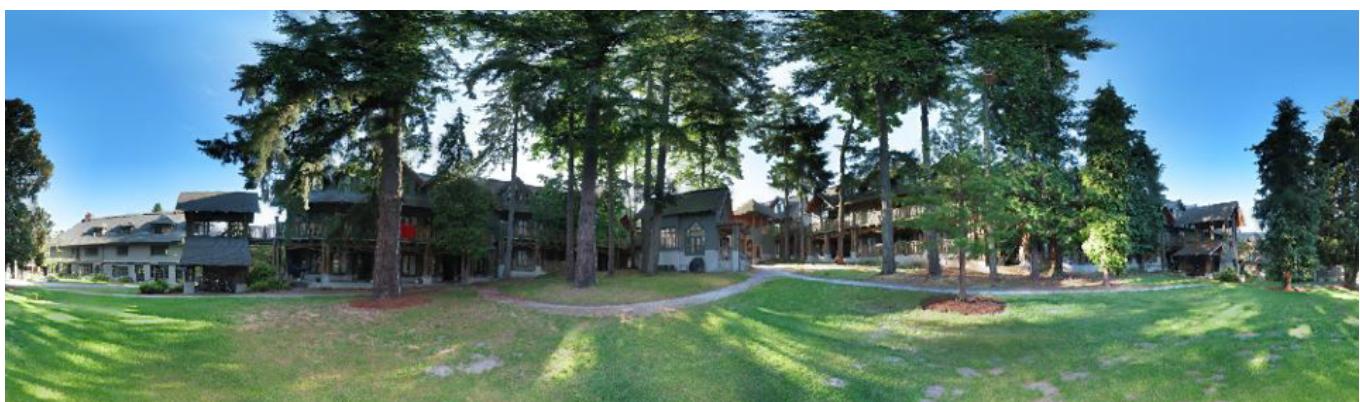
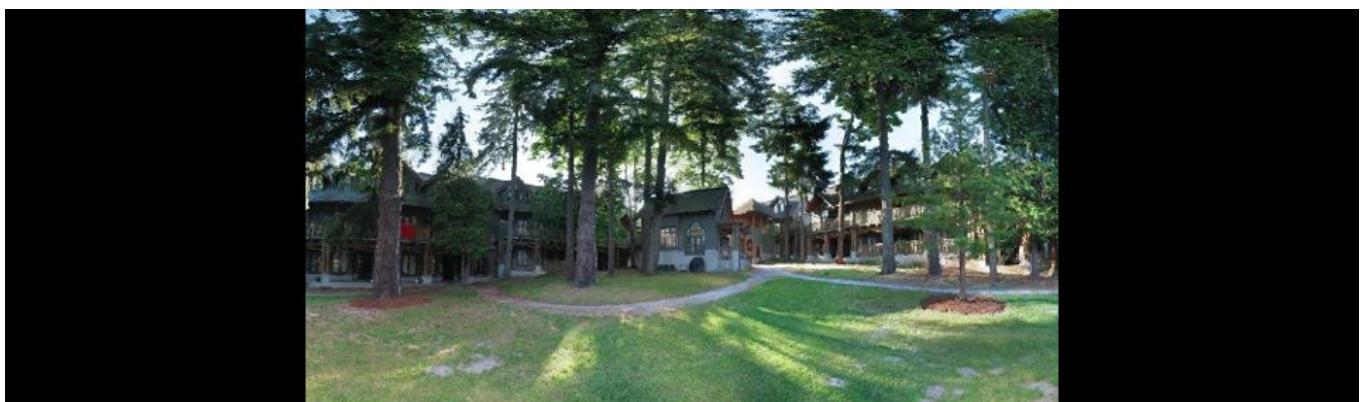
### application 4: Panorama



Obtain a wider angle view by combining multiple images



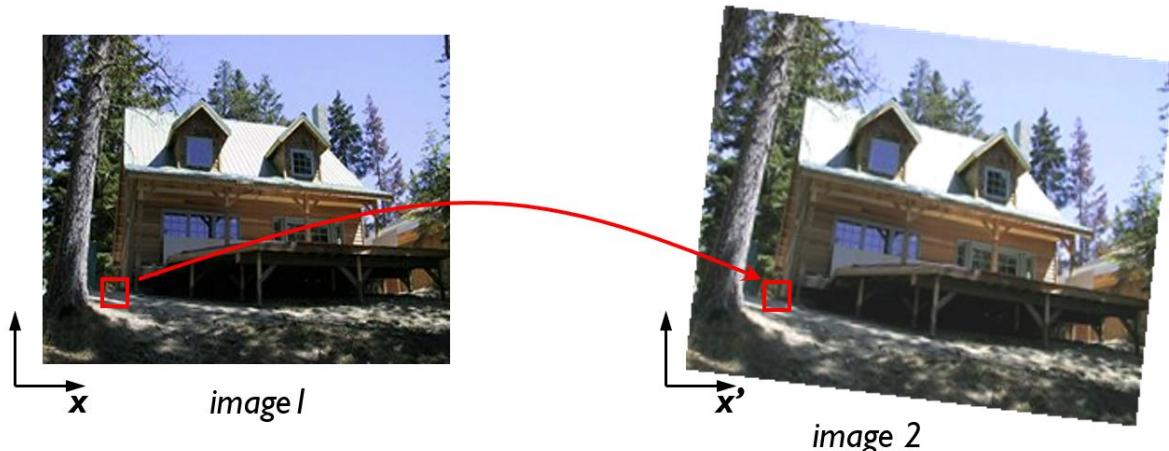
- [Warping](#)
- [Image Blending \(Feathering\)](#)





## Warp - What we need to solve?

- Given source and target images, and the transformation between them, how do we align them?
- Send each pixel  $x$  in image1 to its corresponding location  $x'$  in image 2



### Forward Warping

- What if pixel lands “between” two pixels?
- Answer: add “contribution” to several pixels and normalize (splatting)

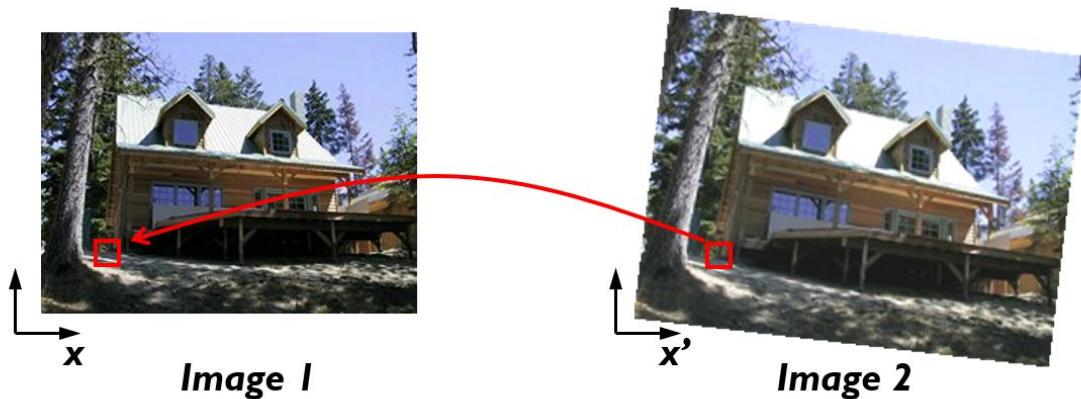


- Limitation: Holes (some pixels are never visited)



### Inverse Warping

- For each pixel  $x'$  in image 2 find its origin  $x$  in image 1
- Problem: What if pixel comes from “between” two pixels?



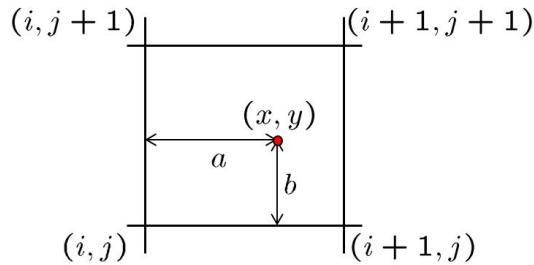
- Answer: interpolate color value from neighbors



## Bilinear Interpolation

---

Sampling at  $f(x, y)$ :



$$\begin{aligned}f(x, y) = & (1 - a)(1 - b)f[i, j] \\& + a(1 - b)f[i + 1, j] \\& + abf[i + 1, j + 1] \\& + (1 - a)bf[i, j + 1]\end{aligned}$$

Python:

- `interp2d()` - <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp2d.html>  
[\(https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp2d.html\)](https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp2d.html)
- Inverse warping in OpenCV: `cv2.warpPerspective(im, *, *, cv2.WARP_INVERSE_MAP)`



### Excercise 3 (Spring 2020 Q1B)

- 
1. Describe forward and backward wraping.
  2. For which out of the two cased before will there be a difference between using inverse or forward warping?
  3. For the case which a difference exists, what type of warping should be used and why?

reminder:

- Image-2 is a translated image 1 by 100 pixels to the left
- Image-2 is a 3x zoom of image 1, so the index (0,0) stay in place

## Answer

1. Read the tutorial :)
2. For the translation there will be no difference between forward/inverse warping, while for the zooming there will be a difference - we'll get holes.

To understand why, let's denote the pixel of image  $k$  at the indices  $(i, j)$  as  $x_{ij}^k$ .

In the first case, each pixel is moved to the left by a whole number, therefore for forward warping we follow the warp defined by:

$$x_{(i-100)j}^2 = x_{ij}^1$$

And for inverse warping we use:

$$x_{ij}^2 = x_{(i+100)j}^1$$

There's no need to interpolate anything here - because the pixel moved by an integer, and thus there is no difference.

In the second case we get for forward warping:

$$x_{(3i)(3j)}^2 = x_{ij}^1$$

And for inverse warping we get an interpolation dependant on  $\alpha, \beta$ :

$$x_{ij}^2 = (1 - \alpha)(1 - \beta)x_{\left(\left\lfloor \frac{i}{3} \right\rfloor\right)\left(\left\lfloor \frac{j}{3} \right\rfloor\right)}^1 + \alpha(1 - \beta)x_{\left(\left\lfloor \frac{i}{3} \right\rfloor + 1\right)\left(\left\lfloor \frac{j}{3} \right\rfloor\right)}^1 + \alpha\beta x_{\left(\left\lfloor \frac{i}{3} \right\rfloor\right)\left(\left\lfloor \frac{j}{3} \right\rfloor + 1\right)}^1 + (1 - \alpha)\beta x_{\left(\left\lfloor \frac{i}{3} \right\rfloor + 1\right)\left(\left\lfloor \frac{j}{3} \right\rfloor + 1\right)}^1$$

We can see that for the forward case we get "holes" in image 2, for example  $x_{1,1}^2$  has no origin in image 1.

1. The backward warping will be of course better, since as we saw no holes will be created.

In [8]:

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

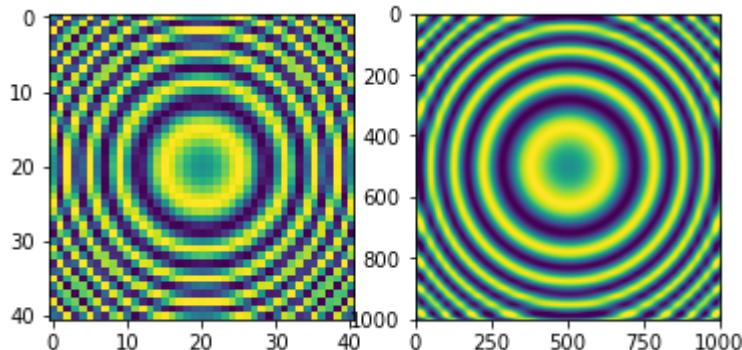
# original samples
x = np.arange(-5.01, 5.01, 0.25)
y = np.arange(-5.01, 5.01, 0.25)
xx, yy = np.meshgrid(x, y)
z = np.sin(xx**2+yy**2)

# interpolated samples
xnew = np.arange(-5.01, 5.01, 1e-2)
ynew = np.arange(-5.01, 5.01, 1e-2)
```

In [9]:

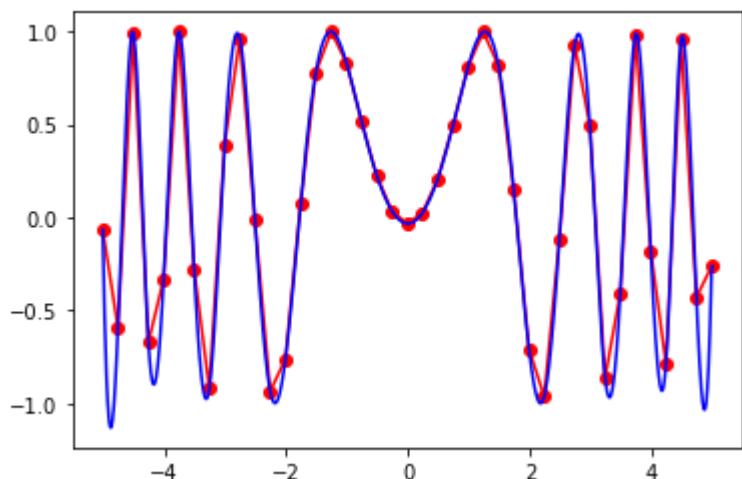
```
## Cubic
f = interpolate.interp2d(x, y, z, kind='cubic')
znew = f(xnew, ynew)

plt.figure()
plt.subplot(121)
plt.imshow(z,vmin=-1,vmax=1)
plt.subplot(122)
plt.imshow(znew,vmin=-1,vmax=1)
plt.show()
```



In [10]:

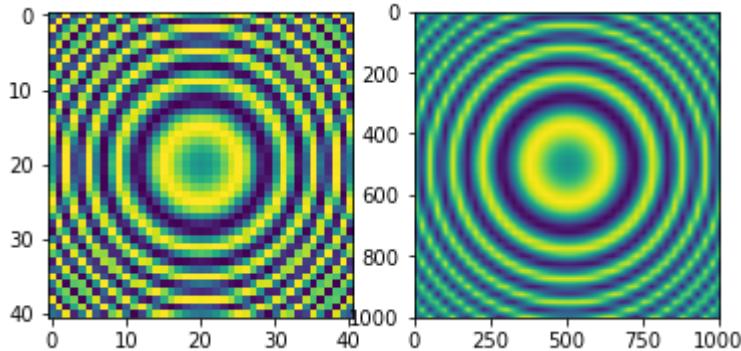
```
plt.figure()
plt.plot(x, z[0, :], 'ro-', xnew, znew[0, :], 'b-')
plt.show()
```



In [11]:

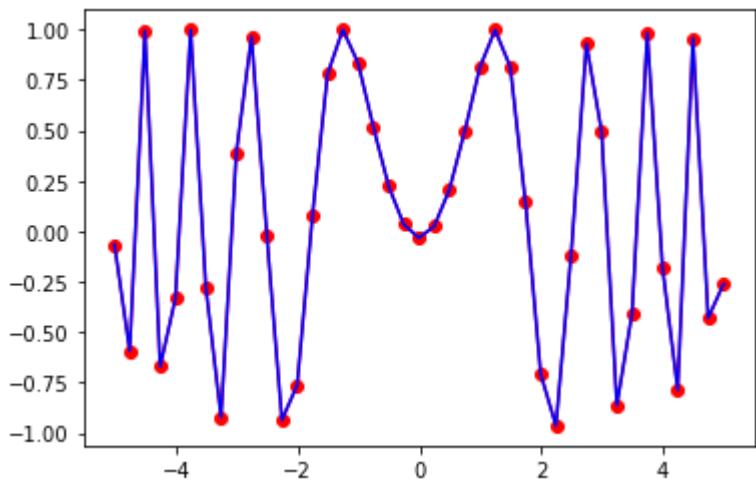
```
##linear
f2 = interpolate.interp2d(x, y, z, kind='linear')
znew2 = f2(xnew, ynew)

plt.figure()
plt.subplot(121)
plt.imshow(z,vmin=-1,vmax=1)
plt.subplot(122)
plt.imshow(znew2,vmin=-1,vmax=1)
plt.show()
```



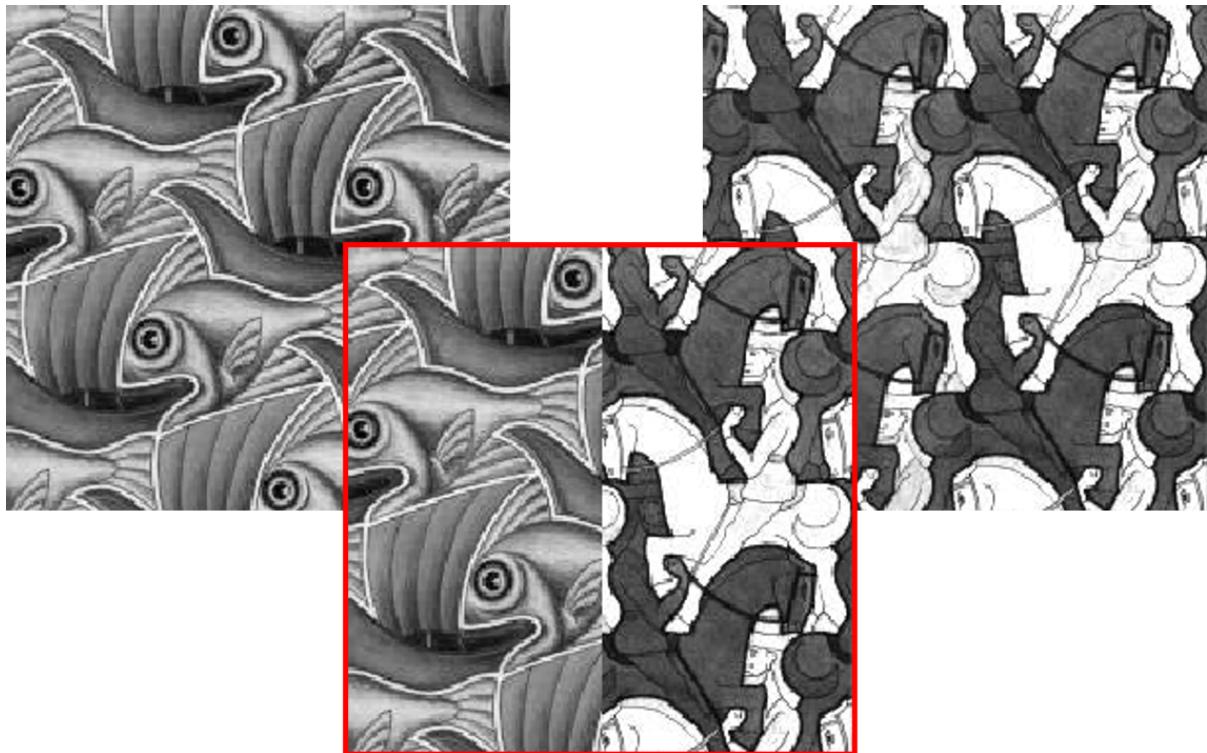
In [12]:

```
plt.figure()
plt.plot(x, z[0, :], 'ro-', xnew, znew2[0, :], 'b-')
plt.show()
```





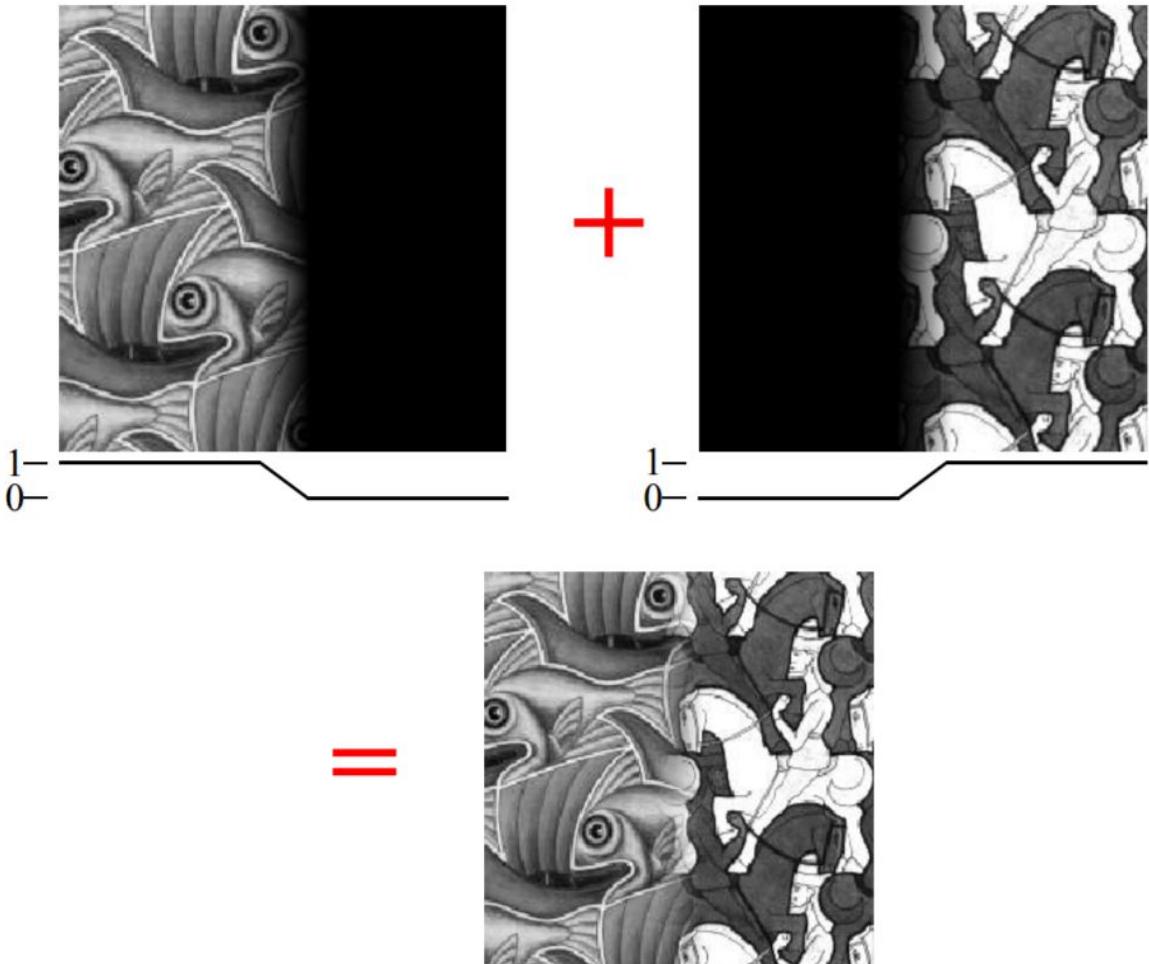
## Image Blending



- Alpha blending
- Pyramid blending

## **Alpha Blending**

---



## **Pyramid Blending:**

---

1. Build a Gaussian pyramid for each image
2. Build the Laplacian pyramid for each image
3. Decide/find the blending border (in the example: left half belongs to image 1, and right half to image 2 -> the blending border is `cols/2`)
  - Split by index, or
  - Split using a 2 masks (can be weighted masks)
4. Construct a new mixed pyramid - mix each level separately according to (3)
5. Reconstruct a blend image from the mixed pyramid

In [13]:

```
def create_pyrs(A,B):
    # generate Gaussian pyramid for A
    G = A.copy()
    gpA = [G]
    for i in range(6):
        G = cv2.pyrDown(G)
        gpA.append(G)
    # generate Gaussian pyramid for B
    G = B.copy()
    gpB = [G]
    for i in range(6):
        G = cv2.pyrDown(G)
        gpB.append(G)
    # generate Laplacian Pyramid for A
    lpA = [gpA[5]]
    for i in range(5,0,-1):
        GE = cv2.pyrUp(gpA[i])
        L = cv2.subtract(gpA[i-1],GE)
        lpA.append(L)
    # generate Laplacian Pyramid for B
    lpB = [gpB[5]]
    for i in range(5,0,-1):
        GE = cv2.pyrUp(gpB[i])
        L = cv2.subtract(gpB[i-1],GE)
        lpB.append(L)
    return lpA,lpB
```

In [14]:

```
def blend_images(A,B):
    lpA,lpB = create_pyrs(A,B)
    # Now add left and right halves of images in each Level
    LS = []
    for la,lb in zip(lpA,lpB):
        rows,cols,dpt = la.shape
        ls = np.hstack((la[:,0:int(cols/2)], lb[:,int(cols/2):])) #mixing can also be done with a mask
        LS.append(ls)
    # now reconstruct
    ls_ = LS[0]
    for i in range(1,6):
        ls_ = cv2.pyrUp(ls_)
        ls_ = cv2.add(ls_, LS[i])
    # image with direct connecting each half
    real = np.hstack((A[:,0:int(cols/2)],B[:,int(cols/2):]))
    return real, ls_
```

In [15]:

```
def switch_texture(A,B):
    lpA,lpB = create_pyrs(A,B)
    # Now add left and right halves of images in each level
    LS = []
    for la,lb in zip(lpA,lpB):
        rows,cols,dpt = la.shape
        ls = np.hstack((la[:,0:int(cols/2)], lb[:,int(cols/2):])) #mixing can also be
done with a mask
    LS.append(ls)
# now reconstruct
ls_ = lpA[0]
for i in range(1,6):
    ls_ = cv2.pyrUp(ls_)
    ls_ = cv2.add(ls_, lpB[i])
# image with direct connecting each half
real = np.hstack((A[:, :int(cols/2)],B[:, int(cols/2):]))
return real, ls_
```

In [16]:

```
def alpha_blend(A,B,MASK):
    A = np.float32(A)
    B = np.float32(B)
    return np.uint8(A*MASK), np.uint8(A*MASK+B*(1-MASK))
```

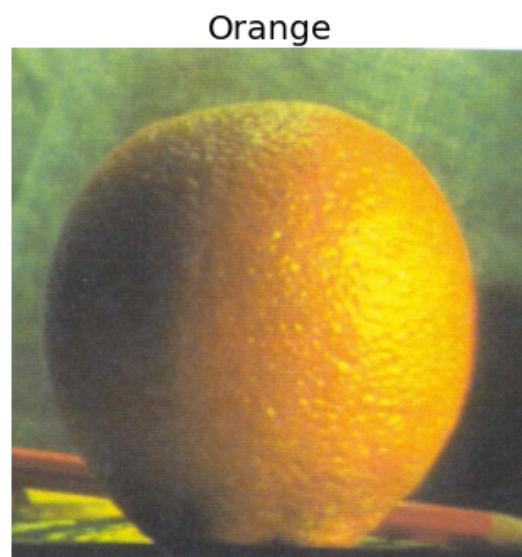
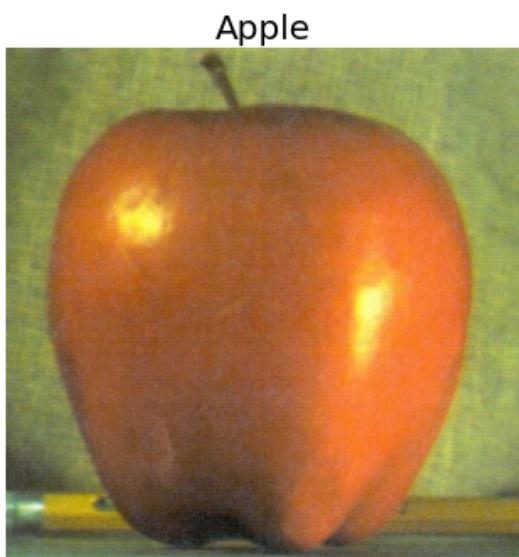
In [17]:

```
A = cv2.imread('./assets/apple.jpg')
B = cv2.imread('./assets/orange.jpg')
real,ls_ = blend_images(A,B)

A_ = cv2.cvtColor(A,cv2.COLOR_BGR2RGB)
B_ = cv2.cvtColor(B,cv2.COLOR_BGR2RGB)
ls_ = cv2.cvtColor(ls_,cv2.COLOR_BGR2RGB)
real = cv2.cvtColor(real,cv2.COLOR_BGR2RGB)
```

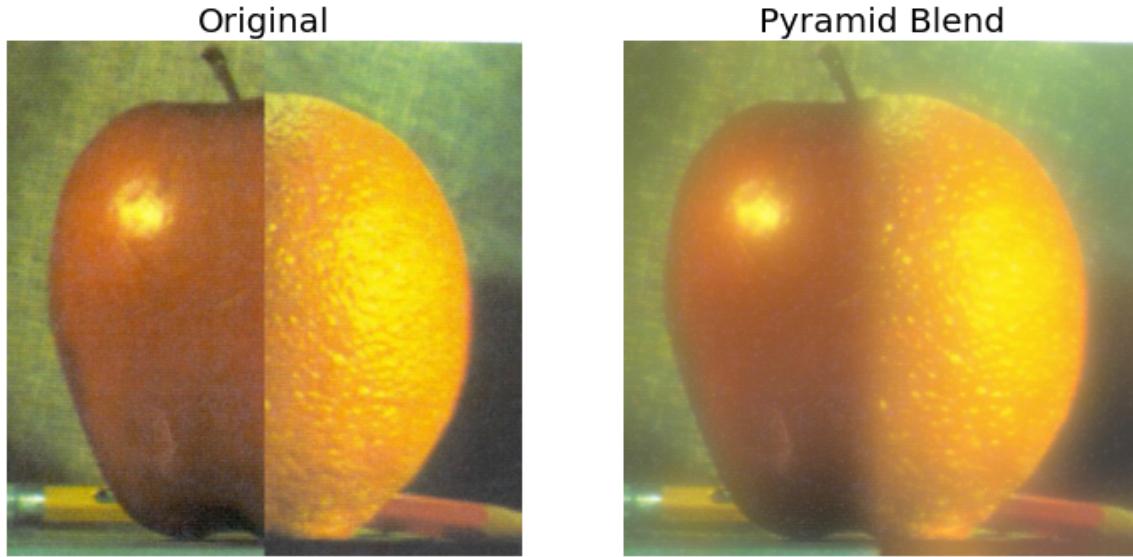
In [18]:

```
plot_images([A_,B_],['Apple','Orange'],(1,2),figsize=(12,12),fontsize=20)
```



In [19]:

```
plot_images([real,ls_],['Original','Pyramid Blend'],(1,2),figsize=(12,12),fontsize=20)
```



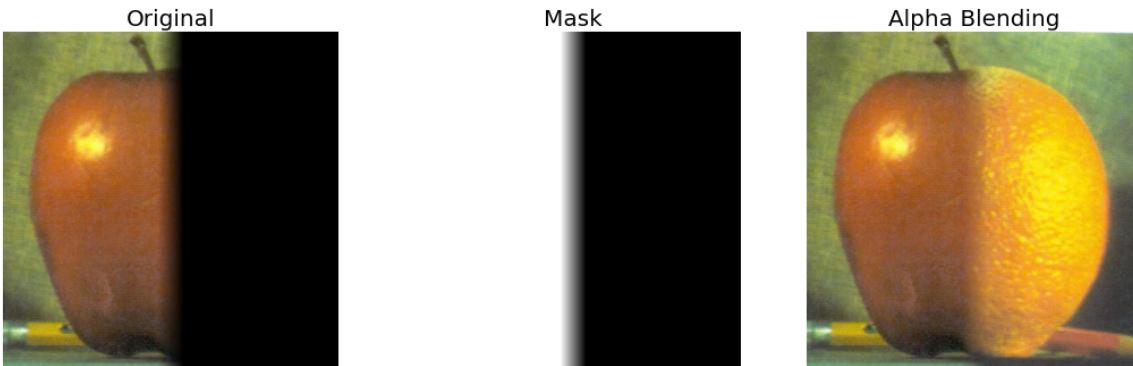
- Alpha blending example:

In [20]:

```
MASK = np.ones_like(A,np.float32)
rows,cols,dpt = MASK.shape
w=20
v_dec = np.linspace(1,0,2*w)
MASK[:,int(cols/2):]=0
MASK[:,(int(cols/2)-w):(int(cols/2)+w)]=np.tile(np.reshape(v_dec,[1,-1,1]),[rows,1,3])
real,ls_ = alpha_blend(A,B,MASK)
ls_ = cv2.cvtColor(ls_,cv2.COLOR_BGR2RGB)
real = cv2.cvtColor(real,cv2.COLOR_BGR2RGB)
```

In [21]:

```
plot_images([real,MASK,ls_],['Original','Mask','Alpha Blending'],(1,3),figsize=(18,12),
fontsize=20)
```



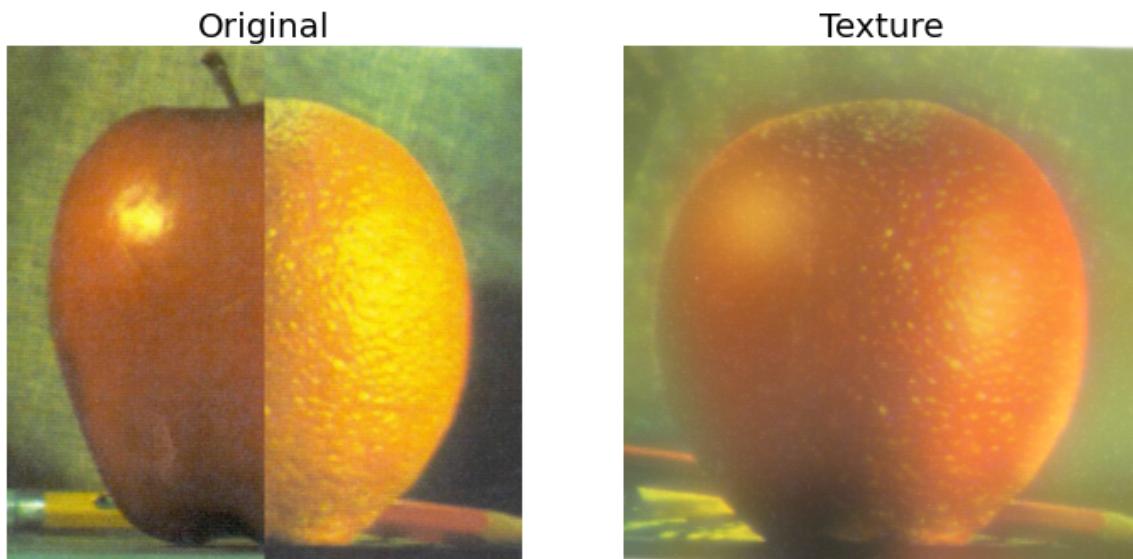
- Stylize image using pyramids:

In [22]:

```
real,ls_ = switch_texture(A,B)
ls_ = cv2.cvtColor(ls_,cv2.COLOR_BGR2RGB)
real = cv2.cvtColor(real,cv2.COLOR_BGR2RGB)
```

In [23]:

```
plot_images([real,ls_],['Original','Texture'],(1,2),figsize=(12,12),fontsize=20)
```



### **Blending Improvements**

- Many algorithms have different variations of combining alpha and pyramid blending (different masks for different frequencies)
- Find the boundaries using **segmentation**



### **Panorama - Summary**

- Detect features
- Compute transformations between pairs of frames
- Can Refine transformations using RANSAC
- Warp all images onto a single coordinate system
- Find mixing borders (e.g. using segmentation)
- Blend



# Transformations in Deep Learning

- Can we incorporate transformations in the pipeline of a deep learning algorithm?
  - Moreover, can we accelerate these transformations by performing them on a GPU?
- YES!

## Kornia - Computer Vision Library for PyTorch



- [Kornia is a differentiable computer vision library for PyTorch \(<https://github.com/kornia/kornia>\)](https://github.com/kornia/kornia)
  - That means you can have gradients for the transformations!
- Inspired by OpenCV, this library is composed by a subset of packages containing operators that can be inserted within neural networks to train models to perform image transformations, epipolar geometry, depth estimation, and low-level image processing such as filtering and edge detection that operate directly on tensors.
- Check out `kornia.geometry` - <https://kornia.readthedocs.io/en/latest/geometry.html>  
[\(https://kornia.readthedocs.io/en/latest/geometry.html\)](https://kornia.readthedocs.io/en/latest/geometry.html)
- [Warp image using perspective transform](https://kornia.readthedocs.io/en/latest/tutorials/warp_perspective.html)  
[\(https://kornia.readthedocs.io/en/latest/tutorials/warp\\_perspective.html\)](https://kornia.readthedocs.io/en/latest/tutorials/warp_perspective.html)



PyTorch

```
import torch
import kornia

frame: torch.Tensor = load_video_frame(...)

out: torch.Tensor = (
    kornia.rgb_to_grayscale(frame)
)
```





## Recommended Videos

---



### Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

## Video By Subject

- Homography
  - Image geometry and planar homography - [ENB339 lecture 9: Image geometry and planar homography](https://www.youtube.com/watch?v=fVJeJMWZcq8) (<https://www.youtube.com/watch?v=fVJeJMWZcq8>)
  - Homography - [Homography in computer vision explained](https://www.youtube.com/watch?v=MlaIWymLCD8) (<https://www.youtube.com/watch?v=MlaIWymLCD8>)
- Transformations - [Lect. 5\(1\) - Linear and affine transformations](https://www.youtube.com/watch?v=4I2S5Xhf24o) (<https://www.youtube.com/watch?v=4I2S5Xhf24o>)
- Matching Local Features
  - SIFT - [CSCI 512 - Lecture 12-1 SIFT](https://www.youtube.com/watch?v=U0wqePj4Mx0) (<https://www.youtube.com/watch?v=U0wqePj4Mx0>)



## Credits

---

- EE 046746 Spring 2022 - [Moshe Kimhi](https://www.linkedin.com/in/moshekimhi/) (<https://www.linkedin.com/in/moshekimhi/>), [Hila Manor](https://github.com/HilaManor) (<https://github.com/HilaManor>)
- EE 046746 Spring 2020 - [Dahlia Urbach](https://il.linkedin.com/in/dahlia-urbach-97a816123) (<https://il.linkedin.com/in/dahlia-urbach-97a816123>)
- Slides - Elad Osherov (Technion), Simon Lucey (CMU)
- Multiple View Geometry in Computer Vision - Hartley and Zisserman - Section 2
- [Least-squares Solution of Homogeneous Equations](http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained_lsq.pdf) ([http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained\\_lsq.pdf](http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained_lsq.pdf)) - Center for Machine Perception - Tomas Svoboda
- [Computer vision: models, learning and inference](http://www.computervisionmodels.com/) (<http://www.computervisionmodels.com/>) , Simon J.D. Prince - Section 15.1
- [Computer Vision: Algorithms and Applications](https://www.springer.com/gp/book/9781848829343) (<https://www.springer.com/gp/book/9781848829343>) - Richard Szeliski - Sections 2,4,6, 9 ([Free for Technion students via remote library](https://piazza.com/class/k81fyipit1f75kx?cid=122) (<https://piazza.com/class/k81fyipit1f75kx?cid=122>))
- Icons from [Icon8.com](https://icons8.com) (<https://icons8.com>) - <https://icons8.com> (<https://icons8.com>)