IMPERIAL COLLEGE LONDON, DEPARTMENT OF MATHEMATICS

Scientific Computing (M3SC)

YOUR NAME
YOUR CID

January 23, 2018

IMAGE BLENDING AND FRACTAL GROWTH- COURSE WORK

In this coursework we will guide you through the development of an image blending algorithm and a fractal growth solver. Your submission must consist of ONE zip folder containing:

- 1 pdf report - using the LaTeX template provided on BB.
- 2 Folders, one for each part (Blending & Fractals)
  - Folder 1: ImBlending.py + 3 given images (source, mask, destination) + the 10 images required in the report.
  - Folder 2: Fractals.py + 5 images required in the report

It is important that you get the results right, but this CW, more than the first one, will penalise inefficient and cumbersome codes. Speed is a priority here.
Do not modify any of the functions that are already provided. If we cannot run your codes from our main script you will lose marks.
You are also advised to use the LaTeX template provided.

1 PART A:IMAGE BLENDING (11 PTS)

This section will guide you through the development of a image blending code. You are asked to apply the algorithm to a given example and to also generate your own image. The algorithm is based on the Poisson equation with Dirichlet boundary conditions applied to the region of the destination image you have to reconstruct with the source image.
The code you are given works on a black and white image. You will modify it later on to handle colour images. You should read or look at this paper: `https://www.cs.virginia.edu/~connelly/class/2014/comp_photo/proj2/poisson.pdf`

$$\begin{aligned} \nabla^2 f &= \nabla \cdot v \qquad \text{on } \Omega, \\ f &= g \qquad \text{on } \partial\Omega \end{aligned} \tag{1.1}$$

1.1 THE NEIGHBOURS [3 PTS]

The first task is to develop the function 'Neighbours' which finds the indices of the masked area and their neighbours defined by the image mask.jpg. The nodes to be found are:

- The nodes of the masked area [Ic] ('Index centre')
- Their northern neighbours [In]
- Their southern neighbours [Is]
- Their eastern neighbours [Ie]
- Their western neighbours [Iw]

You will group the indices in a list called 'Neigh' which is the output of the function.

```
1 def Neighbours(msk):
2 # msk:= binary matrix (only 0 & 1)
3 # Output---
4 # Neigh:=  list containg all the indices
5
6     Neigh=[Ic, In, Is, Iw, Ie]
7     return Neigh
```

Test your function on the example shown in figure 1.1 and check that you obtain the following:

Ic	9, 10, 11, 17, 18, 23, 24, 25, 26, 29, 30, 31, 32, 33, 38, 40
In	8, 9, 10, 16, 17, 22, 23, 24, 25, 28, 29, 30, 31, 32, 37, 39
Is	10, 11, 12, 18, 19, 24, 25, 26, 27, 30, 31, 32, 33, 34, 39, 41
Iw	2, 3, 4, 10, 11, 16, 17, 18, 19, 22, 23, 24, 25, 26, 31, 33
Ie	16, 17, 18, 24, 25, 30, 31, 32, 33, 36, 37, 38, 39, 40, 45, 47

0	7	14	21	28	35	42
1	8	15	22	29	36	43
2	9	16	23	30	37	44
3	10	17	24	31	38	45
4	11	18	25	32	39	46
5	12	19	26	33	40	47
6	13	20	27	34	41	48

Figure 1.1: Example nodes on the masked area

NB if you numbered differently, remake this example with your numbering system and include it in the report.

1.2 Assembling the Discrete Laplace [5 pts]

In this section you will assemble the Discrete Laplace operator for the unknown points and the boundary condition vector. This is done in the discretisation function 'DiscFun' which takes as inputs the list Neigh from the previous function and the destination image 'f'. The first output is the discretisation matrix 'D' which has '4' on the point corresponding to the Ic indices, and '-1' on each entry corresponding to an unknown neighbour (In, Is, Iw, Ie). The second output is the $(n \times m, 1)$ vector that contains the boundary values of those neighbours that are outside the unknown area.
Make sure that the discrete Laplacian is now solved only in the masked domain (points 0-15 as highlighted in red in figure 1.2).
A way to do this is as follows:

1) Convert the indices Ic, In, Is, Ie, Iw to the domain numbering. e.g. $9 \rightarrow 0$, $10 \rightarrow 1$....
2) If an index is the boundary, then allocate the correct value in the rhs vector of boundary conditions.

Once you have those lists, assembling the matrix should be straight forward.
Make sure to implement this efficiently and with the ability to handle very large sizes.
Test that you can produce the right matrix for the small example.

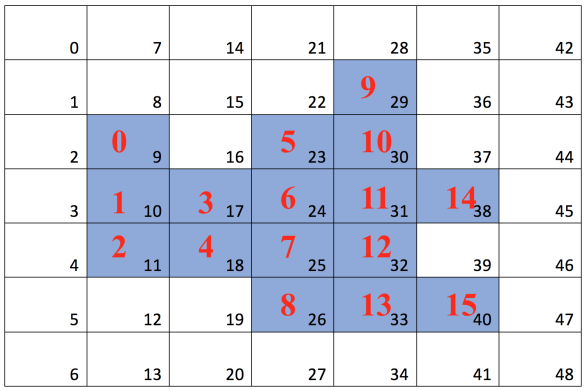

Figure 1.2: Example nodes of the domain

```
def  DiscFun( Neigh, f):
# Neigh from previous function
# f:= f^* (destination image)
# Output---
# D:= Discretization matrix
# bc:= rhs vector of the boundary values

# First convert Neigh to the new index list

# then assemble matrix and BC
    return D, bc
```

Produce your first image by solving the problem with $v = 0$ hence, only determined by the boundary conditions. Save your image in the first subplot of figure 1.5.

1.3 Guiding gradients [0.5 pts]

Write the form of equation 1.1 if we use v as the gradient of some scalar field ϕ. Obtain the right hand side (RHS) of equation 1.1 when we use ϕ as the source image g and destination image f^*. Embed the lines of code you used to generate the RHS. Plot the normalised fields multiplied by a factor of 10 (This is only necessary to plot them, as the gradients are quite weak)

(a) Guiding gradient destination image (b) Guiding gradient source image

Figure 1.3: Guiding gradients

```
#your few lines
```

1.4 Source guiding gradient [0.5 pts]

Having obtained the guiding gradient of the source image g, how do you modify the previous code to include the guiding gradient of the source image g ? Plot the result of solving eq. 1.1. in the second subplot of figure 1.5.

```
# few lines
```

What happens to the solution if you use the destination image f^* as a guiding gradient?

1 line answer

1.5 Mixed guiding gradient [0.5 pts]

How do you implement a mixed guiding gradient? where you take:

$$v = \begin{cases} \nabla f^* & \text{if } |\nabla f^*| > |\nabla g| \\ \nabla g & \text{otherwise} \end{cases} \tag{1.2}$$

```
your code (you may want to write a function for this)
```

Produce the resulting image and place it in subplot 3 of figure 1.5.

1.6 AN RGB IMAGE [0.5 PTS]

How do you modify the algorithm to apply it to a colour image? Show your result for a mixed guiding gradient in subplot 4 of figure 1.5.

Answer

```
# few important lines,I will check the main function
```

(a) Boundary only result

(b) Source guiding gradient

(c) Strongest Gradient

(d) Colour result

Figure 1.4: Your answer

1.7 IMPRESS US [0.5 +0.5]

Apply the algorithm developed or a different type of RHS to an example of your choice, we will give 0.5 extra mark for going above and beyond the class material. Plot together the source, mask, destination and final image. You might find useful the provided scripts Masking.py and roipoly.py that will help you generate the mask for an image (read the *README* file before using them). Otherwise develop your own mask and way to adjust it.

(a) Source

(b) Mask

(c) Destination

(d) Final result

Figure 1.5: Your answer

2 SECTION B: FRACTAL GROWTH [9 PTS]

In this section you will develop a code to simulate fractal growth. It is based on the steady state solution of the Fick's equation augmented by a stochastic element. We suggest you to make use of the function(s) developed in PART A. If you use those functions as they are, you don't need to show them again, just show how you use them.
You will grow the fractal in a 256× 256 unit square domain starting from a seed in the middle. The equation you solve is given by the following:

$$\begin{aligned} \nabla^2 f &= 0 && \text{on } \Omega, \\ f &= 1 && \text{on } \partial\Omega_f \\ f &= g(r) && \text{on } \partial\Omega_b \end{aligned} \tag{2.1}$$

where the boundary on the fractal edges are denoted by $\partial\Omega_f$ and the box boundaries are $\partial\Omega_b$. $g(r)$ is a function of the distance between the fractal and the edge (r) but for the first part of the problem take $g(r)$=0 on all the box boundaries.

2.1 THE DOMAIN [0.5 PTS]

Use the function 'Neighbours' from the previous part to obtain the domain in which you will solve the Laplace equation (2.1).

Explain how you use it

```
# explain what is the input matrix
# one line to call the function
```

2.2 THE SOLUTION IN THE DOMAIN [0.5 PTS]

Use the previous function to obtain the discretisation matrix D and the boundary conditions to solve problem of equation 2.1 in the domain obtained from the previous point.

Explain how you use it

```
# explain what is the input matrix
# one line to call the function
```

2.3 THE NEIGHBOURS OF THE FRACTALS [1 PTS]

How can you obtain the indices of the neighbouring cells of the fractals? How do you obtain the value of the potential field for these cells?

Explain if you use previous functions

```
#  lines of code
```

2.4 Deposition [4 pts]

You will deposit a new seed in the neighbour specified by the algorithm explained in class. Complete the function 'Deposition' such that it returns the index of the location where the new seed has to be placed.

```
def Deposition(IB, p, alpha):

    # IB := indices of the neighbours of the fractals
    # p:= solution field everywhere
    #alpha := exponent
    # outputs:---
    # Idx := index to place next particle

    return Idx
```

2.5 Fractal growth [1 pts]

Add the seed to the location given by 'Idx' from the previous function and repeat the routine until one of the seeds is deposited on the boundaries. Produce 2 plots, one with $\alpha = 1$ and one with $\alpha = 3/2$. How long did your code take to run?

.... minutes

2.6 Boundaries [0.5 pts]

Change the boundary conditions on the box such that they are:

$$g(r) = \frac{1}{2\pi}\ln(r) \tag{2.2}$$

where r is simply the distance from the centre of your square.
Produce the results for 4000 iterations. How long did your code take to run?

.... minutes

2.7 IMPRESS US [0.5 PTS]

Make up your own fractal. Change boundaries and/or seeds and produce a cool fractal. Describe what you did.

(a) Fractals $\alpha = 1$

(b) Fractals $\alpha = 3/2$

Figure 2.1: Your answer

(a) Fractals $\alpha = 1$

(b) Fractals $\alpha = 3/2$

Figure 2.2: Your answer

Figure 2.3: Your fractal