Face Swap 🤦🏻

Júlio Raphael de Oliveira Silva, 2016085924

Lucas Moreira e Silva Alves, 20170027336

Mateus Fonseca Henriques, 2016019061

Renan Almeida Goes Vieira de Melo, 20170021539

Introdução

Gostaríamos de fazer um projeto que abarcasse os conteúdos passsados na disciplina e apresentasse desafios na medida certa para testar nossos conhecimentos. Com soluções contemporâneas e utilizando bibliotecas no estado-da-arte de inteligência artifical para realizar atividades secundárias no projeto (*DeepFace*), que fora usada para realizar operações que seriam muito custosas caso não houvesse uma biblioteca de alto nível para realizar essas operações (no caso encontrar a menor distância de uma imagem para com um banco de dados e determinar características como gênero, etnia, idade e emoção)

Outrossim, utilizamos técnicas demasiadamente relacionadas ao processamento digital de imagens, com as operações utilizando OpenCV para conseguir operar sobre as imagens e conseguir fazer um *face swap* harmônico e diminuindo a quantidade de falhas

Métodos

Durante o desenvolvimento do trabalho, foi necessário a compreensão de diversos conhecimentos relacionados a disciplina de processamento de digital de imagens, tivemos auxílio de algumas bibliotecas para abertura das imagens, criação de arrays, colorir imagens e geração delas, realizar cálculos matemáticos e transformação de matrizes, harmonização, entre imagens entre outras funcionalidades. Essas bibliotecas podem ser observadas abaixo:

- Foram utilizadas duas redes neurais já treinadas para a detecção de pontos faciais e para detecção de faces,com a utilização da biblioteca dlib. Essa biblioteca multi-plataforma contém softwares open-source independentes que incluem processamento digital de imagens e deep learning. Como a própria biblioteca já possui uma rede neural treinada para detecção de faces, optamos por utilizá-la, porém para a detecção de pontos faciais foi necessário baixar um rede treinada e utilizar o dlib apenas para gerar os resultados retornados pela função shape predictor

- Para a leitura e manipulação de imagens foi utilizada a biblioteca OpenCV, a qual permite uma gama abrangente de funcionalidades que se fizeram necessárias no projeto, não só para os objetivos principais, mas também para a visualização e interface. Por outro lado a biblioteca numpy foi utilizada para a manipulação de arrays em python, visto que esta é a estrutura de dados utilizada para manuseamento de imagens

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from deepface import DeepFace
import pandas as pd
import dlib
import warnings
import os
from IPython.display import Markdown as md
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
warnings.filterwarnings('ignore')
import tensorflow as tf
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

Desenvolvimento

Definir função para apresentar múltiplas imagens, ou apenas uma

In [2]:
```
def show_multiple_images(pixels_array, rows, imgs_per_row, title):
    f, ax = plt.subplots(rows, imgs_per_row, figsize=(12,4))

    for idx, img in enumerate(pixels_array):
        ax[idx].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

    f.suptitle(title, fontsize=16)
    plt.tight_layout()
    plt.show()

def show_image(pixels, title=''):
    f, ax = plt.subplots(1, 1, figsize=(12,4))
    plt.imshow(cv2.cvtColor(pixels, cv2.COLOR_BGR2RGB))
    f.suptitle(title, fontsize=16)
    plt.tight_layout()
    plt.show()
```

Identificar faces e carregar um método pré-treinado que vai pegar os principais pontos faciais do indivíduo, no caso, 68 pontos

In [3]:
```
frontal_face_detector = dlib.get_frontal_face_detector()
frontal_face_predictor = dlib.shape_predictor('./utils/shape_predictor_68_fac
```

In [4]:
```
img_1_path = './input/input-img/obama.jpeg'
```

Utilizando a primeira imagem como comparação, atravessaremos o banco de dados com imagens de famosos e utilizaremos a imagem que possui a maior ou menor distância em comparação com a imagem de origem, dependendo do parâmetro passado para função

Esta etapa será feita com a biblioteca DeepFace, uma biblioteca de estado da arte para reconhecimento facial e Deep Learning desenvolvido pelo facebook, mesclando modelos como: VGG-Face, Google FaceNet, OpenFace, Facebook DeepFace, DeepID e DLib. Usaremos ela para capturar distâncias entre a imagem de input e extrair outras informações das imagens de origem e da imagem pós-swap

Verifica no banco de dados faces similares à que foi dada como input e dentre as semelhantes, pode-se selecionar a mais ou menos parecidas

In [5]:
```
df = DeepFace.find(img_path=img_1_path, db_path='./input/famous-br-db', enfor
```

```
Analyzing:   0%|          | 0/1 [00:00<?, ?it/s]
WARNING: Representations for images in  ./input/famous-br-db  folder were prev
iously stored in  representations_vgg_face.pkl . If you added new instances af
ter this file creation, then please delete this file and call find function ag
ain. It will create it again.
There are  139  representations found in  representations_vgg_face.pkl
Analyzing: 100%|██████████| 1/1 [00:02<00:00,  2.78s/it]
find function lasts  5.0544209480285645  seconds
```

In [6]:
```
df
```

Out[6]:

	identity	VGG-Face_cosine
0	./input/famous-br-db/chico-buarque.jpg	0.283071
1	./input/famous-br-db/pele.jpg	0.302331
2	./input/famous-br-db/dunga.jpg	0.321168
3	./input/famous-br-db/ratinho.jpg	0.335312
4	./input/famous-br-db/renato_gaucho.png	0.338069
5	./input/famous-br-db/supla.jpg	0.344691
6	./input/famous-br-db/frank_aguiar.jpg	0.346024
7	./input/famous-br-db/renato_russo.jpg	0.351736
8	./input/famous-br-db/alexandre_frota.jpg	0.352265
9	./input/famous-br-db/rodinei.jpg	0.374776
10	./input/famous-br-db/lineu.jpg	0.384169
11	./input/famous-br-db/deltan.jpg	0.386080
12	./input/famous-br-db/indio2.jpg	0.389901
13	./input/famous-br-db/jo.jpg	0.397318

In [7]:
```
def fetch_img(df, mode='Min'):
    if mode == 'Min':
        idx = df['VGG-Face_cosine'].idxmin()
    else:
        idx = df['VGG-Face_cosine'].idxmax()

    return df.iloc[idx]
```

In [8]:
```
second_image_path = fetch_img(df, 'Min')['identity']
```

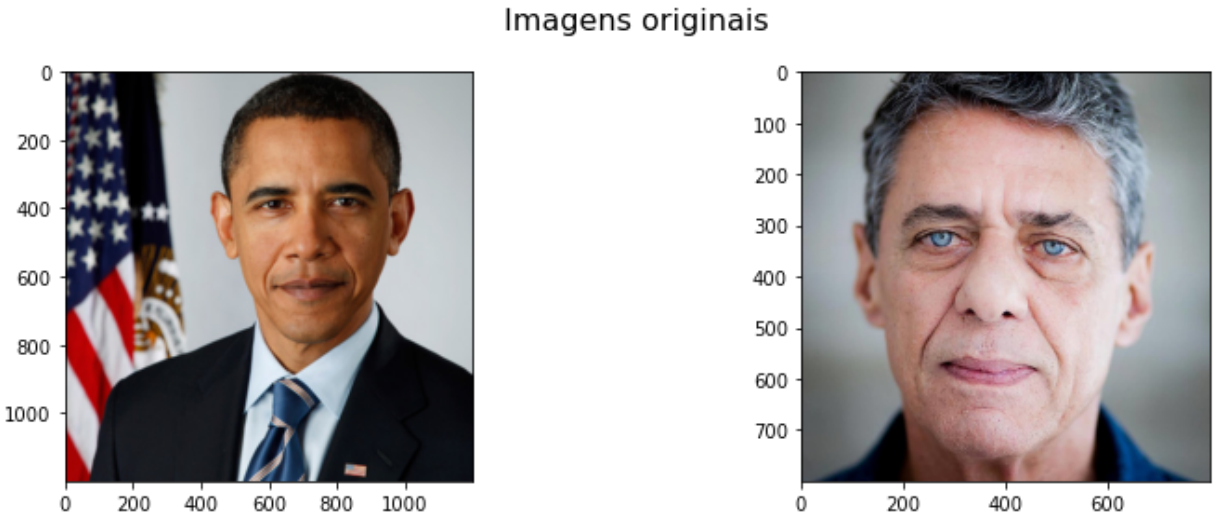
Ler as imagens originais e convertê-las para preto-e-branco, facilitando o processo de preenchimento do canvas posteriormente, também, será feita uma analíse das imagens de entrada

In [9]:
```
img_1, img_2 = cv2.imread(img_1_path), cv2.imread(second_image_path)
img_1_copy, img_2_copy = img_1, img_2
img_1_original, img_2_original = img_1.copy(), img_2.copy()
img_1_grayscale, img_2_grayscale = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY), cv
img_1_stats, img_2_stats = DeepFace.analyze(img_1_original, actions = ['age',
                           DeepFace.analyze(img_2_original, actions = ['age'
```

```
Action: emotion: 100%|██████████| 4/4 [00:09<00:00,  2.49s/it]
Action: emotion: 100%|██████████| 4/4 [00:09<00:00,  2.31s/it]
```

In [10]:
```
def emojify_gender(gender):
    if gender == 'Man':
        return '♂'
    else:
        return '♀'

def generate_statistics_markdown(img_1_stats, img_2_stats=None):
    if img_2_stats is not None:
        return md(f"""
### Análise das faces utilizando DeepFace ⚡

<hr>

| <h2>Característica 🔵</h2> |  <h2>[ Imagem 1 📸]</h2> | <h2>[ Imagem 2 📷]</h
|--------------------|----------|:--------:|
| <h3>Gênero</h3> | <h3>{emojify_gender(img_1_stats['gender'])}</h3> | <h3>{e
| <h3>Idade</h3> |   <h4>{img_1_stats['age']}  </h4>    |    <h4> {img_2_stat
| <h3>Etnia</h3>  |  <h4>{img_1_stats['dominant_race']} </h4>       |       <
| <h3>Emoção</h3> | <h4>  {img_1_stats['dominant_emotion']}  </h4>     |  <h4

""")
    else:
        return md(f"""
### Análise da face após o swap utilizando DeepFace 🔎

<hr>

| <h2>Característica 🔵</h2> |  <h2>[ Imagem após swap 📸]</h2> |
|--------------------|----------|
| <h3>Gênero</h3> | <h3>{emojify_gender(img_1_stats['gender'])}</h3> |
| <h3>Idade</h3> |   <h4>{img_1_stats['age']}  </h4>    |
| <h3>Etnia</h3>  |  <h4>{img_1_stats['dominant_race']} </h4>       |
| <h3>Emoção</h3> | <h4>  {img_1_stats['dominant_emotion']}  </h4>     |

""")
```

In [11]:
```
show_multiple_images([img_1_original, img_2_original], 1, 2, 'Imagens origina
generate_statistics_markdown(img_1_stats, img_2_stats)
```



Out[11]:

Análise das faces utilizando DeepFace ⚡

Característica 🔵	[Imagem 1 📸]	[Imagem 2 📷]
Gênero	♂	♂
Idade	30	39
Etnia	latino hispanic	white
Emoção	neutral	neutral

Criando uma matriz do tamanho da primeira imagem em preto-e-branco

Pegar o tamanho e canais da segunda imagem

Criar uma matriz com as dimensões e canais da segunda imagem

In [12]:
```
img_1_canvas = np.zeros_like(img_1_grayscale)
height, width, number_of_channels = img_2.shape
img_2_canvas = np.zeros((height, width, number_of_channels), np.uint8)
```

Achar as faces nas imagens, o que que retornará um array contendo as diagonais da face (canto superior esquerdo e inferior direito)

In [13]:
```
faces_1, faces_2 = frontal_face_detector(img_1_grayscale), frontal_face_detec
```

Atravessar a face da primeira e segunda imagem e detectar os pontos principais delas

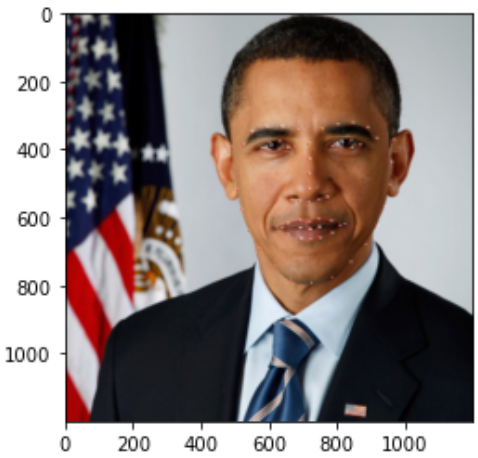
Pega os 68 pontos principais das faces e adiciona-os a lista de pontos faciais

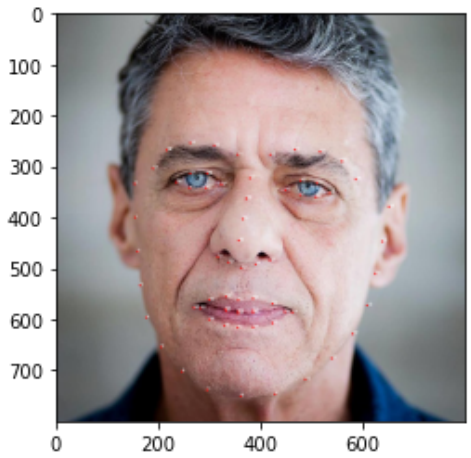
Destaca os pontos faciais principais e apresenta as imagens

In [14]:
```
def set_landmarks(face_landmarks, face_landmark_points, img):
    for landmark in range(68):
        x = face_landmarks.part(landmark).x
        y = face_landmarks.part(landmark).y

        face_landmark_points.append((x, y))
        cv2.putText(img, str(landmark), (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.2
        cv2.circle(img, (x, y), 2, (0, 0, 255), -1)
```

In [15]:
```
face_1_landmarks, face_2_landmarks = frontal_face_predictor(img_1_grayscale,
                                                            frontal_face_predictor(img_2_grayscal
face_1_landmark_points, face_2_landmark_points = [], []
img_1_with_landmarks_and_numbers, img_2_with_landmarks_and_numbers = img_1_or

set_landmarks(face_1_landmarks, face_1_landmark_points, img_1_with_landmarks_
set_landmarks(face_2_landmarks, face_2_landmark_points, img_2_with_landmarks_

show_multiple_images([img_1_with_landmarks_and_numbers, img_2_with_landmarks_
```

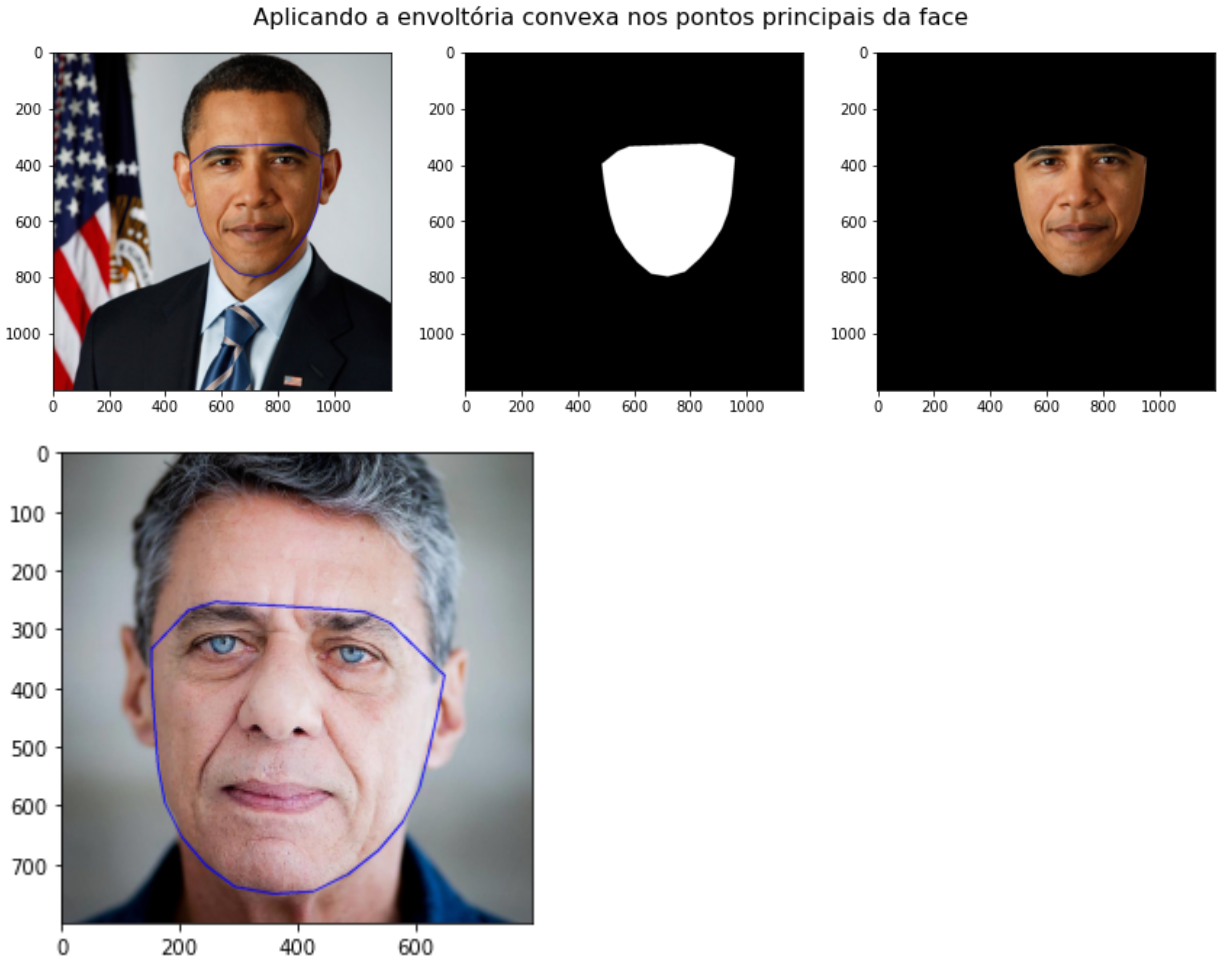
Faces com seus principais pontos destacados





Achar a envoltória convexa dos pontos principais da imagem, ou seja, o menor conjunto convexo que contém os pontos destacados anteriormente. Aplicando a face na máscara posteriormente

In [16]:
```
face_1_landmark_points_array, face_2_landmark_points_array = np.array(face_1_
                                                             np.array(face_
face_1_convex_hull, face_2_convex_hull = cv2.convexHull(face_1_landmark_point
                                         cv2.convexHull(face_2_landmark_po

cv2.polylines(img_1, [face_1_convex_hull], True, (255, 0, 0), 2)
cv2.polylines(img_2, [face_2_convex_hull], True, (255, 0, 0), 2)

cv2.fillConvexPoly(img_1_canvas, face_1_convex_hull, 255), cv2.fillConvexPoly
face_1_trimmed = cv2.bitwise_and(img_1_original, img_1_original, mask=img_1_c
show_multiple_images([img_1, img_1_canvas, face_1_trimmed], 1, 3, 'Aplicando
show_image(img_2)
```



Utilizar a Triangulação de Delaunay para a criação de uma malha contígua de pontos triangulares que não se entrelaçam

Um retângulo será encontrado ao redor dos pontos da envoltória convexa da imagem, criação de uma subdivisão de Delaunay, incluindo todos os pontos do retângulo e retornar a lista de triangulos posteriormente, com 6 itens por vetor de triangulo (x, y) dos 3 pontos

Ao invés de trabalhar com os triângulos, pegaremos os índices do principais pontos faciais do indivíduo, sendo assim, agora teremos o índice dos pontos principais armazenados facilitando o processo de swap com a outra imagem. Basicamente transformando os triângulos de um sistema de coordenadas (x, y) para o formato de pontos faciais principais

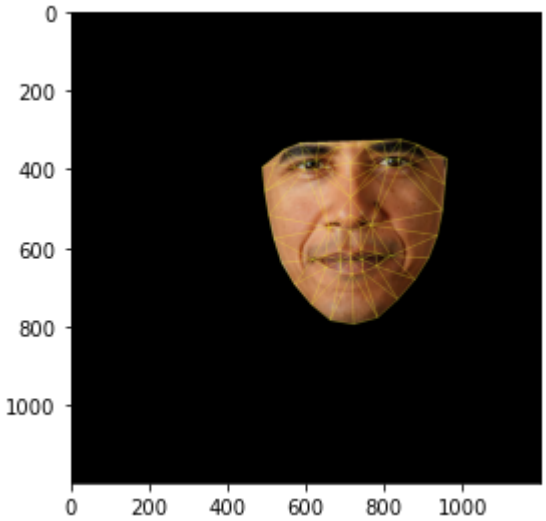
In [17]:
```
bounding_rectangle = cv2.boundingRect(face_1_convex_hull)
subdivision = cv2.Subdiv2D(bounding_rectangle)
subdivision.insert(face_1_landmark_points)
triangles_vector = subdivision.getTriangleList()
triangles_array = np.array(triangles_vector, dtype=np.int32)
triangles_vertices = []
```

In [31]:
```
def fetch_index_from_numpy_array(np_array):
    return np_array[0][0]

def get_vertex_as_landmark_point(landmark_points_array, vertex):
    vertex = np.where((face_1_landmark_points_array == vertex).all(axis=1))
```

```
    vertex = fetch_index_from_numpy_array(vertex)
    return vertex
```

In [32]:

```
for triangle in triangles_array:
    vertex_1 = (triangle[0], triangle[1])
    vertex_2 = (triangle[2], triangle[3])
    vertex_3 = (triangle[4], triangle[5])
    line_color = (51,255,255)
    cv2.line(face_1_trimmed, vertex_1, vertex_2, line_color, 1)
    cv2.line(face_1_trimmed, vertex_2, vertex_3, line_color, 1)
    cv2.line(face_1_trimmed, vertex_3, vertex_1, line_color,1)

    vertex_1 = get_vertex_as_landmark_point(face_1_landmark_points_array, ver
    vertex_2 = get_vertex_as_landmark_point(face_1_landmark_points_array, ver
    vertex_3 = get_vertex_as_landmark_point(face_1_landmark_points_array, ver

    triangle = [vertex_1, vertex_2, vertex_3]
    triangles_vertices.append(triangle)

show_image(face_1_trimmed)
```



Cortar os triangulos demarcados de ambas as fotos

- Capturar as coordenadas dos triângulos
- Destacar um retângulo ao redor desse triângulo
- Cortar o retângulo para depois colá-lo na segunda imagem
- Remover os pontos do retângulo
- Para determinar os triângulos para a segunda foto os mesmos índices dos pontos faciais principais são utilizados (da lista de triâgulos da primeira foto, contendo os índices dos pontos faciais princiais), e também haverá um retângulo ao redor do triângulo da segunda imagem. Para cada triângulo demarcado na segunda imagem uma nova máscara é criada com os pontos do triangulo relativos a segunda imagem
- Ajustar a máscara da imagem 1 para caber na máscara recortada da imagem 2 utilizando transformação afim
- Reconstrução da segunda imagem utilizando os triângulos após a transformação a fim de manter as proporções da face original
- Por fim, recorta-se um setor da nova imagem 2 (será utilizado um canvas preto a ser preenchido), transforma-o para preto-e-branco e retira-se as secções brancas do triângulo para melhorar na interpolação destes. Utilizando esta máscara que filtra os

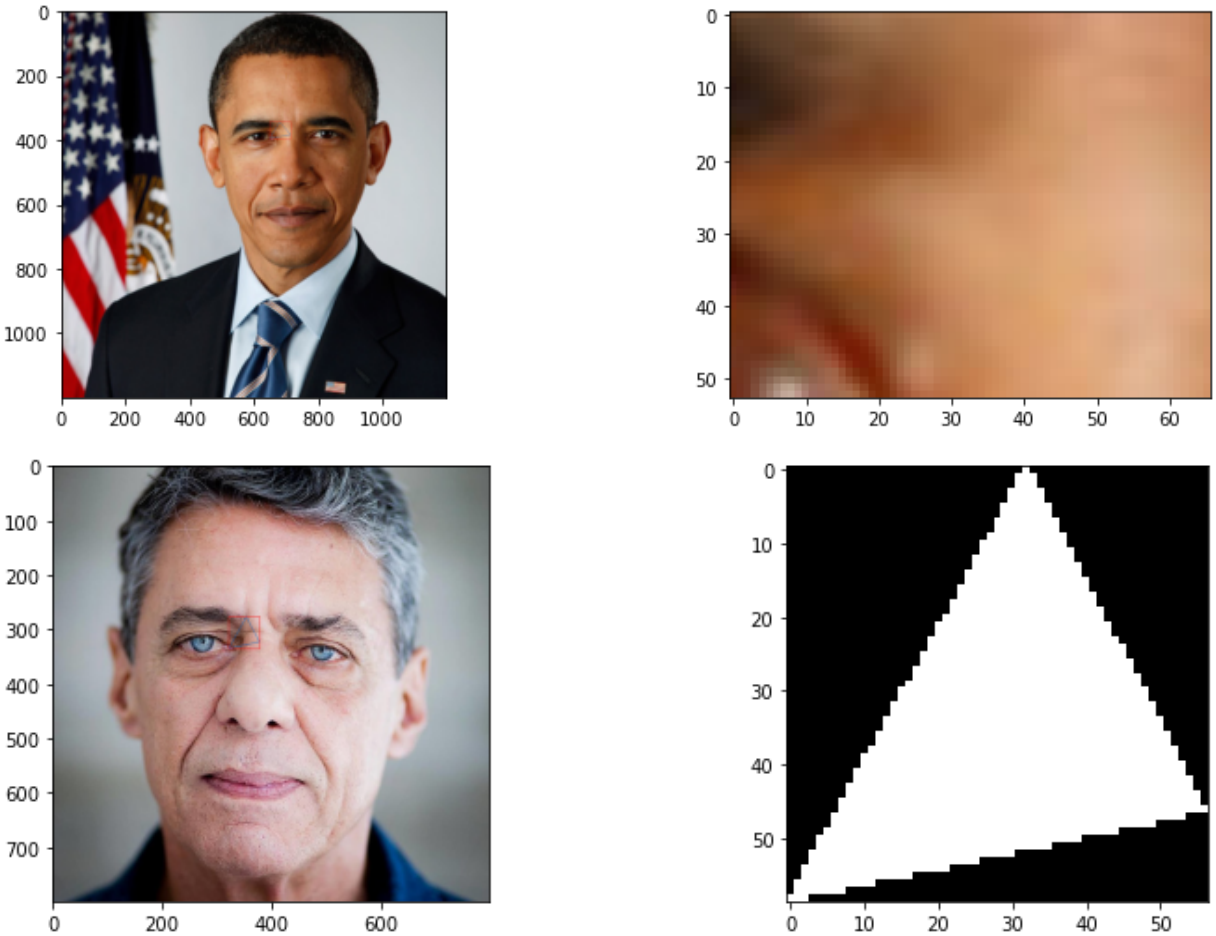
pontos brancos, o triângulo recortado relativo a segunda imagem é adicionado ao canvas. Para cada triângulo extraído da imagem 1

In [20]:
```
def extract_triangle(face_landmark_points, triangle_index_points):
    point_1 = face_landmark_points[triangle_index_points[0]]
    point_2 = face_landmark_points[triangle_index_points[1]]
    point_3 = face_landmark_points[triangle_index_points[2]]
    triangle = np.array([point_1, point_2, point_3], np.int32)

    return point_1, point_2, point_3, triangle

def get_rectangle(triangle):
    rectangle = cv2.boundingRect(triangle)
    (x, y, w, h) = rectangle
    return rectangle, (x, y, w, h)

def draw_rectangle(point_1, point_2, point_3, rectangle_visualization, rectan
    (x, y, w, h) = rectangle
    cv2.line(rectangle_visualization, point_1, point_2, (200, 100, 0), 1)
    cv2.line(rectangle_visualization, point_2, point_3, (200, 100, 0), 1)
    cv2.line(rectangle_visualization, point_3, point_1, (200, 100, 0), 1)
    cv2.rectangle(rectangle_visualization, (x, y), (x + w, y + h), (0, 0, 255
    show_multiple_images([rectangle_visualization, crop], 1, 2, txt)

def extract_triangle_points(point_1, point_2, point_3, x, y):
    return np.array([[point_1[0] - x, point_1[1] - y],
                     [point_2[0] - x, point_2[1] - y],
                     [point_3[0] - x, point_3[1] - y]], np.int32)
```

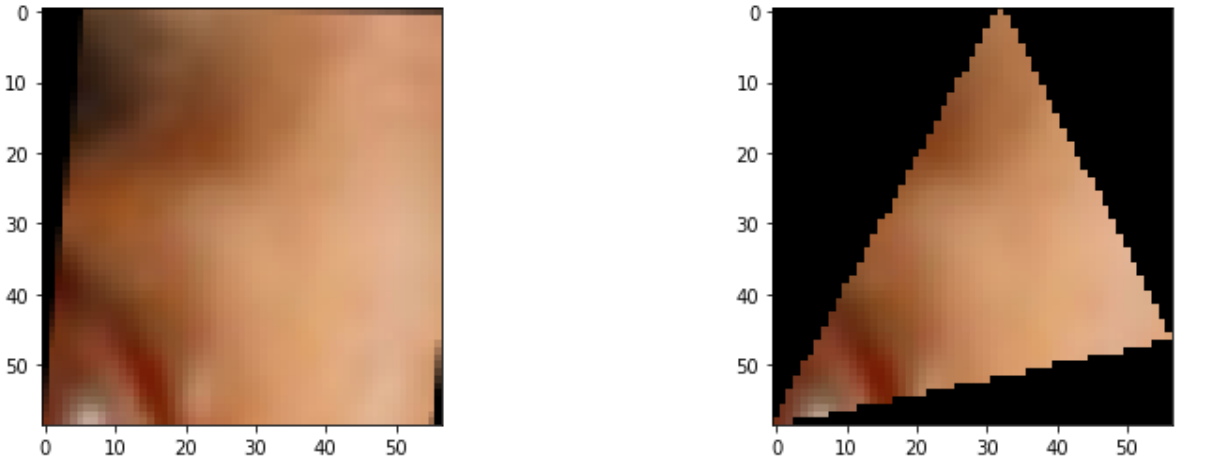
In [21]:
```
rectangle_visualization_1, rectangle_visualization_2  = img_1_original.copy()
img_2_canvas = np.zeros((height, width, number_of_channels), np.uint8)
for i, triangle_index_points in enumerate(triangles_vertices):
    triangle_1_point_1, triangle_1_point_2, triangle_1_point_3, triangle_1 =
    rectangle_1, (x, y, w, h) = get_rectangle(triangle_1)

    cropped_rectangle_1 = img_1_original[y: y + h, x: x + w]

    triangle_1_points = extract_triangle_points(triangle_1_point_1, triangle_

    triangle_2_point_1, triangle_2_point_2, triangle_2_point_3, triangle_2 =
    rectangle_2, (x, y, w, h) = get_rectangle(triangle_2)

    cropped_rectangle_2_mask = np.zeros((h, w), np.uint8)

    triangle_2_points = extract_triangle_points(triangle_2_point_1, triangle_
    cv2.fillConvexPoly(cropped_rectangle_2_mask, triangle_2_points, 255)

    triangle_1_points, triangle_2_points = np.float32(triangle_1_points), np.
    transformation_matrix = cv2.getAffineTransform(triangle_1_points, triangl
    warped_rectangle = cv2.warpAffine(cropped_rectangle_1, transformation_mat
    warped_triangle = cv2.bitwise_and(warped_rectangle, warped_rectangle, mas

    face_2_canvas_area = img_2_canvas[y: y + h, x: x + w]
    face_2_canvas_area_grayscale = cv2.cvtColor(face_2_canvas_area, cv2.COLOR_
    _, mask_created_triangle = cv2.threshold(face_2_canvas_area_grayscale, 1,

    wraped_triangle = cv2.bitwise_and(warped_triangle, warped_triangle, mask=m

    face_2_canvas_area = cv2.add(face_2_canvas_area, wraped_triangle)
    img_2_canvas[y: y + h, x: x + w] = face_2_canvas_area
    if i == 23:
        draw_rectangle(triangle_1_point_1, triangle_1_point_2, triangle_1_poi
                       rectangle_visualization_1, rectangle_1, cropped_recta
        draw_rectangle(triangle_2_point_1, triangle_2_point_2, triangle_2_poi
```

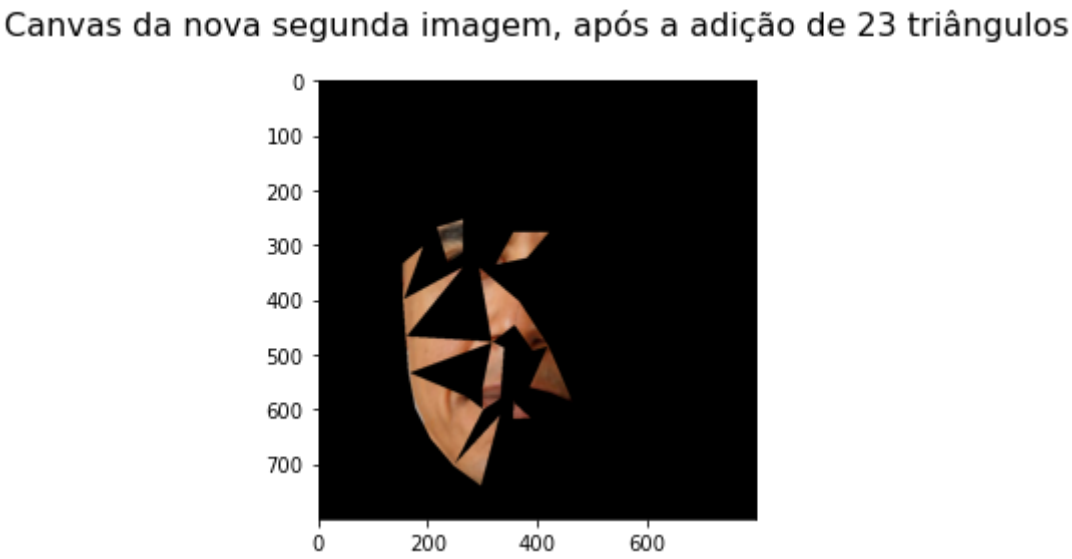
```
                    rectangle_visualization_2, rectangle_2, cropped_recta
show_multiple_images([warped_rectangle, warped_triangle], 1, 2, 'Tran
show_image(img_2_canvas, 'Canvas da nova segunda imagem, após a adiçã
```

Retângulo extraído do triângulo 23 da imagem



Transformação afim da primeira máscara para essa região da segunda imagem

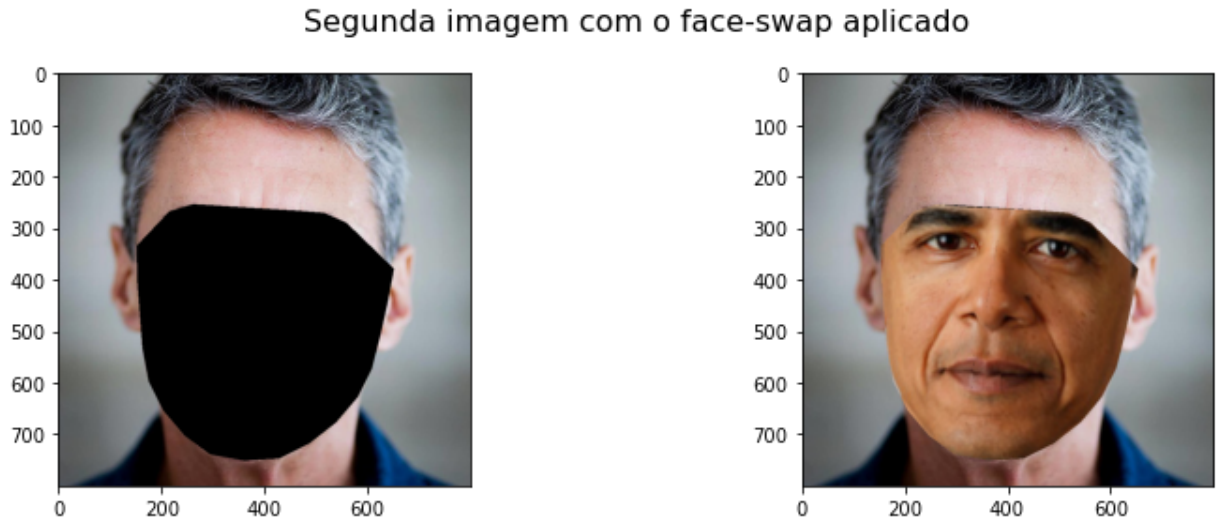




In [22]:
```
show_image(img_2_canvas, 'Canvas após a adição de todos os triângulos')
```

Criação de um novo canvas para guardar a face após o swap, esse canvas terá o tamanho da segunda imagem. Depois há o preenchimento da envoltória convexa com esses pontos (envolvendo a face da segunda imagem), após isso a máscara é adicionada a imagem original e por fim, ela é preenchida pela face pós-swap

In [23]:
```
img_2_face_mask_canvas = np.zeros_like(img_2_grayscale)
img_2_face_mask = cv2.fillConvexPoly(img_2_face_mask_canvas, face_2_convex_hu
img_2_face_mask_canvas = cv2.bitwise_not(img_2_face_mask)
img_2_without_face = cv2.bitwise_and(img_2_original, img_2_original, mask=img_
img_2_with_face = cv2.add(img_2_without_face, img_2_canvas)
show_multiple_images([img_2_without_face, img_2_with_face], 1, 2, 'Segunda im
```

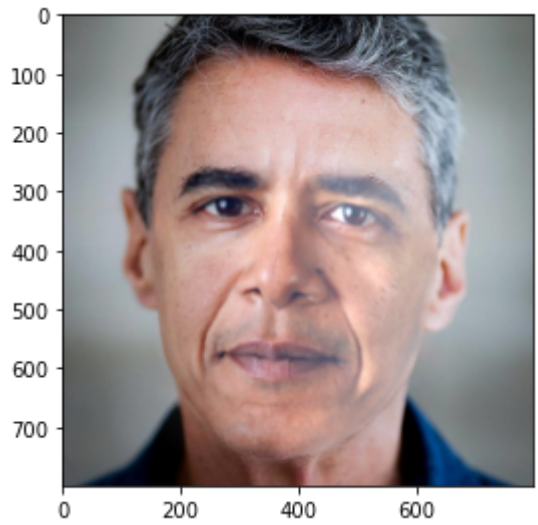


Utilizar o método Seamless Clone deixando a nova face mais natural e ajustada a imagem original, para tanto, faz-se necessário encontrar o ponto central da envoltória convexa da imagem, tornando-o mais natural a partir deste ponto, outrossim, utilizará a imagem original, visando capturar a iluminação e cores dela

In [24]:
```
(x, y, w, h) = cv2.boundingRect(face_2_convex_hull)
face_2_center_point = (int((x+x+w)/2), int((y+y+h)/2))
img_seamless_cloned = cv2.seamlessClone(img_2_with_face, img_2_original, img_
img_swapped_stats = DeepFace.analyze(img_seamless_cloned, actions = ['age', '
```

```
Action: emotion: 100%|██████████| 4/4 [00:19<00:00,  4.75s/it]
```

In [25]:
```
show_image(img_seamless_cloned)
generate_statistics_markdown(img_swapped_stats)
```



Out[25]:

Análise da face após o swap utilizando DeepFace 🔎

Característica 🔵	[Imagem após swap 📸]
Gênero	♂
Idade	33

Característica 🔵	[Imagem após swap 📸]
Etnia	latino hispanic
Emoção	neutral

Conclusão e Discussão 📈

O projeto com o tema Face Swap possui diversas qualidades que podemos destacar. Além de possuir um conteúdo atual e em crescente alta devido a utilização de filtros e manipulação de imagens com o advento das redes sociais e a ampla utilização de imagens em suas redes; o tema possui uma forte base matemática durante todo seu desenvolvimento, que age em conjunto com técnicas de programação para auxiliar o desenvolvedor em um sistema alto nível

Para o desenvolvimento desse projeto, optamos pela linguagem de programação Python devido a quantidade de informações na rede e a numerosa comunidade que auxilia e incentiva a elaboração de projetos e pesquisas, bem como pela afinidade do grupo com a linguagem de experiências anteriores

Fez-se uso de algumas bibliotecas que devem ser citadas pela sua contribuição com o projeto: Deepface, Numpy, Pandas, Matplotlib, OpenCV e dlib

Dentre as bibliotecas citadas anteriormente, podemos destacar a utilização da biblioteca Deepface para detecção de gêneros, emoções, etnia e idades médias. Essa biblioteca é uma evolução muito grande na área de IA visto que com poucas linhas pode-se obter muitas informações com resultados satisfatórios

Ao todo, o tratamento das imagens possuem particularidades. Imagens que possuem coloração mais clara e pigmentações distintas, como também qualidade de imagens diferentes são alguns dos fatores que podem agravar os resultados inusitados gerados pelo Face Swap

Logo, podemos concluir que toda a pesquisa está fortemente adotada em vários setores, como o de filmes ou séries ("Velozes e Furiosos 7", "Homem de Ferro 3"), filtros utilizados em redes sociais (Instagram, Facebook, Snapchat, Tik Tok) entre outros mercados