به نام خدا

گزارش پروژه پایانی درس طراحی الگوریتم

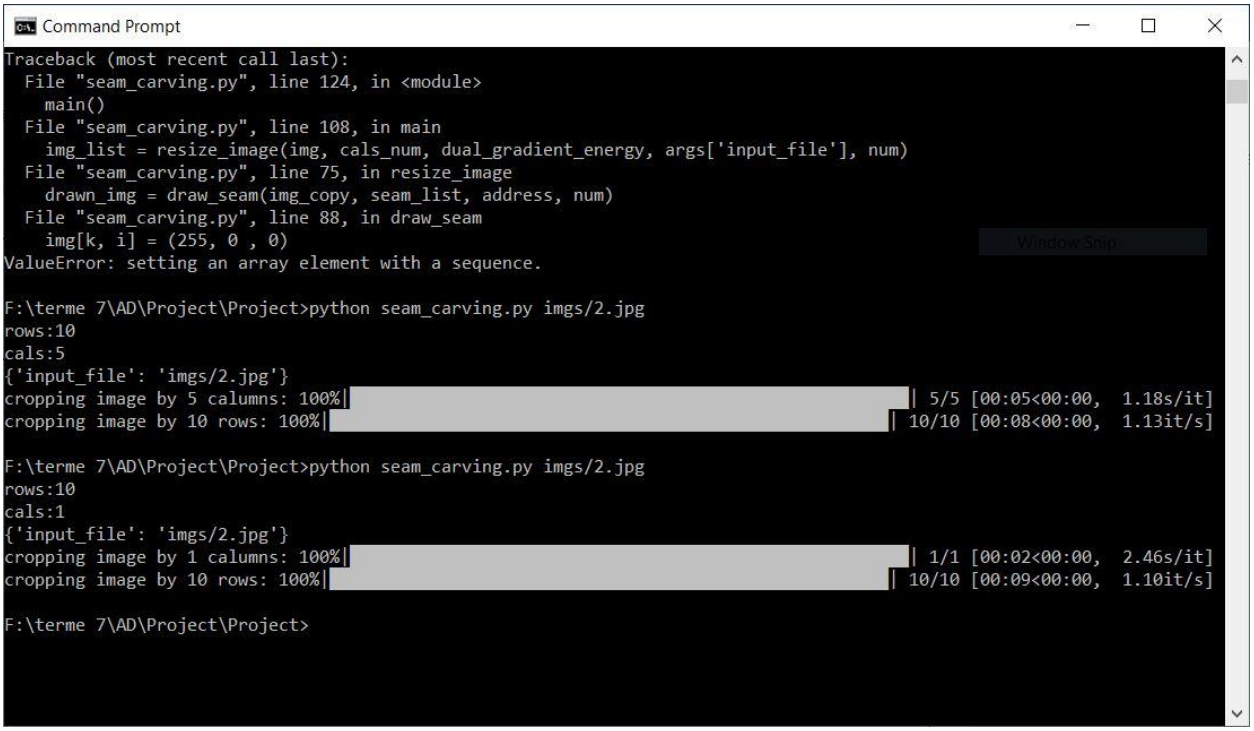
محمد یارمقدم

96462104

موضوع : Seam carving

استاد انتظاری ملکی

در ابتدا به توضیح نحوه اجرای کد می پردازم و سپس چند نمونه ورودی و خروجی را می بینیم و در آخر به توضیح جز به جز کد می پردازم.



در این نحوه اجرا ، عکسی مه قصد انجام الگوریتم بر روی آنرا داریم باید در پوشه Imgs **در فایل پروژه قرار داشته باشد.**

در آرگومان داده شده به فایل نام عکس داده میشود.

سپس در دو خط بعدی به ترتیب تعداد سطر و ستون مورد نظر برای حذف وارد میشود.

سپس محاسبات با یک عملیات load **به کاربر نمایش داده میشود و در انتها** 3 **عکس در فولدر** Imgs **چاپ میشود** : (**فرض میکنیم نام عکس ورودی** 2 **است)**

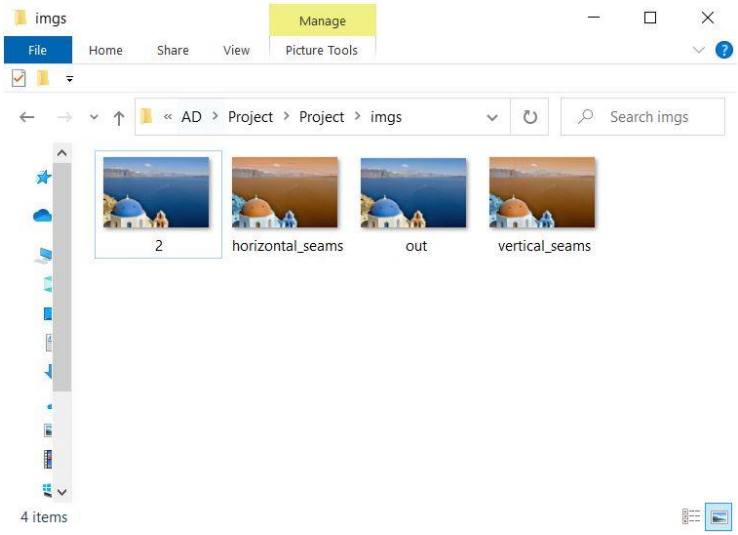
1– **عکس** out **که عکس تغییر داده شده است.**

2– **عکس** vertical_seams **که شامل نقش درز های عمودی بر روی عکس اصلی است.**

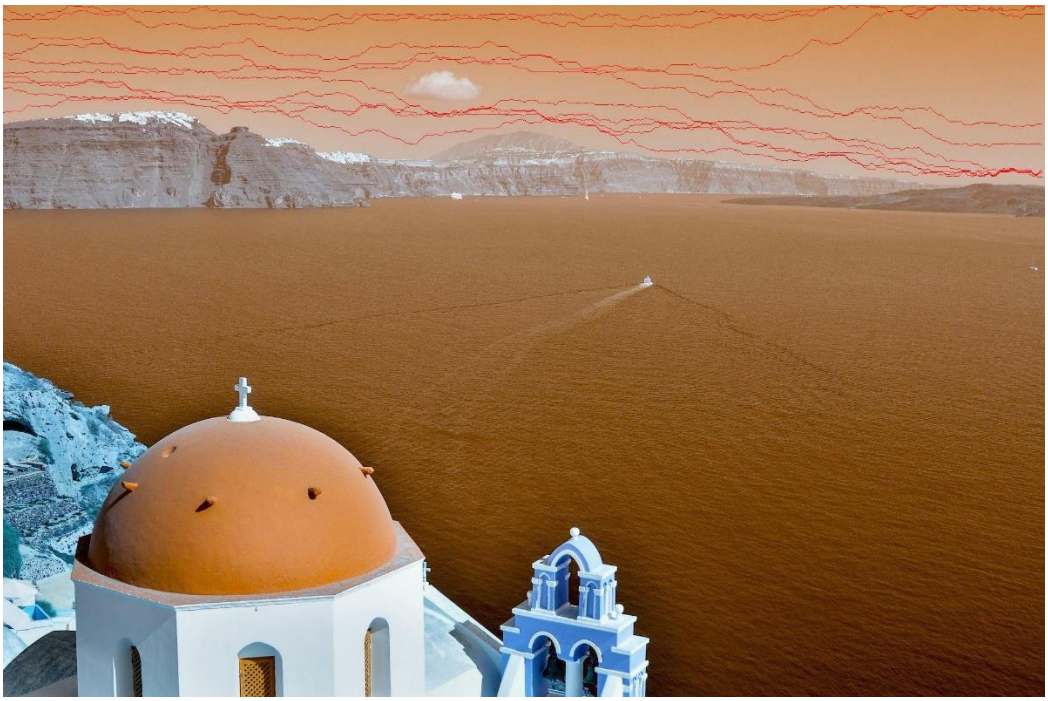
3– **عکس** horizontal_seams **که شامل نقش درز های افقی بر روی عکس اصلی است.**

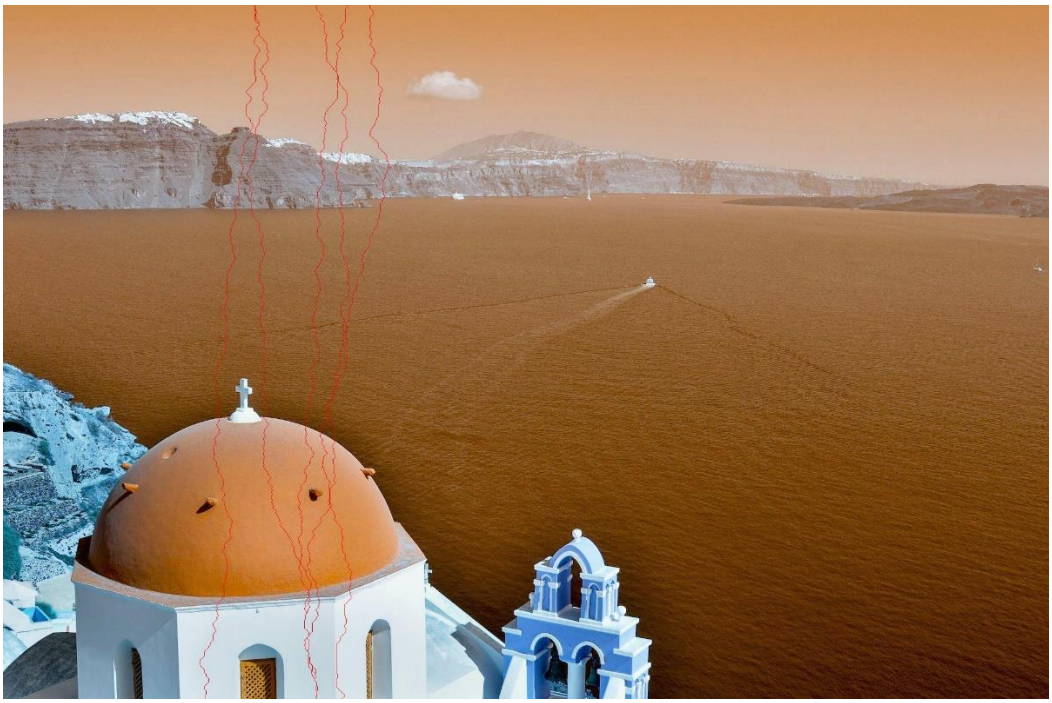
حال چند نمونه ورودی و خروجی را می بینیم :

(فرض میکنیم در تمامی حالات 10 **سطر و** 5 **ستون را میخواهیم حذف کنیم)**



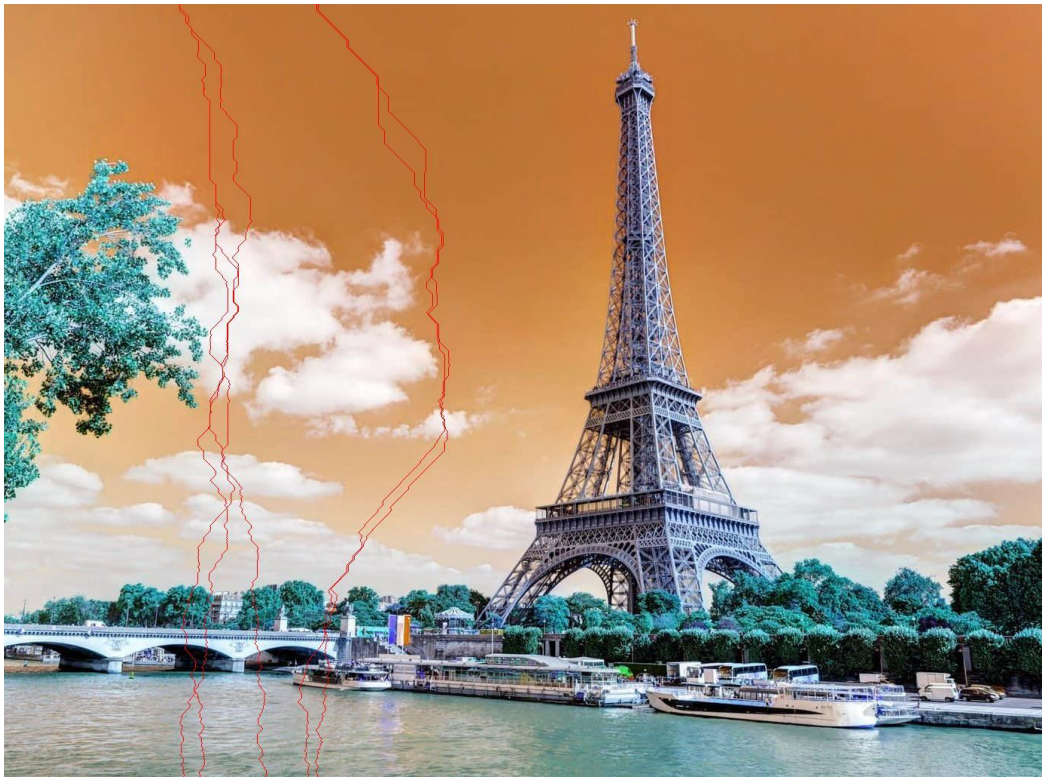
خروجی ها :





ورودی 2 :

خروجی ها :



ورودی 3 **:**

خروجی ها :

حال به توضیح کد و الگوریتم می پردازیم و جز به جز هر قسمت را شفاف سازی می کنم.

در قسمت اول برنامه از main **شروع به کار می کند.در این قسمت ابتدا آدرس فایل ورودی و تعداد سطر و ستون مورد نظر برای حذف از عکس داده می شود.**

سپس با استفاده از argumentparser **آرگومان ورودی را** parse **می کنم تا آدرس فایل ورودی به دست بیاید.**

در این مرحله ابتدا توسط کتابخانه Image **از کلاس** PIL **عکس را از آدرس داده شده باز میکنیم.**

سپس توسط تابع array **از کلاس** np **آرایه متناظر پیکسل های عکس داده شده را بدست می آوریم.**

حال تابع resize_image **را فراخوانی میکنیم. این تابع را ابتدا برای حذف ستون ها روی عکس اصلی صدا میزینم و به آن عکس و تعداد سطر موردنظر برای حذف و تابع انرژی مورد نظر و یک عدد پاس میدهیم.**

عدد آخر برای تابع مشخص می کند که در حال حذف سطر یا ستون هستیم. اگر 1 **باشد سطر و اگر** 0 **باشد ستون.**

سپس برای حذف سطر عکس خروجی از تابع را توسط تابع transpose **از کلاس** np **دوران میدهیم و دوباره تابع را فراخوانی میکنیم.**

در نهایت عکس خروجی از فراخوانی دوم تابع را دوبار دوران میدهیم تا به حالت اولیه باز گردد و سپس با استفاده از تابع formarray **از کلاس** np **عکس را سیو میکنیم.**

```
def main():
    rows_num = int(input("rows:"))
    cals_num = int(input("cals:"))
    parser = argparse.ArgumentParser(description="Intelligently crop an image along one axis")
    parser.add_argument('input_file')
    args = vars(parser.parse_args())
    print(args)
    num = 0
    img = np.array(Image.open(args['input_file']))
    img_list = resize_image(img, cals_num, dual_gradient_energy, args['input_file'], num)
    cropped_img = img_list[1]
    img = np.transpose(cropped_img, axes=(1, 0, 2))
    num = 1
    img_lists = resize_image(img, rows_num, dual_gradient_energy, args['input_file'], num)
    cropped_img = img_lists[1]
    seam_img_1 = img_list[0]
    seam_img_3 = img_lists[0]
    final_img = np.transpose(cropped_img, axes=(1, 0, 2))
    seam_img_4 = np.transpose(seam_img_3, axes=(1, 0, 2))
    seam_img_5 = np.transpose(seam_img_4, axes=(1, 0, 2))
    Image.fromarray(final_img).save("imgs/out.jpg")
    Image.fromarray(seam_img_1).save("imgs/vertical_seams.jpg")
    Image.fromarray(seam_img_5).save("imgs/horizontal_seams.jpg")
```

حال به بررسی توابع و الگوریتم پیاده شده می رسیم :

در این تابع ابتدا دو کپی از آرایه عکس گرفته میشود.

سپس در یک حلقه به تعداد سطر یا ستون مورد نظر (که این از روی پارامتر num **مشخص میشود) ابتدا تابع** energy_map **فراخوانی میشود. در این تابع پیکسل راست و چپ پیکسل مورد نظر به تابع انرژی داده میشود تا انرژی پیکسل محاسبه شود.**

سپس خروجی تابع energy_map **به تابع** cumulative_energy **داده میشود.**

در تابع cumulative_energy **در سطرهای** 1 **تا انتهای عکس به ازای تمامی ستون های عکس حلقه زده میشود و در آن به ازای هر پیکسل ، مقایسه بین مقادیر بالا چپ و بالا و بالا راست (در صورت وجود) انجام میشود و مینیمم این مقادیر با انرژی پیکسل جمع زده میشود و ذخیره میشود.**

هم چنین مسیر رسیدن به این پیکسل نیز ذخیره میشود.

در مرحله بعد توسط تابع find_seam **از پایین ترین سطر و کمترین مقدار انرژی کار آغاز و مسیر رسیدن به آن دنبال میشود تا** seam **شناسایی شود . سپس در یک لیست ، درز شناسایی شده ذخیره و در نهایت پس از یافتن و اضافه شدن تمامی آنها به سراغ رنگ کردن پیکسل های درز میرویم.**

```
def resize_image(full_img, cropped_pixels, energy_fn, address, num):
    img = full_img.copy()
    img_copy = full_img.copy()
    seam_list = []
    img_list = []
    if num == 0:
        for i in trange(cropped_pixels, desc='cropping image by {0} calumns'.format(cropped_pixels)):
            e_map = energy_map(img, energy_fn)
            e_paths, e_totals = cumulative_energy(e_map)
            seam = find_seam(e_paths, seam_end(e_totals))
            seam_list.append(seam)
            img = remove_seam(img, seam)
    if num == 1:
        for i in trange(cropped_pixels, desc='cropping image by {0} rows'.format(cropped_pixels)):
            e_map = energy_map(img, energy_fn)
            e_paths, e_totals = cumulative_energy(e_map)
            seam = find_seam(e_paths, seam_end(e_totals))
            seam_list.append(seam)
            img = remove_seam(img, seam)
    drawn_img = draw_seam(img_copy, seam_list, address, num)
    img_list.append(drawn_img)
    img_list.append(img)
    return img_list
```

در تابع draw_seam **تک به تک پیکسل های هر سیم رنگ میشود.**

فقط باید از طریق پارامتر num **مشخص شود درز یافت شده مربوط به سطر یا ستون است که متاسب با آن ایندکس مناسب از آرایه پیکسل ها رنگ شود. سپس از روی تغییر** RGB **پیکسل مورد نظر اینکار صورت میگیرد. از آنجا که** RGB **رنگ قرمز** (255, 0 , 0) **است پیکسل ها قرمز میشوند.**

برای رنگ کردن پیکسل های درز های سطر یا ستون از دو عکس مجزا استفاده شد تا جزییات بهتر نمایش داده شود.

سپس عکس اصلاح شده و عکس های درز های سطر و ستون ها در یک لیست اضافه و برگشت داده میشود.

```python
def draw_seam(img, seam_list, address, num):
    img = cv2.imread(address)
    if num ==0:
        for seam in seam_list:
            k = 0
            for i in seam:
                img[k, i] = (255, 0 , 0)
                k += 1
    if num == 1:
        for seam in seam_list:
            k = 0
            for i in seam:
                img[i, k] = (255, 0 , 0)
                k += 1
    return img
```

```python
def seam_end(energy_totals):
    return list(energy_totals[-1]).index(min(energy_totals[-1]))


def find_seam(paths, end_x):
    height, width = paths.shape[:2]
    seam = [end_x]
    for i in range(height-1, 0, -1):
        cur_x = seam[-1]
        offset_of_prev_x = paths[i][cur_x]
        seam.append(cur_x + offset_of_prev_x)
    seam.reverse()
    return seam


def remove_seam(img, seam):
    height, width = img.shape[:2]
    return np.array([np.delete(img[row], seam[row], axis=0) for row in range(height)])
```

تابع انرژی استفاده شده نیز همانند مطلب گفته شده در داک به شکل زیر است:

```
import math
def dual_gradient_energy(x0, x1):
    return sum(pow((x0-x1), 2))
```

محاسبه پیچیدگی زمانی و مکانی :

برای هر پیکسل یک زیر برنامه موجود است. برای هر زیربرنامه نیز حداکثر 3 **رابط وجود دارد. به این ترتیب ما میزان کار ثابتی خواهیم داشت. چون از روش** dp **استفاده کردیم.**

پس اگر عکس ما عرض W **پیکسل و طول** H **پیکسل را داشته باشد ، پیچیدگی زمانی ما** O(W*H+W) **میشود.**

در هر iteration **هم ما دو لیست ذخیره میکنیم که یکی برای سطر قبل و یکی برای سطر حال حاضر هست که در این صورت پیچیدگی مکانی ما** O(2W) **میشود که همان** O(W) **است.**