seam

June 29, 2021

[ ]:
```
import cv2
import numpy as np

from scipy.ndimage.filters import convolve

from tqdm import trange
```

0.1 Computing Energy

The energy is computed by $e_i(I) = \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right|$.

Using Sobel Filters, $p'_u = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \circledast I$ and $p'_v = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \circledast I$, we compute the derivative as $e_i(I) = |p'_u| + |p'_v|$.

Source

0.2 Fnding the Seam with least Energy

To compute the seam with the least energy, we dynamically compute $M(i,j) = e(i,j) + \min\left(M(i-1,j-1), M(i-1,j), M(i-1,j+1)\right)$ where $M(i,j)$ stores the minimum energy value stored upto the pixel (i,j).

The minimum energy thus will be stored in the last row of **M** and **backtrack** stores the list of pixels present in this s

Inspired from

0.3 Deleting the Minimum Seam

We recursively remove the seam with minimum energy till we reach the image of required size.

[ ]:
```
class Seam():
    def __init__(self, path):
        self.image = cv2.imread(path)
        self.name = path.split('.')[0]

    def findEnergy(self, image):
```

```
        filter_du = np.array([ [1.0, 2.0, 1.0], [0.0, 0.0, 0.0], [-1.0, -2.0,␣
↪-1.0],])
        filter_dv = np.array([ [1.0, 0.0, -1.0], [2.0, 0.0, -2.0], [1.0, 0.0,␣
↪-1.0],])

        filter_du = np.stack([filter_du] * 3, axis=2)
        filter_dv = np.stack([filter_dv] * 3, axis=2)

        convolved = np.absolute(convolve(image, filter_du)) + np.
↪absolute(convolve(image, filter_dv))

        energy = convolved.sum(axis=2)

        return energy

    def minimum_seam(self, image):
        row, column, _ = image.shape
        energy = self.findEnergy(image)

        M = energy.copy()
        backtrack = np.zeros_like(M, dtype=np.int)

        for i in range(1, row):
            for j in range(0, column):

                if j == 0:
                    idx = np.argmin( M[i-1, j:j+2] )
                    backtrack[i, j] = idx + j
                    min_energy = M[i-1, idx+j]
                else:
                    idx = np.argmin(M[i-1, j-1:j+2])
                    backtrack[i, j] = idx +j-1
                    min_energy = M[i-1, idx+j-1]

                M[i, j] += min_energy

        return M, backtrack

    def carve(self, image):
        row, column, _ = image.shape

        M, backtrack = self.minimum_seam(image)

        mask = np.ones((row, column), dtype=np.bool)

        j = np.argmin(M[-1])
```

```
        for i in reversed(range(row)):

            mask[i, j] = False
            j = backtrack[i, j]

        mask = np.stack([mask] * 3, axis=2)

        image = image[mask].reshape((row, column-1, 3))

        return image

    def crop(self, image, scale):
        row, column, _ = image.shape
        new = int(scale * column)

        for i in trange(column - new):
            image = self.carve(image)

        return image

    def crop_column(self, scale_column):
        image = self.image
        image = self.crop(image, scale_column)

        fname = self.name + "_scaled_column_by_" + str(scale_column) + ".png"
        cv2.imwrite(fname, image)
        return image

    def crop_row(self, scale_row):
        image = self.image
        image = np.rot90(image, 1, (0, 1))
        image = self.crop(image, scale_row)
        image = np.rot90(image, 3, (0, 1))

        fname = self.name + "_scaled_row_by_" + str(scale_row) + ".png"
        cv2.imwrite(fname, image)

        return image
```

[ ]:
```
img1 = Seam("Image-1.jpg")
out1 = img1.crop_column(0.5)
```

[ ]:
```
img2 = Seam("Image-2.jpg")
out2 = img2.crop_column(0.5)
```

[ ]:
```

```