python-image_aliasing_new

September 5, 2024

0.1 image_aliasing_new.m

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import scipy.fft as fft
```

[ ]:
```
import skvideo
skvideo.setFFmpegPath("C:/Users/74005/Downloads/
 ↪ffmpeg-20210728-0068b3d0f0-win64-shared/bin")
```

[ ]:
```
import skvideo.io
```

Definimos `write_video` para escribir una secuencia de frames en un archivo de video. Esta función ajusta el tamaño de cada frame para que todos tengan el mismo tamaño, rellenando con ceros si es necesario, y luego los escribe en un archivo de video utilizando la biblioteca skvideo.

[ ]:
```
def write_video(file_path, frames):
    # Calculate the maximum height and width of all frames
    max_height = max(frame.shape[0] for frame in frames)
    max_width = max(frame.shape[1] for frame in frames)

    # Pad each frame to the maximum height and width
    padded_frames = [np.pad(frame, ((0, max_height - frame.shape[0]), (0,␣
 ↪max_width - frame.shape[1]))) for frame in frames]

    skvideo.io.vwrite(file_path, np.array(padded_frames, dtype=np.uint8))
```

Con `centered_affine_transform` se aplica una transformación afín a una imagen, asegurando que la transformación esté centrada. Esta función calcula la nueva posición de la imagen transformada y aplica la transformación para obtener la imagen resultante.

[ ]:
```
def centered_affine_transform(t, src_im):

    h, w = src_im.shape[:2]

    # Extract translation components from the transformation matrix
    trans = t[:-1]
```

```
    inv_t = np.linalg.inv(t)
    inv_trans = inv_t[:-1]

    # Define source points for the transformation
    src_pts = np.float32([[0, 0], [w-1, 0], [0, h-1], [w-1, h-1]])

    # Apply the transformation to the source points
    dst_pts = cv2.transform(np.array([src_pts]), trans)[0]

    # Calculate the bounds of the transformed image
    min_x, max_x = np.min(dst_pts[:, 0]), np.max(dst_pts[:, 0])
    min_y, max_y = np.min(dst_pts[:, 1]), np.max(dst_pts[:, 1])

    # Calculate the size of the transformed image
    dst_w, dst_h = int(max_x - min_x + 1), int(max_y - min_y + 1)

    # Calculate the center of the transformed image
    dst_center = np.float32([[(dst_w-1.0)/2, (dst_h-1.0)/2]])

    # Project the center of the transformed image back onto the source image
    src_projected_center = cv2.transform(np.array([dst_center]), inv_trans)[0]

    # Calculate the translation needed to center the transformation
    translation = src_projected_center - np.float32([[(w-1.0)/2, (h-1.0)/2]])

    # Update the translation components of the transformation matrix
    trans[:, 2] = translation

    # Apply the centered affine transformation to the image
    return cv2.warpAffine(src_im, trans, (dst_w, dst_h))
```

Con `scale_shrink` definimos las matrices de la transformación:

[ ]:
```
def scale_shrink(xshrink, xsize):
    scale_shrink = (xsize - xshrink) / xsize
    return np.array([
        [scale_shrink, 0, 0],
        [0, scale_shrink, 0],
        [0, 0, 1],
    ])
```

Con `scale_boost` creamos una matriz de transformación que vuelve a escalar la imagen a su tamaño original después de haber sido reducida por la función `scale_shrink`:

[ ]:
```
def scale_boost(xsize, f2):
    scale_boost = xsize / f2.shape[1]
    return np.array([
        [scale_boost, 0, 0],
```

```
        [0, scale_boost, 0],
        [0, 0, 1],
    ])
```

Cargamos el gif e iteramos por los valores de xshrink:

[ ]:
```
frames = []
specs = []

f = plt.imread('../assets/barbara.gif')

for xshrink in range(0, 600, 5):
    xsize = f.shape[1]
    t1 = scale_shrink(xshrink, xsize)
    f2 = centered_affine_transform(t1, f)
    t2 = scale_boost(xsize, f2)
    f3 = centered_affine_transform(t2, f2)
    frames.append(f3[:, :, 0])
    Fd = np.log(1 + np.abs(np.fft.fftshift(np.fft.fft2(frames[-1]))))
    specs.append(np.uint8(256 * Fd / Fd.max()))
```

Creamos un video con los frames:

[ ]:
```
write_video('aliasing_pics.avi', frames)
```

Creamos un video con los espectros de frecuencia de las imágenes de cada frame:

[ ]:
```
write_video('aliasing_specs.avi', specs)
```