Step by Step Face Swap

Sylwester Brzeczkowski

TrustStamp

About me and Trust Stamp

Sylwester Brzeczkowski - husband, father, python developer at Trust Stamp

sylwek@truststamp.net

/sylwekb

/in/sylwekb

TrustStamp

Agenda

- Face detection
- Facial landmarks detection
- Finding face border
- Approximating nonlinear operations
- Finding triangles for image transformation
- Blending images together
- Stabilization

TrustStamp

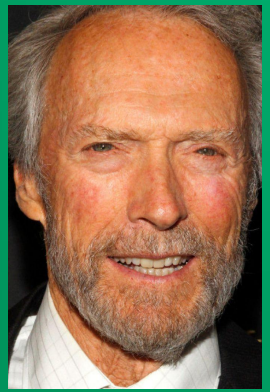
Goal of this talk

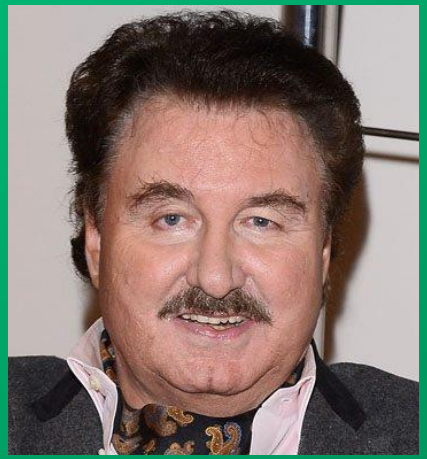
- You more or less know what mechanics are behind apps like Snapchat.
- You know how to tackle the problem of face swap.
- You know that most of the heavy lifting is already implemented in dlib or OpenCV (with 1 exception)
- You know why face swapping fails so often ;)

TrustStamp

Meta info

- The ideas for solutions comes mainly from: learnopencv.com, pyimagesearch.com and wikipedia ;)
- Examples are written in python with OpenCV, dlib and numpy
- Save questions for the Q&A, unless you notice a real bu  it. Remember the slide number!
- We will use pictures of following stars in the examples:



TrustStamp

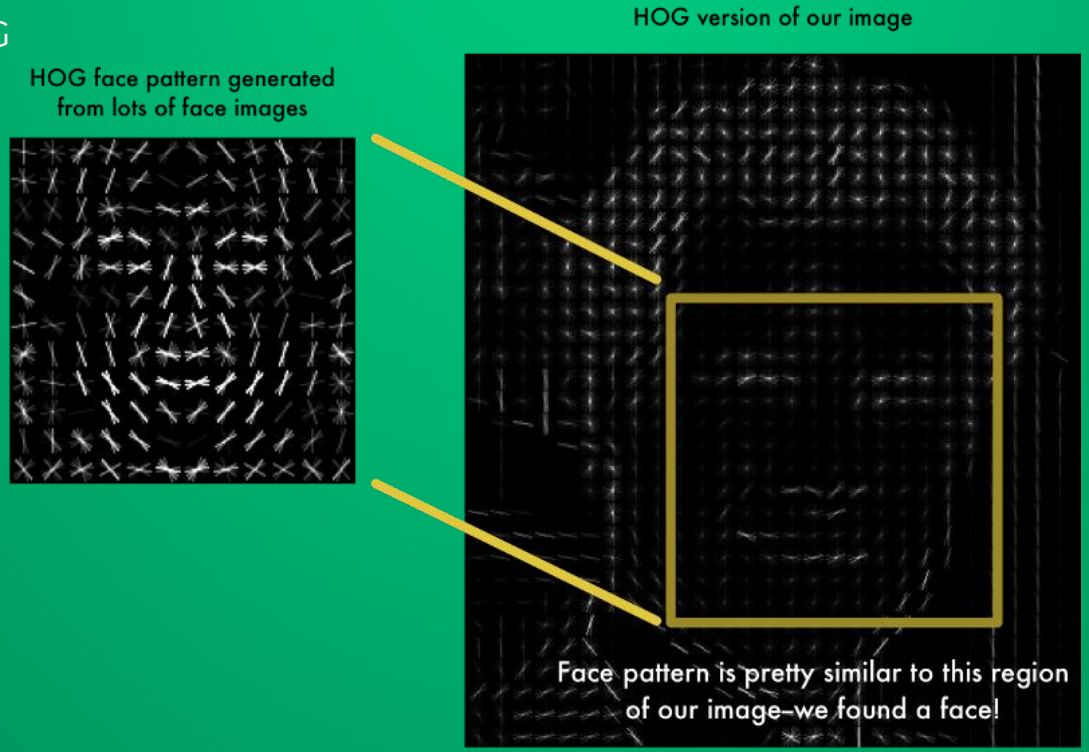
0. Face Detection

HOG

TrustStamp

0. Face Detection

HOG

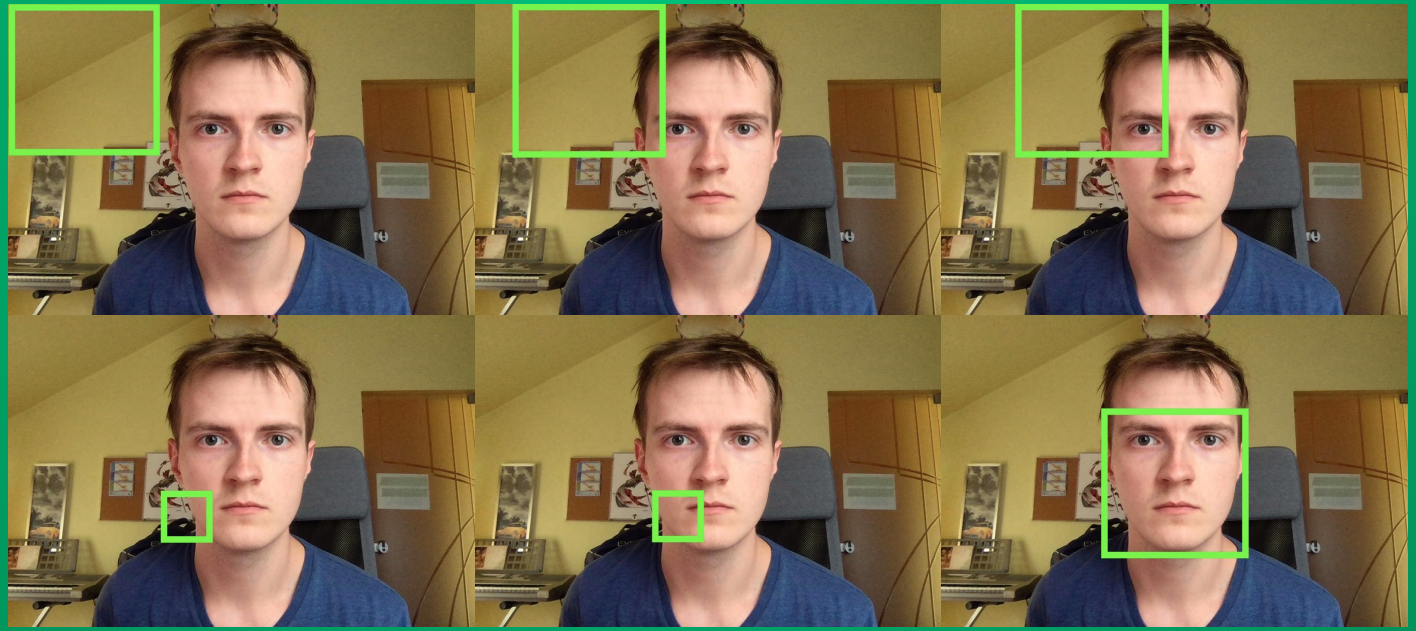


Source: https://datascience.stackexchange.com/questions/21926/application-of-histogram-of-oriented-gradients-in-colored-image

TrustStamp

0. Face Detection

classify patches



TrustStamp

0. Face Detection

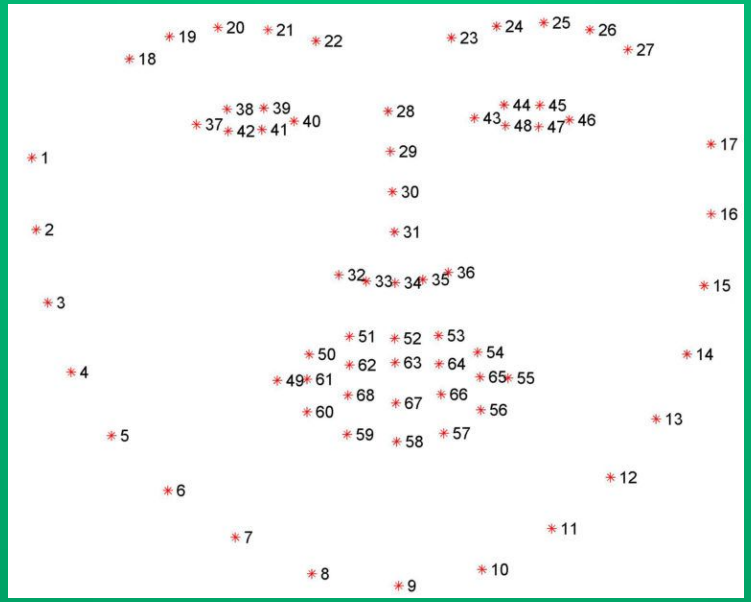
dlib

```
import cv2
import dlib

detector = dlib.get_frontal_face_detector()


img = cv2.imread('image.jpg')

rectangles = detector(img)
```

TrustStamp

1. Facial Landmarks Detection

aka Face Alignment

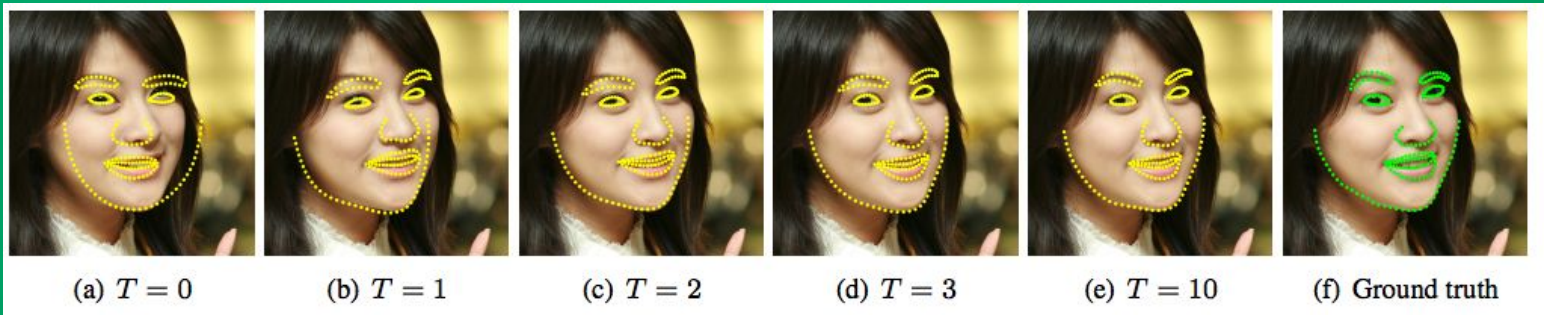


One Millisecond Face Alignment:
https://www.pyimagesearch.com/wp-content/uploads/2017/04/facial_landmarks_68markup-768x619.jpg

TrustStamp

1. Facial Landmarks Detection

aka Face Alignment



One Millisecond Face Alignment:
https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Kazemi_One_Millisecond_Face_2014_CVPR_paper.pdf

TrustStamp

1. Facial Landmarks Detection

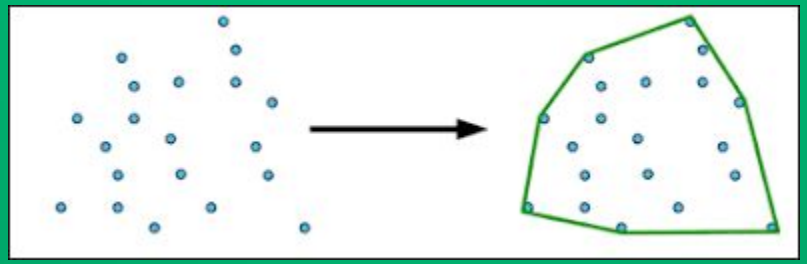
aka Face Alignment

```
import cv2
import dlib

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")


img = cv2.imread('image.jpg')

rectangles = detector(img)

face = max(rectangles, key=lambda r: r.area())
landmarks = predictor(img, face)
```

TrustStamp

2. Find face border

Convex hull

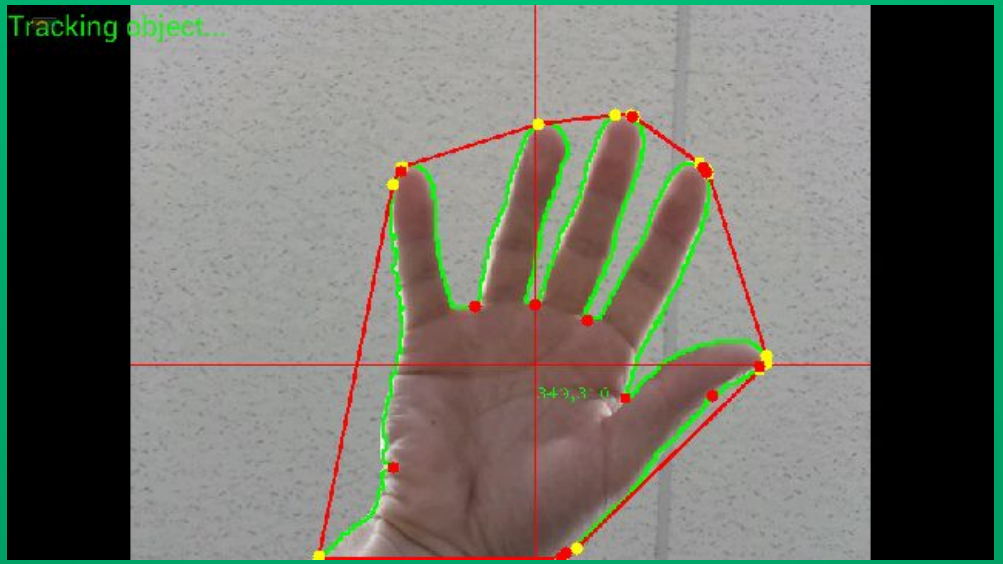


Source: http://mind.cs.byu.edu/courses/312/projects/project2_files/ConvexHull_python.php

TrustStamp

2. Find face border

Convex hull vs contour



Source:
https://stackoverflow.com/questions/18143077/computer-vision-filtering-convex-hulls-and-convexity-defects-with-opencv

TrustStamp

2. Find face border

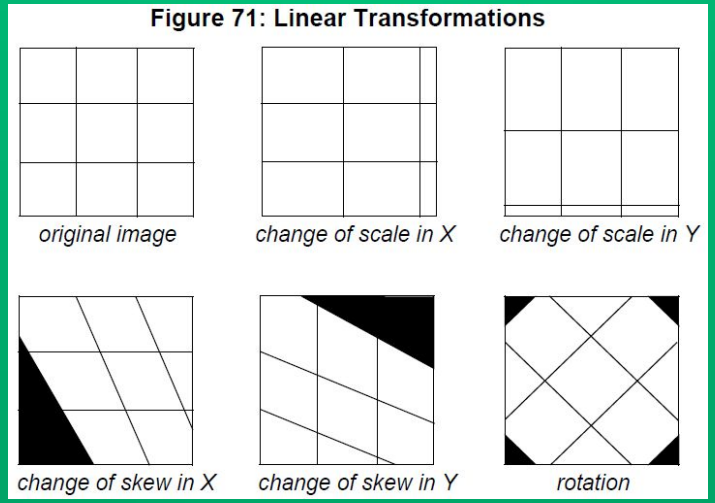
Convex hull

```
…

landmarks = predictor(img, face)

points = [(p.x, p.y) for p in landmarks.parts()]

hull = cv2.convexHull(np.array(points), returnPoints=False)
```

TrustStamp

3. Approximating nonlinear operations with linear ones.

Affine transformation



Source http://jun.hansung.ac.kr/SIIS/Notes/FieldGuideNote07.html

TrustStamp

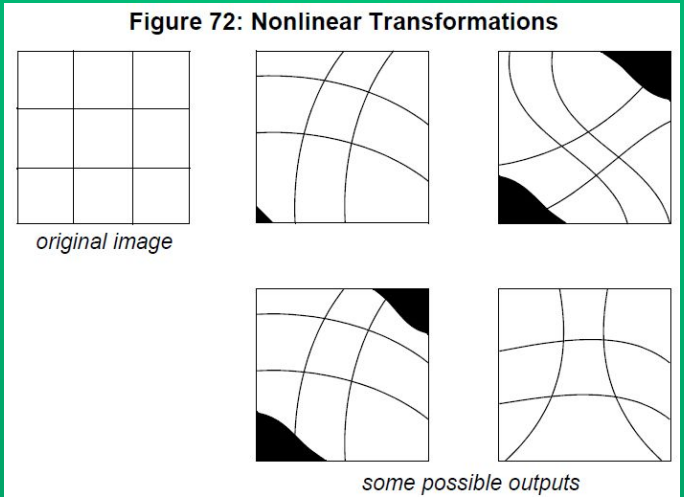
3. Approximating nonlinear operations with linear ones.

Affine transformation

TrustStamp

3. Approximating nonlinear operations with linear ones.

Nonlinear transform

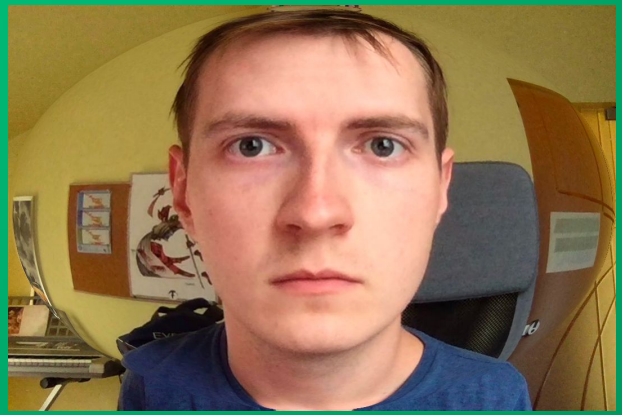


Source http://jun.hansung.ac.kr/SIIS/Notes/FieldGuideNote07.html

TrustStamp

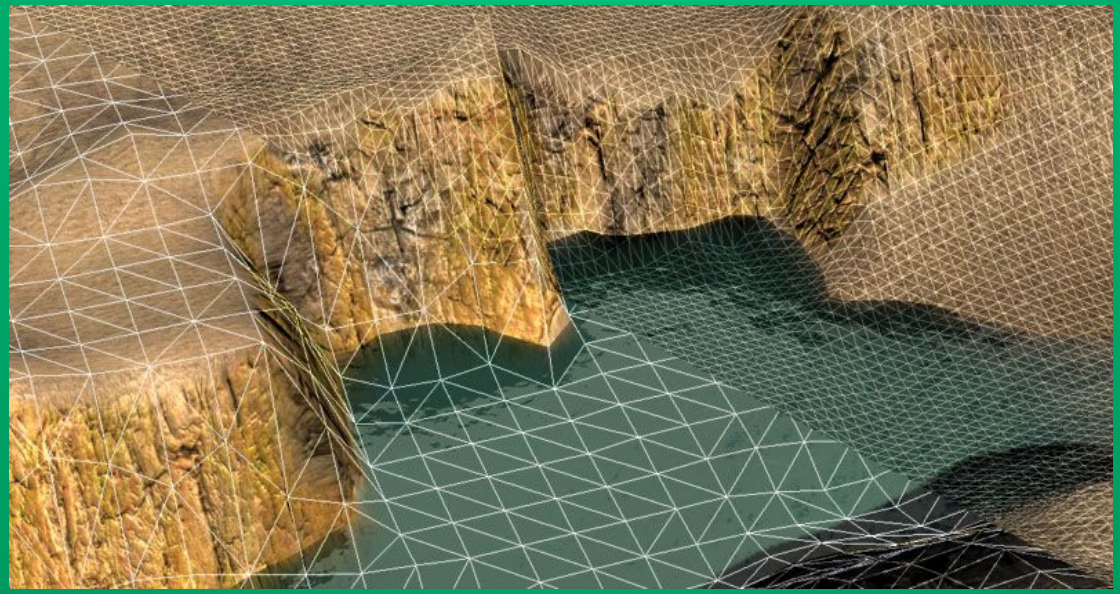
3. Approximating nonlinear operations with linear ones.

Nonlinear transform



TrustStamp

3. Approximating nonlinear operations with linear ones.



TrustStamp

3. Approximating nonlinear operations with linear ones.

There is no such operation in OpenCV :/ Steps:

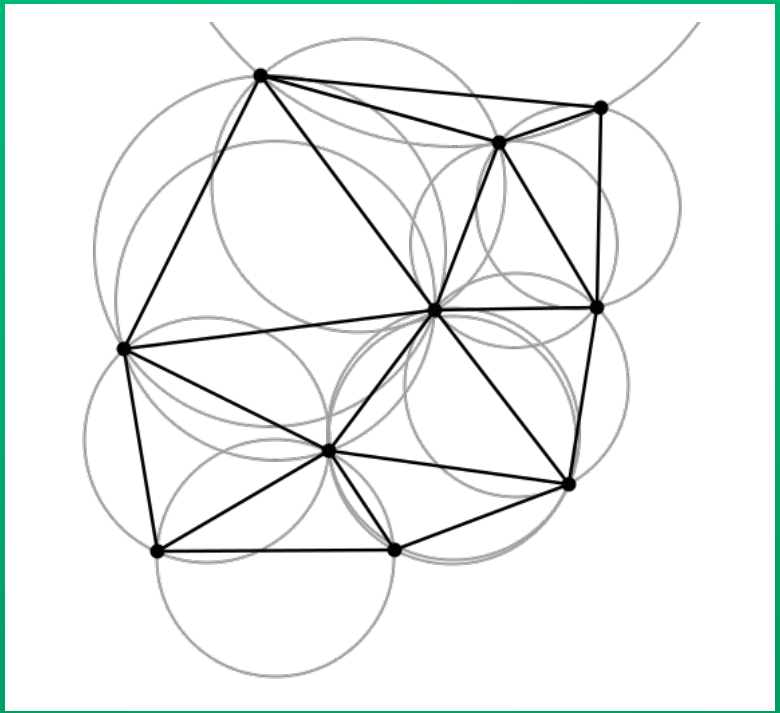
1. Get coordinates of input triangle and output triangle.
2. Find bounding rectangles around those triangles with cv2.boundingRect().
3. Crop out these bounding rectangles out of the image.
4. Create a mask with 1.0 inside triangle, and 0.0 outside.
5. getAffineTransform() returns matrix of how to transform from triangle 1 to triangle 2
6. warpAffine() uses this matrix to transform bounding rect 1 to bounding rect 2
7. Multiply output of warpAffine with mask (inside triangle (1.0) stays, outside (0.0) is out)
8. Put the warped fragment back into image.

Source: https://www.learnopencv.com/warp-one-triangle-to-another-using-opencv-c-python/

TrustStamp

4. Finding triangles

Delaunay Triangulation



TrustStamp

4. Finding triangles

Delaunay Triangulation

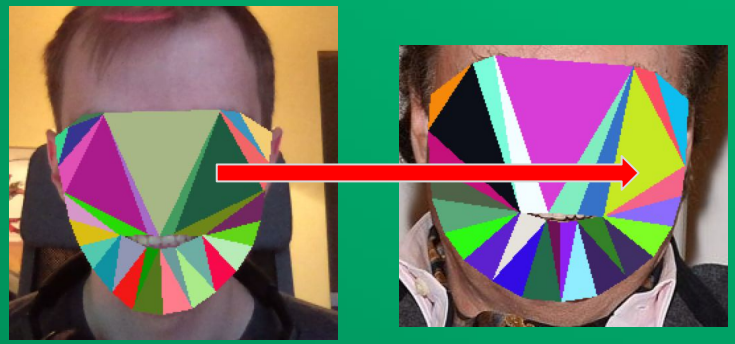
```
…
mouth_points = [[60], [61], [62], [63], [64], [65], [66], [67]]

hull = np.concatenate([hull, mouth_points_indexes])

rect = (0, 0, img.shape[1], img.shape[0])
subdiv = cv2.Subdiv2D(rect)

subdiv.insert(hull)

triangles = subdiv.getTriangleList()
```

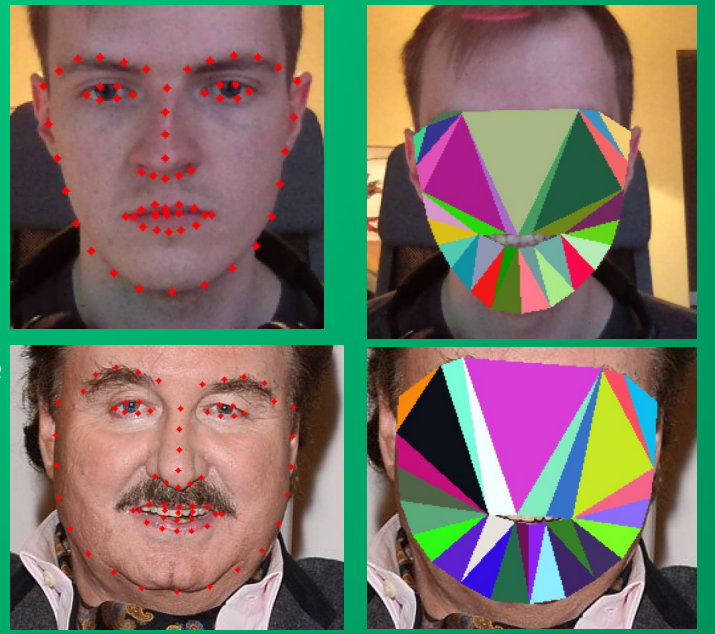
TrustStamp

3 + 4. Transforming face into another.

Step by step.

1. Get some points:
   a. Face border (convex hull)
   b. Add mouth points (specific indexes)
2. Find Delaunay Triangulation on the first image
3. Find corresponding triangles on the second image
4. Transform every triangle from 1. img to 2nd. img





TrustStamp

5. Blending images together

Seamless cloning

Seamless Poisson cloning

- Given vector field *v* (pasted gradient), find the optimize:

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

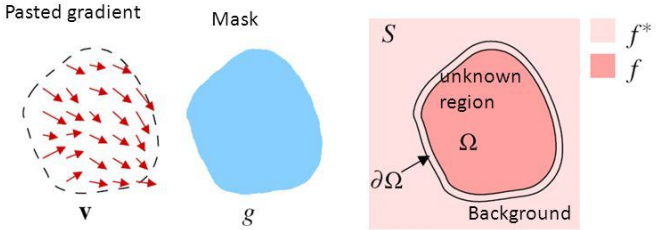


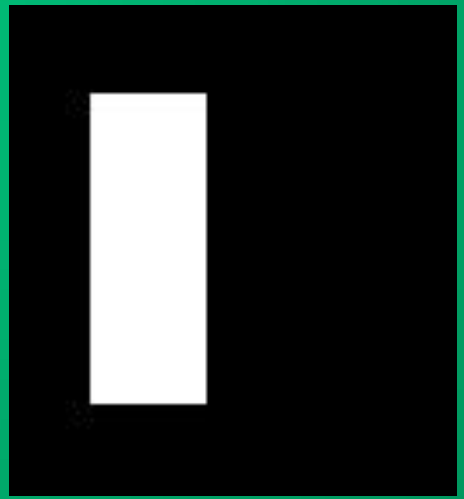
Figure 1: **Guided interpolation notations**. Unknown function f interpolates in domain Ω the destination function f^*, under guidance of vector field $\mathbf{v}$, which might be or not the gradient field of a source function g.

Poisson Image Editing
http://www.irisa.fr/vista/Papers/2003_siggraph_perez.pdf

TrustStamp

5. Blending images together

Seamless cloning

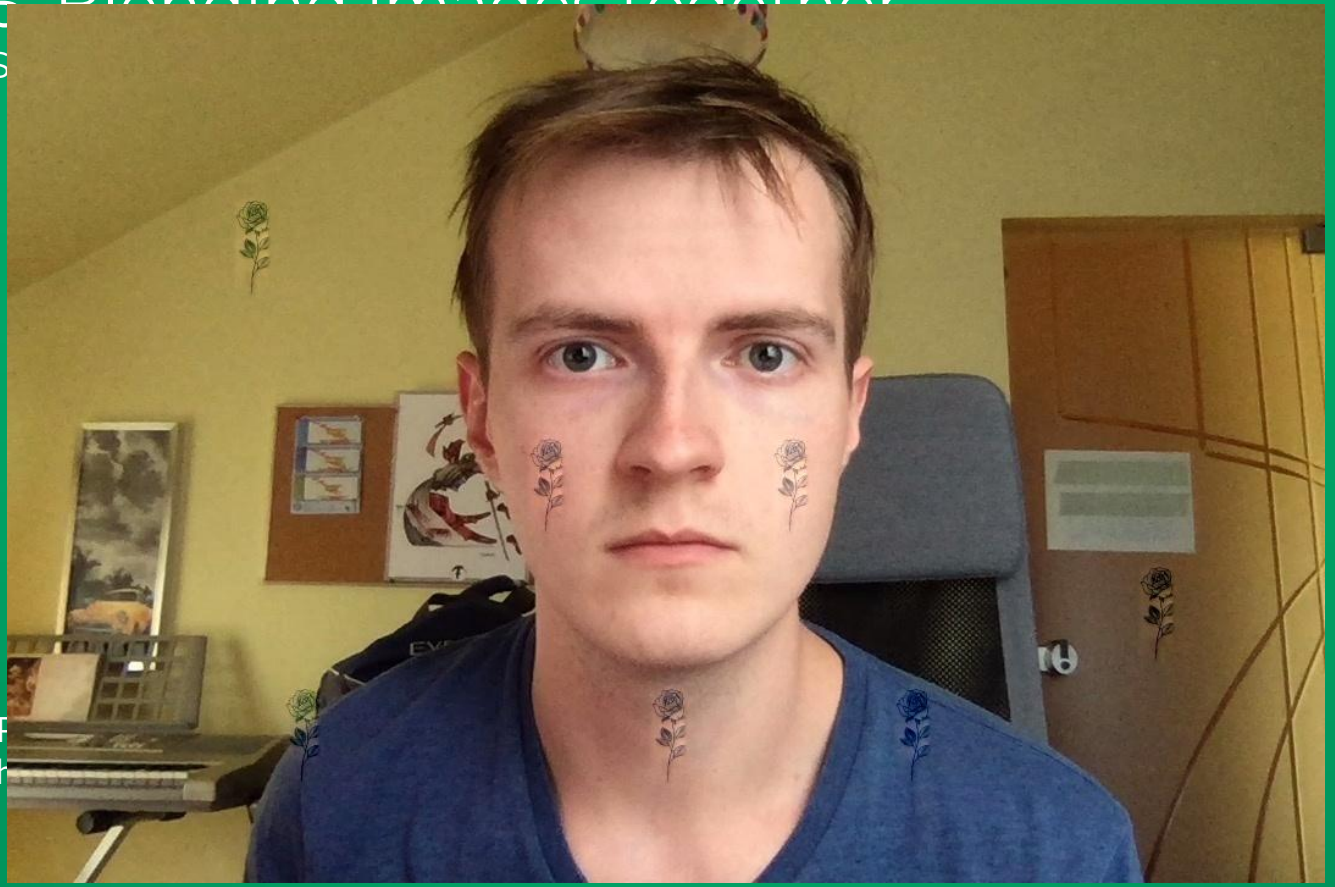


Poisson Image Editing
http://www.irisa.fr/vista/Papers/2003_siggraph_perez.pdf

TrustStamp

5. Blending images together

S



F
h

TrustStamp

5. Blending images together

Seamless cloning

```
dest = cv2.imread("image1.jpg")

source = cv2.imread("image2.jpg")

mask = np.zeros(source.shape[:2], dtype=np.float32)
rect = [(23, 25), (55, 112)]
mask = np.uint8(cv2.rectangle(mask, *rect, (1.0, 1.0, 1.0), -1) * 255)

center = 641, 395

cloned = cv2.seamlessClone(source, dest, mask, center,
cv2.MIXED_CLONE)
```

TrustStamp

6. Stabilization

Optical Flow with Lucas–Kanade method

Optical flow tries to predict future position of a point.

TrustStamp

6. Stabilization

Optical Flow with Lucas–Kanade method

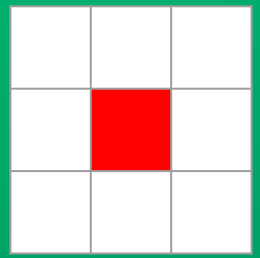
Optical flow tries to predict future position of a point.

$(u, v) = (dx / dt, dy / dt)$

Assumption 1.: Brightness Constancy Assumption

$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$

Additional Assumptions: the flow is essentially constant in a local neighbourhood of the pixel under consideration (Lucas–Kanade)



Source: https://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method

TrustStamp

6. Stabilization

Optical Flow in OpenCV

```
hull_next, *_ = cv2.calcOpticalFlowPyrLK(img_gray_prev, img_gray, hull_prev, hull)
```

TrustStamp

6. Stabilization

landmarks detection and optical flow prediction

To run smoothly we want to use both information: detected landmarks and optical flow prediction.

```
points[t][i] = 0.3  * landmarks[t][i] + 0.7 predicted[t][i]
```

TrustStamp

Summary

What do you know?

1. You know the basic steps for the face swap:
    a. face detection,
    b. landmarks detection,
    c. warping triangles,
    d. seamless cloning
    e. optical flow.
2. You more or less know how these methods work in general.
3. You have the basic idea if they are hard or not to implement in python.

TrustStamp

THANKS!

Sylwester Brzeczkowski - husband, father, python developer at Trust Stamp.

sylwek@truststamp.net

/sylwekb

/in/sylwekb

Example's code is here: https://github.com/sylwekb/face_swap/

TrustStamp