Seamless Cloning

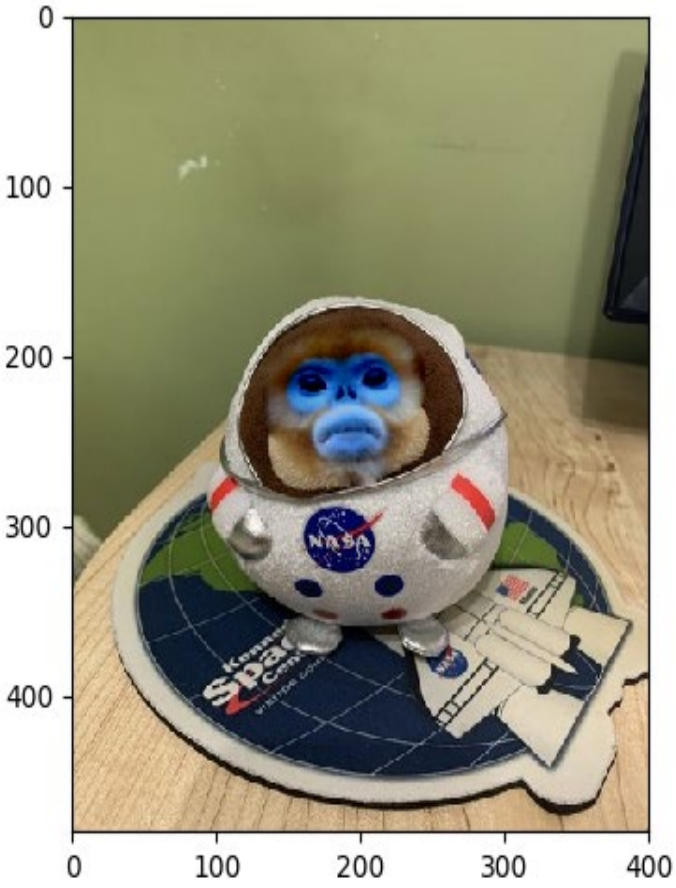
Tamer Ghattas | Computational Photography | 23.5.2021

Part A

I've followed the paper in solving the equation #4:
$\Delta f = div\ v$ over Ω with $f|_{\partial\Omega} = f^*|_{\partial\Omega}$

In the implementation I've used a binary mask as Omega and have built a linear equation where for each pixel, a variable in the equation system, and set equation inside the mask based on Poisson discrete matrix while set outside the mask as the identity matrix. Whereas on the right hand side of the equation, is a vector having a pixel inside the mask the value of operating div g = div "source" and target value for those outside the mask.

Results:



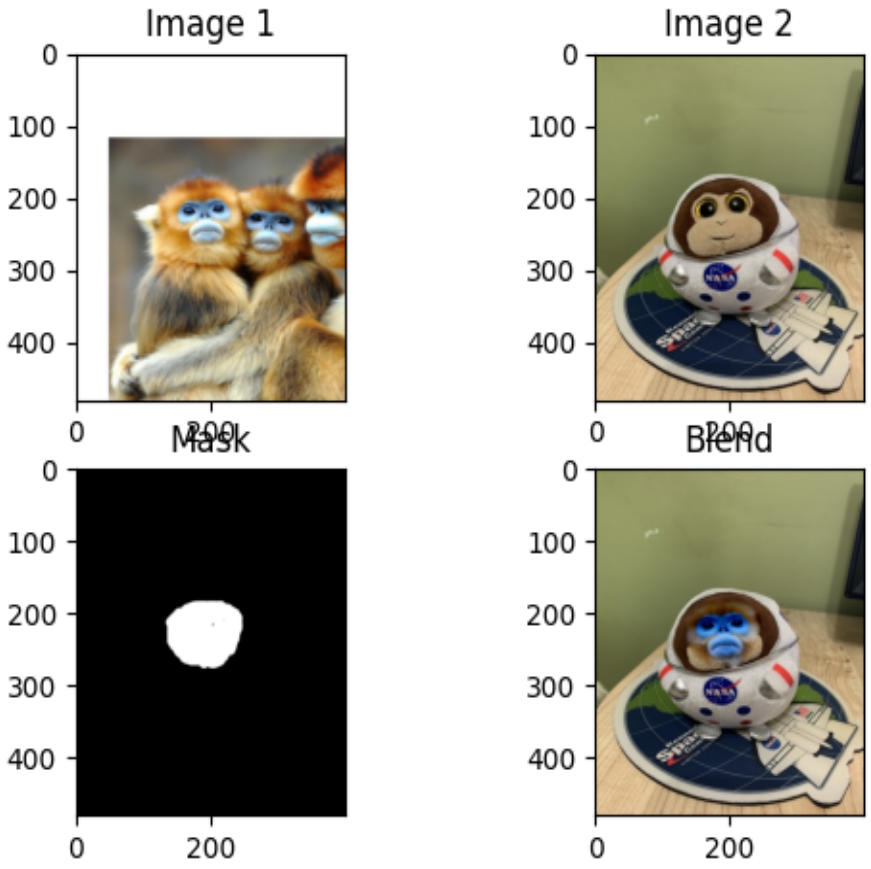
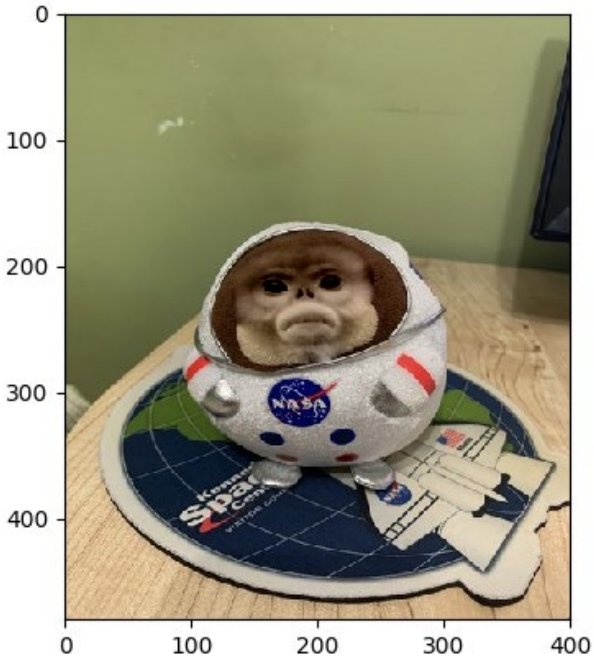
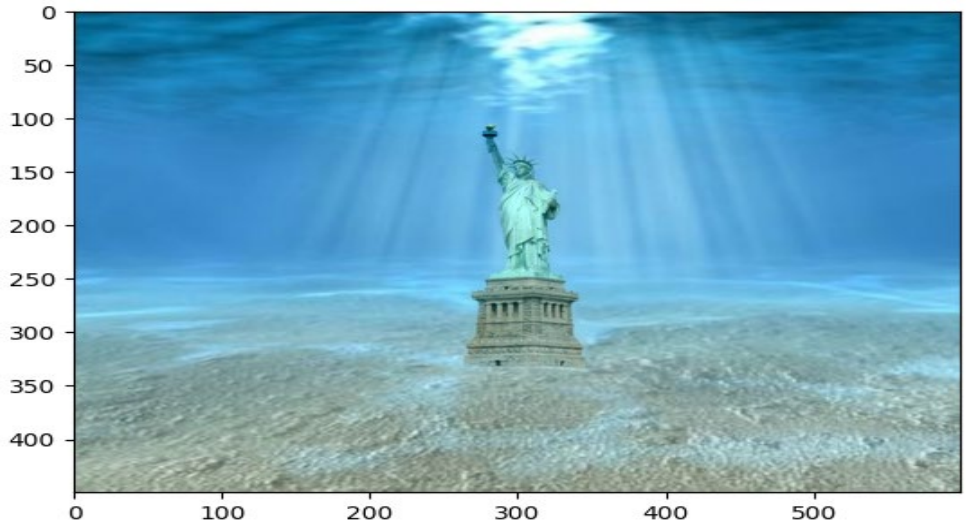


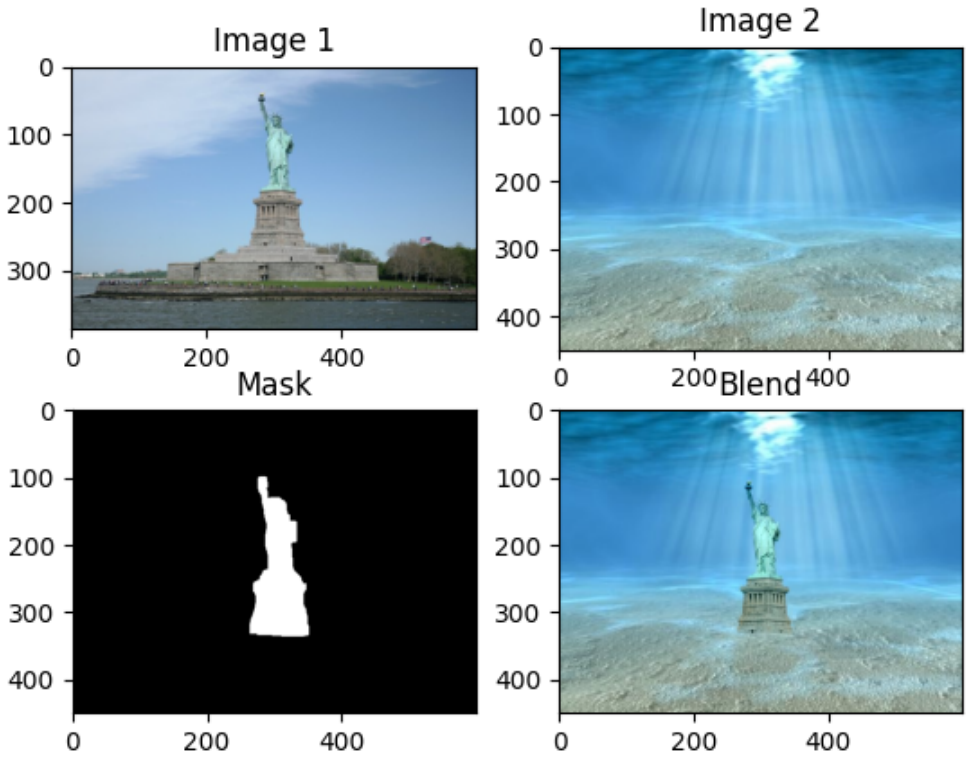
Image 1
Image 2
Mask
Blend
0
100
200
300
400
0
200

Since there has been darkening to the center of the source image monkey face color, I've also tried using the source as monochromatic and got:



Additional example:

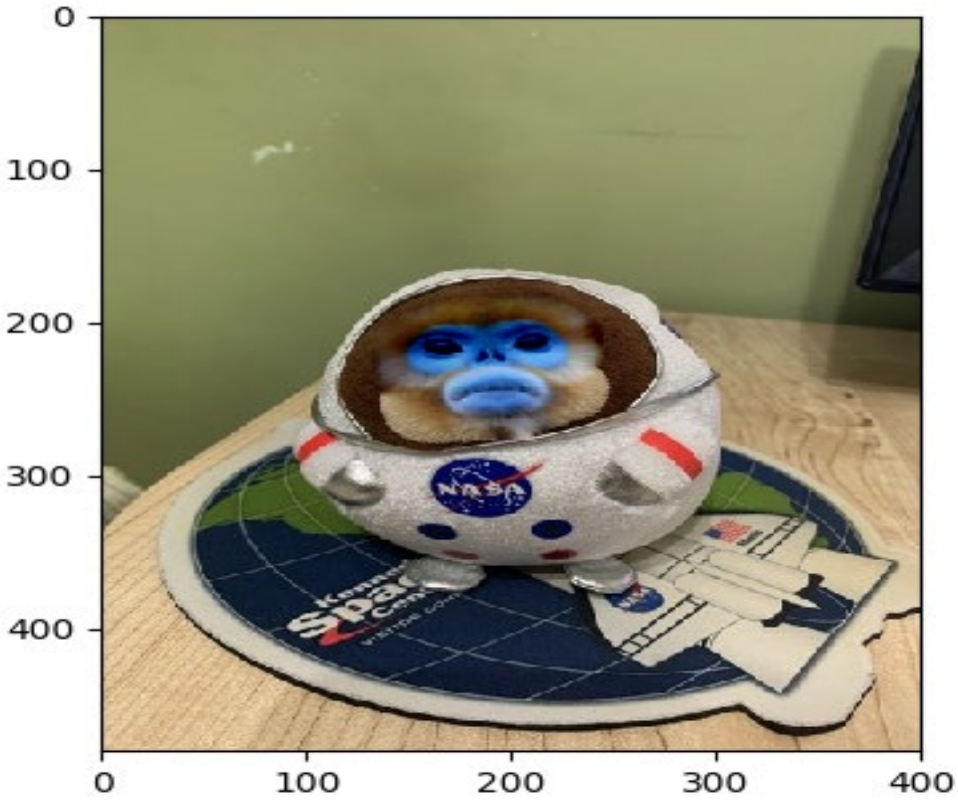




Part B

As for this part, I've followed the revert section in the "Convolution Pyramids" paper and built in the implementation a Shepard's interpolation kernel.

Results:



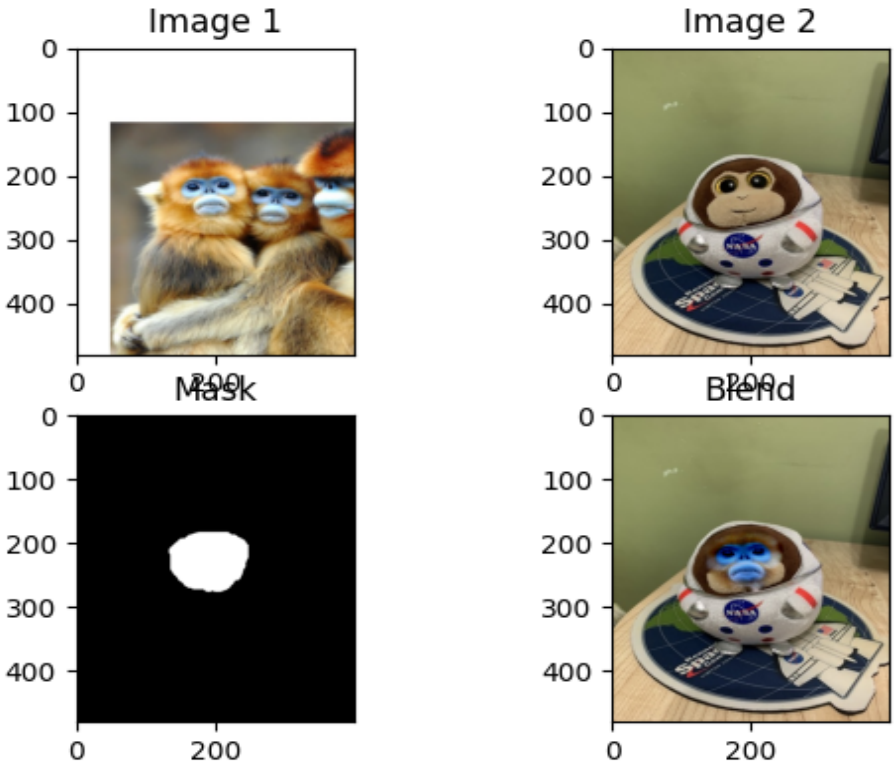
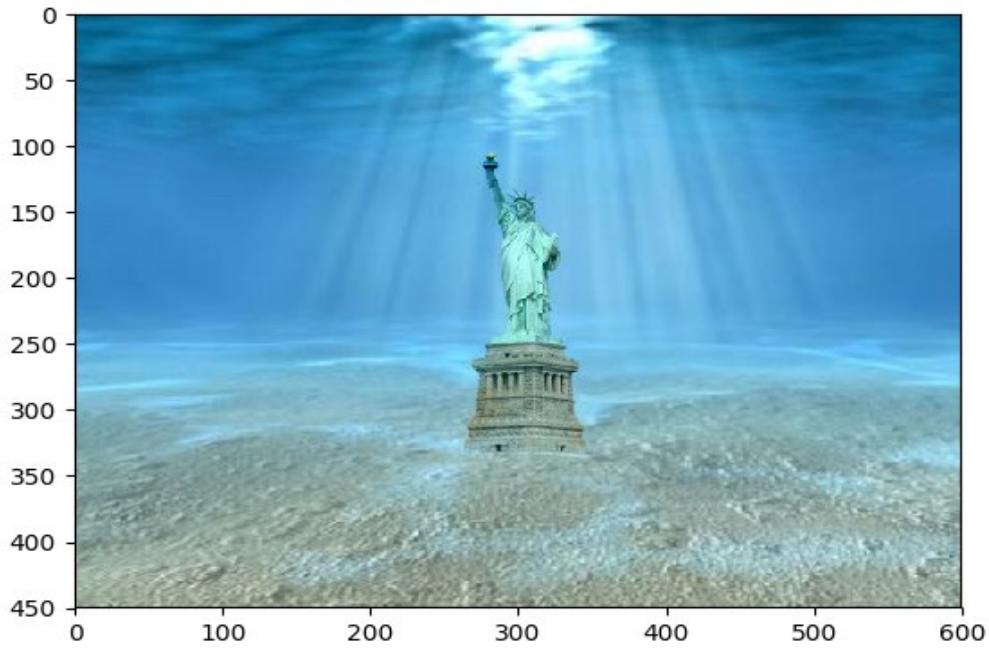
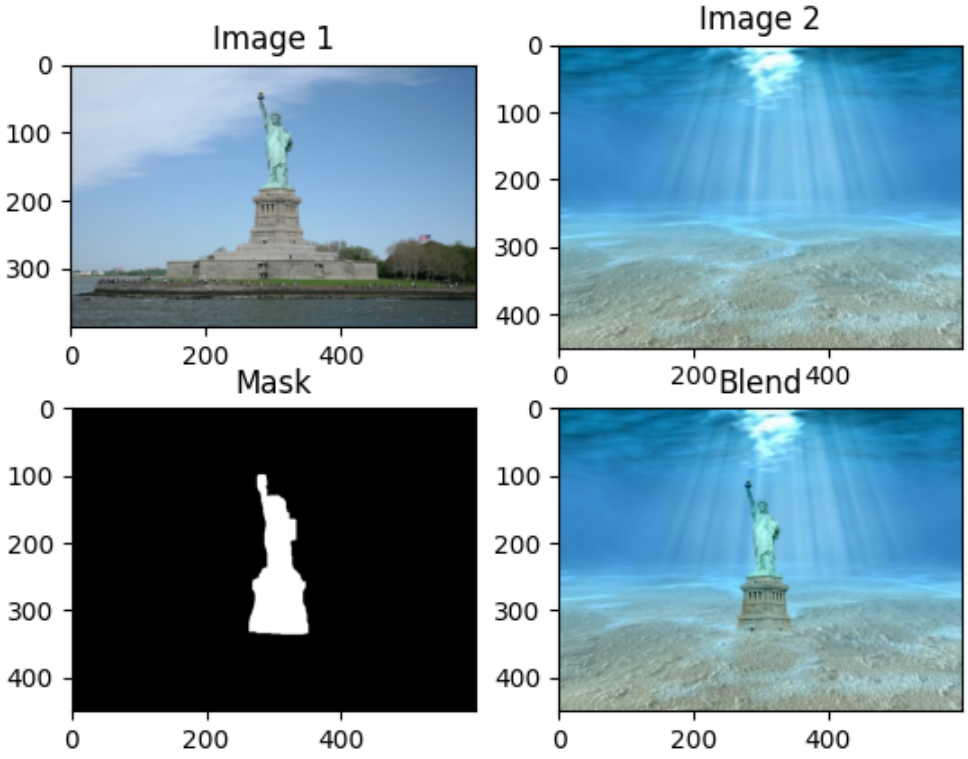


Image 1
Image 2
Mask
Blend
0
100
200
300
400
0
200

Additional example:





Part C

In the time complexity analysis for both parts we can tell that the heaviest operation is the seamless cloning as all the other preparations (copying, resizing, and finding Euclidean distance of single element) are a linear function of N, the number of pixels in the biggest image of source, target and mask. Therefore, we can state that since Shepard's based convolution uses cv2.Filter2d which uses the frequency domain to apply the filter, the time complexity of the blending is bound by O(N log N).
However, in Poisson based solver, for the mixed gradient and the standard source gradient it builds the blend by solving a sparse linear equation using LU factorization which is bound by $O(N^3)$.

From profiling the execution on example 1 in each method on Macbook Pro mid-14, intel i5, 8 GB RAM**:**
Shepard's seamless cloning RGB: 100%|██████████| 3/3 [00:00<00:00, **8.56it/s**]
Poisson seamless cloning RGB: 100%|██████████| 3/3 [03:36<00:00**, 72.16s/it**]

Where one iteration corresponds to one of the 3 RGB channel processing.

Part D

I have implemented a simple GUI using open-cv interface API.

Usage:

python run_clone.py [options]

Options:

-h Flag to specify a brief help message and exits.

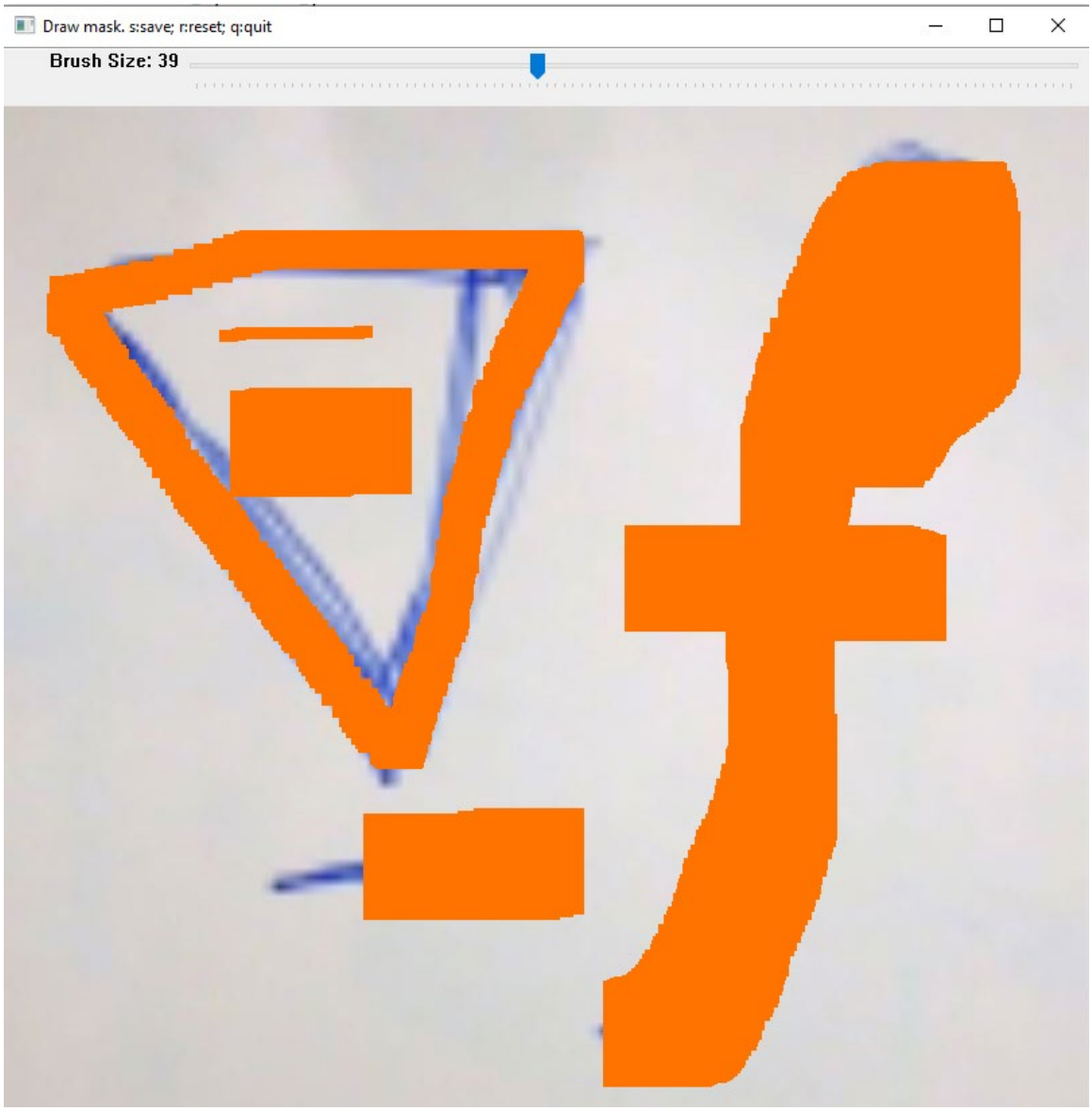
-s (Required) Specify a source image.

-t (Required) Specify a target image.

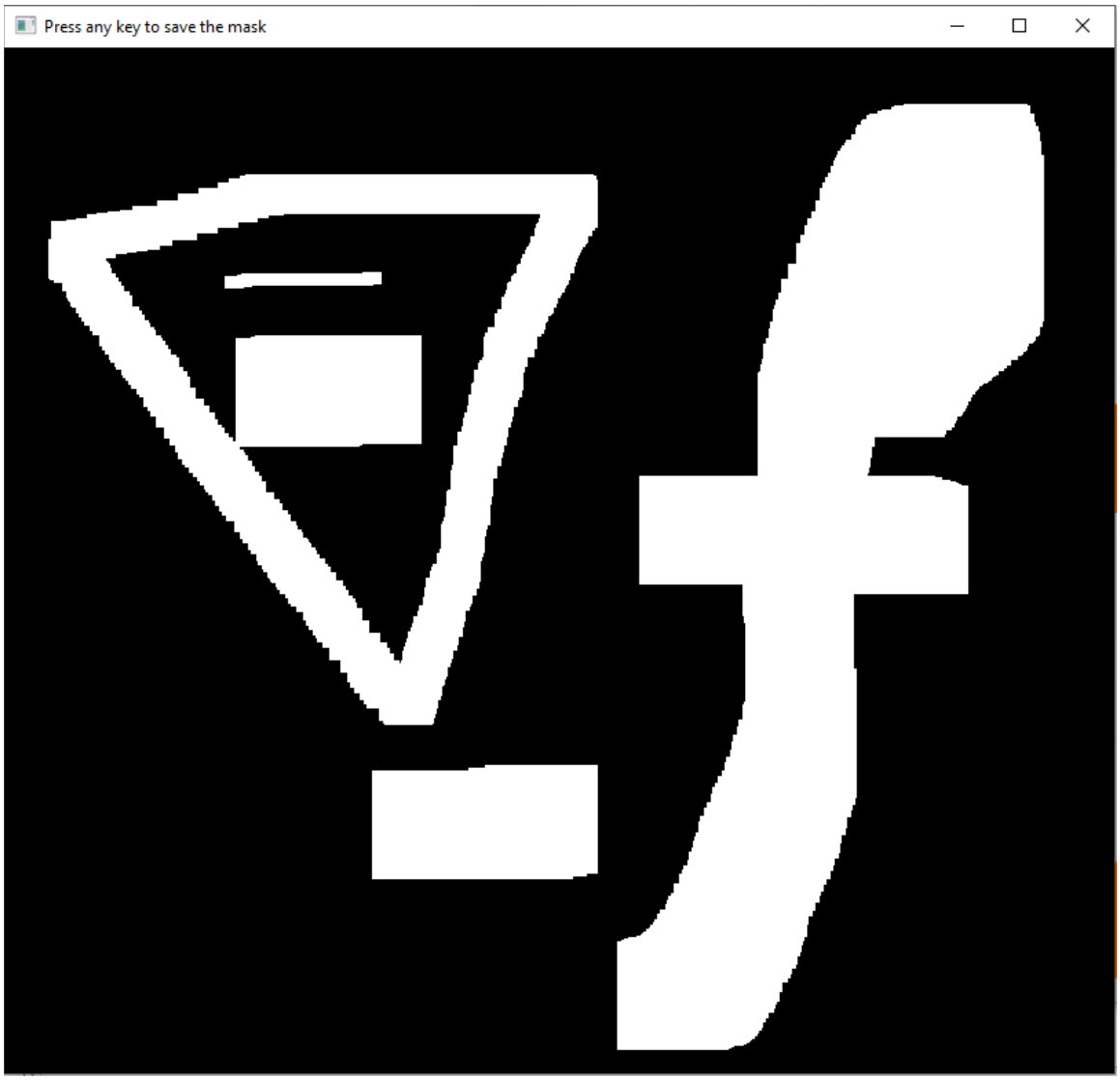
-m (Optional) Specify a mask image with the object in white and other part in black, ignore this option if you plan to draw it later.

-x (Optional) Flag to specify a mode, either 'poisson' or 'shepard'. default is Poisson.

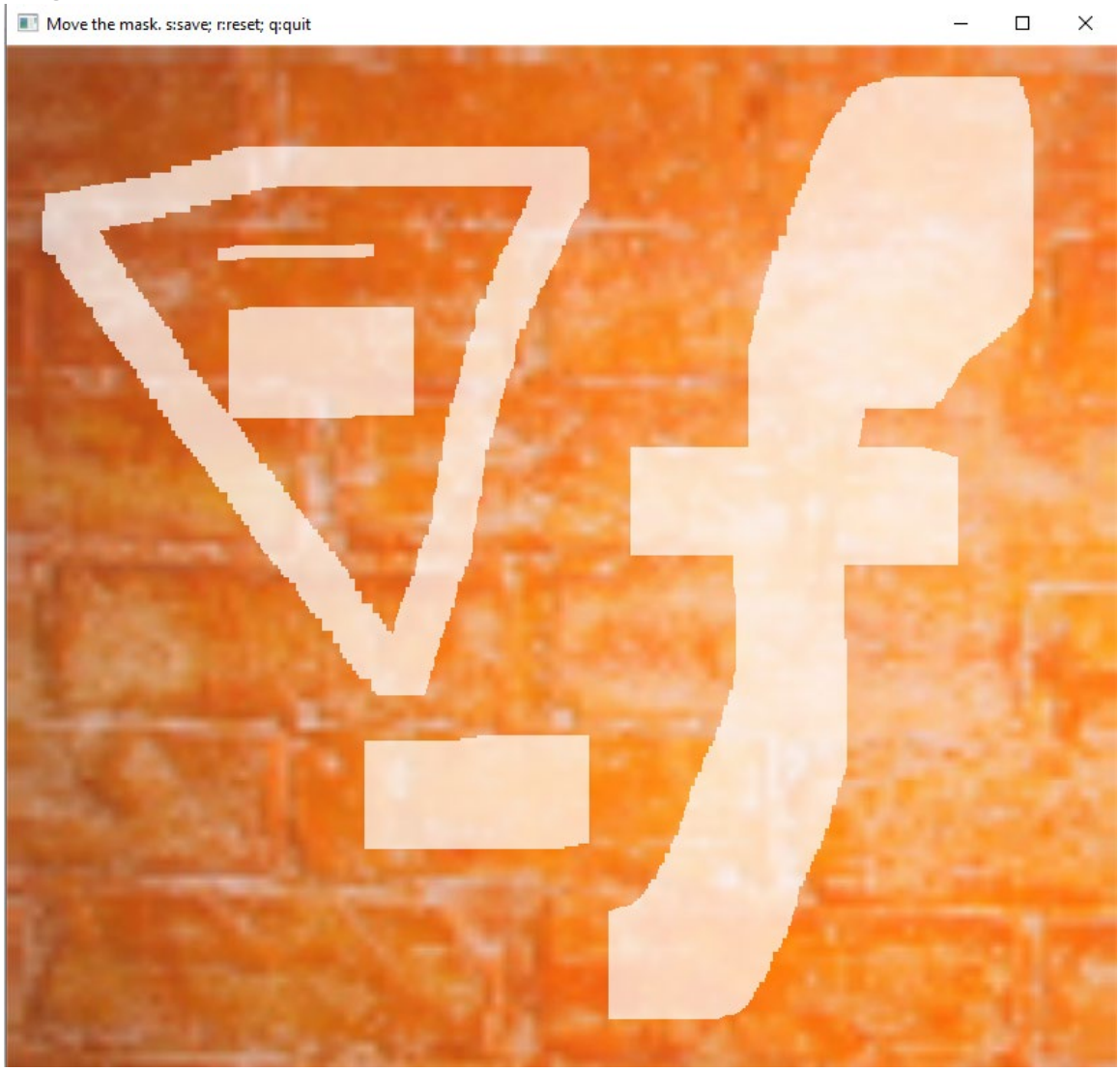
In the mask painter, I've put a slider to enable controlling the brush size:



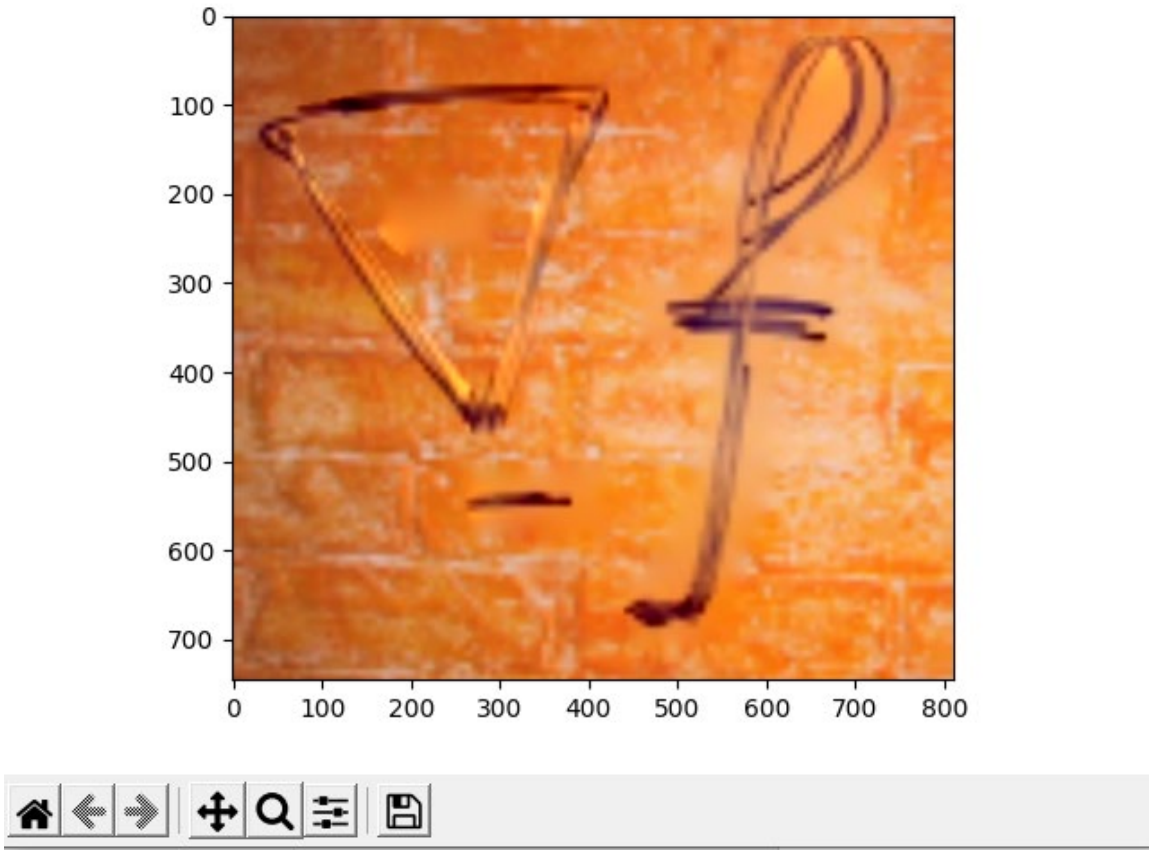
When finished and clicked 's' for save, a new window will open to confirm and save drawn mask:


Press any key to save the mask

When saved, additional window will pop up to enable emplacing the mask on target image:



And finally when saving the positioned mask, the selected cloning algorithm will operate on the images and output the result:



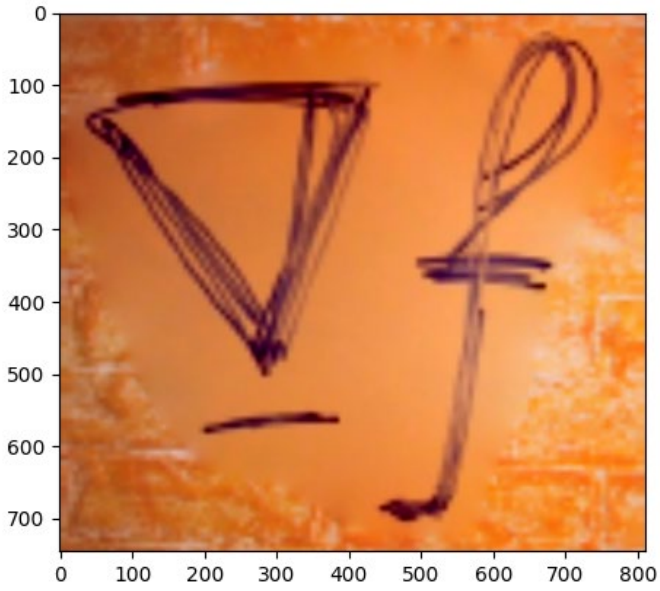
Part D

I've added for Poisson solver of part a the ability to pass a special gradient field.
One of which I've implemented is the one from the paper equation 12:

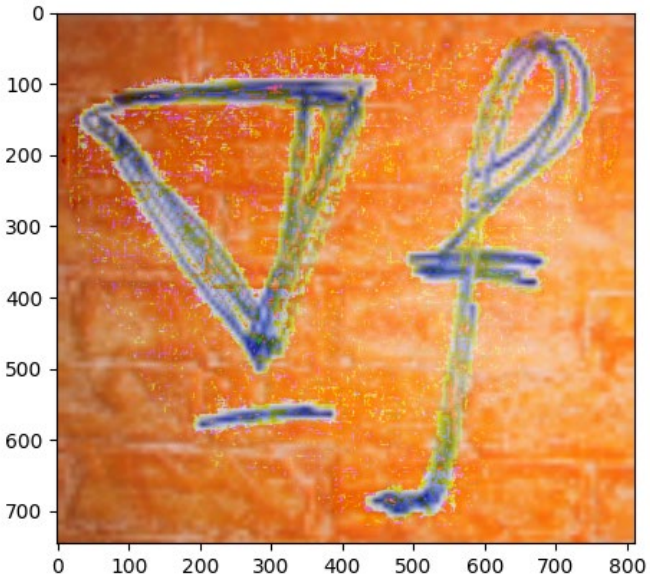
$$\text{for all } \mathbf{x} \in \Omega, \mathbf{v}(\mathbf{x}) = \begin{cases} \nabla f^*(\mathbf{x}) & \text{if } |\nabla f^*(\mathbf{x})| > |\nabla g(\mathbf{x})|, \\ \nabla g(\mathbf{x}) & \text{otherwise.} \end{cases} \quad (12)$$

In the result we can see the advantage of not blurring the background where there are no written letters, however, it comes with bad color artifacts.

Without the special gradient field:



With the (12) equation gradient field:



Modules Description

clone.py - have the main logic of both Shepard's interpolation & Poisson based solver.
demo.py - includes examples.
gui.py - implements cv2 based interactive GUI for marking mask on source image.
run_clone.py - CLI for the solution.
utils.py - includes basic auxiliary functions for image processing and a Gaussian pyramid based blending implementation.