

# Assignment 1 - Foreground Segmentation and Poisson Blending

Idan Drori                    Alon Zajicek  
205983406                    324171768

December 27, 2024

## 1 GrabCut

Table 1: Metric Results table - `n_components = 5`

Image Name	Accuracy	Jaccard	Runtime (in seconds)
banana1.jpg	0.6701	0.4367	82.37
banana2.jpg	0.9925	0.9689	68.32
book.jpg	0.9563	0.8942	157.46
bush.jpg	0.8692	0.5591	42.05
cross.jpg	0.5519	0.4514	36.17
flower.jpg	0.9965	0.9821	36.30
fullmoon.jpg	0.9681	0.6543	16.03
grave.jpg	0.9877	0.9042	46.66
llama.jpg	0.9887	0.9359	48.89
memorial.jpg	0.9899	0.9453	37.46
sheep.jpg	0.9957	0.9227	38.95
stone2.jpg	0.9961	0.9842	85.81
teddy.jpg	0.9907	0.9580	24.22
		<b>Average Runtime:</b>	<b>55.44</b>

### 1.1 Examples where GrabCut worked decently well:



Figure 1: banana2.jpg

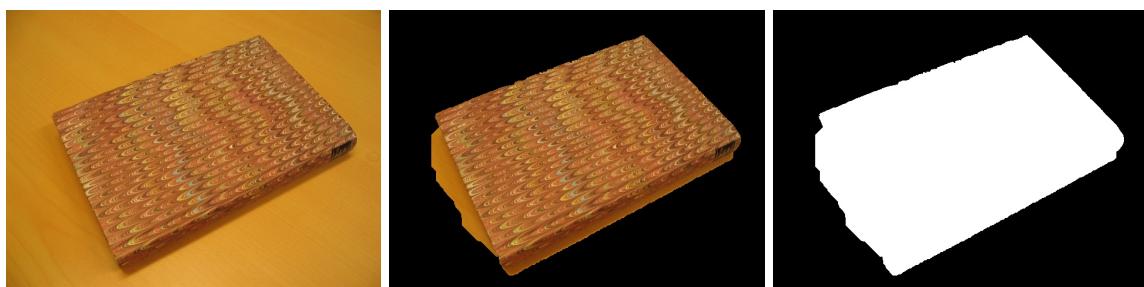


Figure 2: book.jpg



Figure 3: flower.jpg



Figure 4: grave.jpg



Figure 5: llama.jpg

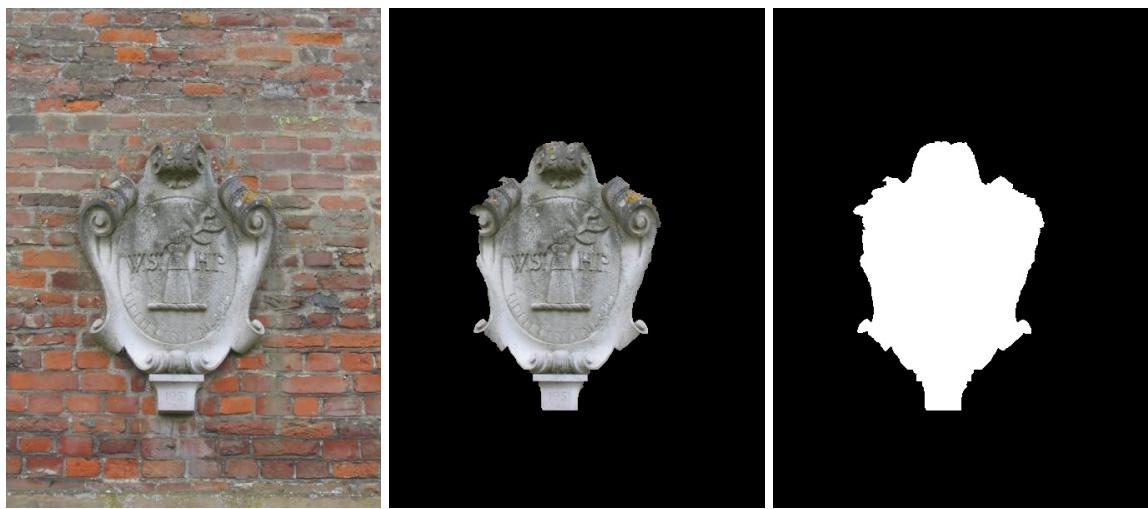


Figure 6: memorial.jpg



Figure 7: sheep.jpg



Figure 8: stone2.jpg



Figure 9: teddy.jpg

## 1.2 Examples where GrabCut did not work well:



Figure 10: fullmoon.jpg



Figure 11: banana1.jpg



Figure 12: bush.jpg



Figure 13: cross.jpg

For `bush.jpg` what I assume happened is that the background had too similar a color to the foreground, so it was difficult to truly separate the thin trunk and leaves from the background grasses. Regarding `banana1.jpg`, `cross.jpg` and `fullmoon.jpg` I suspect that the issue is that the GMMs have too many components. Perhaps too many components also played a part of the failure of `bush.jpg`

### 1.3 Effect of various parameters on GrabCut:

We'll first test the effect of a different number of `n_components` on GrabCut:

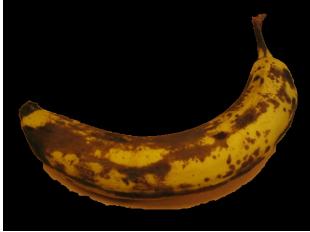
Img Name	n_components		
	1	3	5
banana1.jpg (Accuracy, Jaccard)			
cross.jpg (Accuracy, Jaccard)			
bush.jpg (Accuracy, Jaccard)			
fullmoon.jpg (Accuracy, Jaccard)			

Table 2: Comparison of Accuracy and Jaccard across n\_components

It seems that for `banana1.jpg` and `fullmoon.jpg` the optimal number of `n_components` is 3, while for `cross.jpg` and `bush.jpg` the optimal number is 1. Although the segmentation in `bush.jpg` is still quite bad.

Now we'll look at the effect of different amounts of blur on GrabCut's performance:

Img Name	blur		
	low	high	no blur
flower.jpg (Accuracy, Jaccard)			
llama.jpg (Accuracy, Jaccard)			
teddy.jpg (Accuracy, Jaccard)			

Table 3: Comparison of Accuracy and Jaccard across different levels of blur

It seems like blurring the image resulted in a worse outcome than not blurring the image. Specifically, higher levels of blur resulted in worse results than low levels of blur which in turn was worse than no blur at all.

Now we'll test different initializations of rectangles on the final result of GrabCut. We'll use the default rectangle that's defined in `\data\bbboxes\image_name.txt`, a larger rectangle and a rectangle that covers almost the whole image (its  $x$  and  $y$  are both zeros, and its width and height are 1 pixel less than the image's width and height) and see whether it has an effect on the final result or the runtime. It should be noted that in the unmodified skeleton, the pre-defined rectangles go past the width and height of the image, meaning the rectangle's bottom edge and right edge are below and to the right of image's edges respectively. I modified the skeleton such that it subtracts the  $x$  and  $y$  coordinates from the rectangles  $w$  and  $h$ . This makes the pre-defined rectangles much tighter and prevents some bugs.

Img Name	rectangle		
	default	larger	largest
flower.jpg			
(Acc, Jaccard, Runtime)	(0.9965, 0.9821, 37.51s)	(0.9961, 0.9805, 52.86s)	(0.3953, 0.2426, 37.80s)
sheep.jpg			
(Acc, Jaccard, Runtime)	(0.9957, 0.9226, 38.51s)	(0.9956, 0.9210, 42.61s)	(0.3953, 0.2426, 36.80s)

Table 4: Comparison of Accuracy, Jaccard similarity and runtime with different rectangles

We can clearly see that a tighter rectangle is better. Larger rectangles mean more chance of noise, which lowers accuracy. And the largest possible rectangle just gives terrible results. Regarding runtime, it doesn't seem like rectangle size affects runtime in any meaningful way.

## 2 Poisson blending

we implemented the discrete version of the Poisson blending algorithm we saw in class and ran it on random pairs of foreground images, detected by our GrabCut algorithm, and background images. As you have seen before GrabCut did not do too well on some images, so we will compare them to a tight mask given with the homework. The following figures will be the grab-cut result, followed by the detected mask and the blending result.



Figure 14: banana1 blended with a table

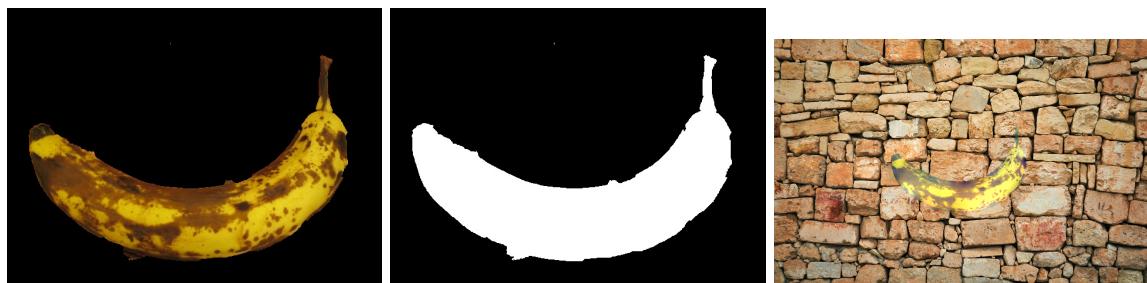


Figure 15: banana2 blended with a wall

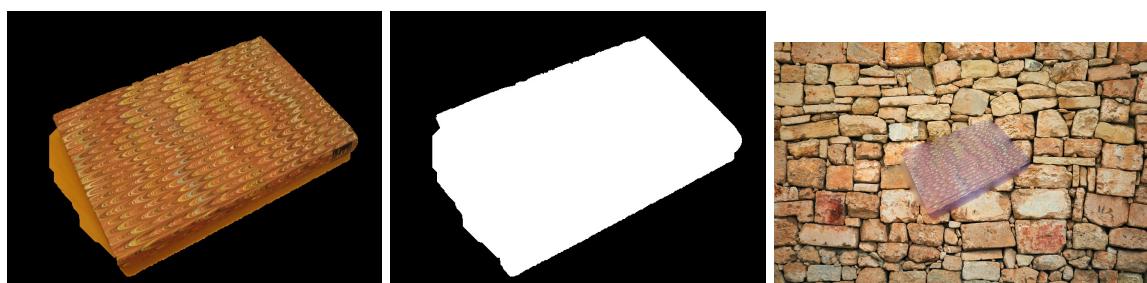


Figure 16: book blended with a wall

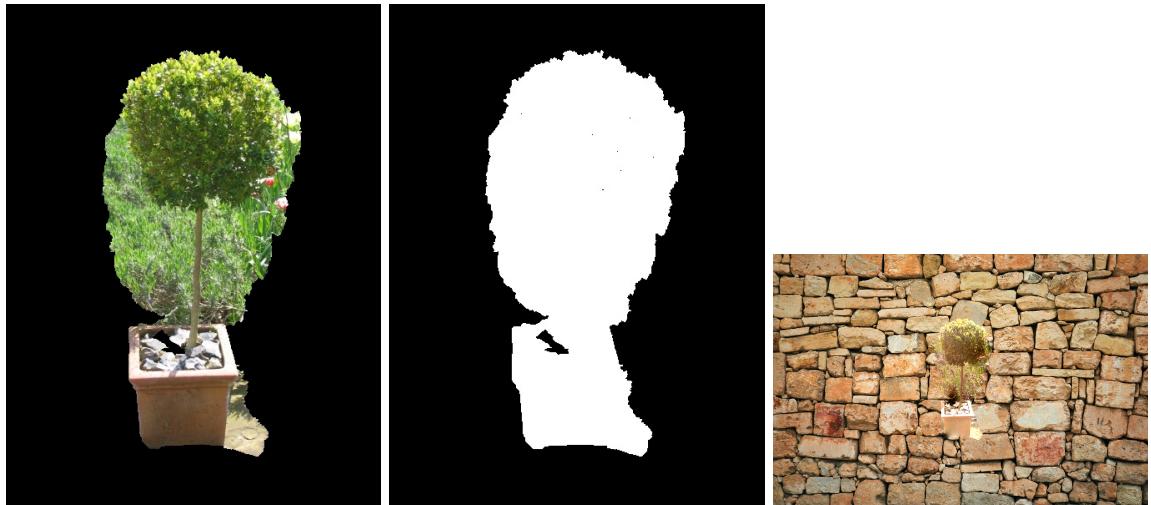


Figure 17: bush blended with a wall



Figure 18: cross blended with a table

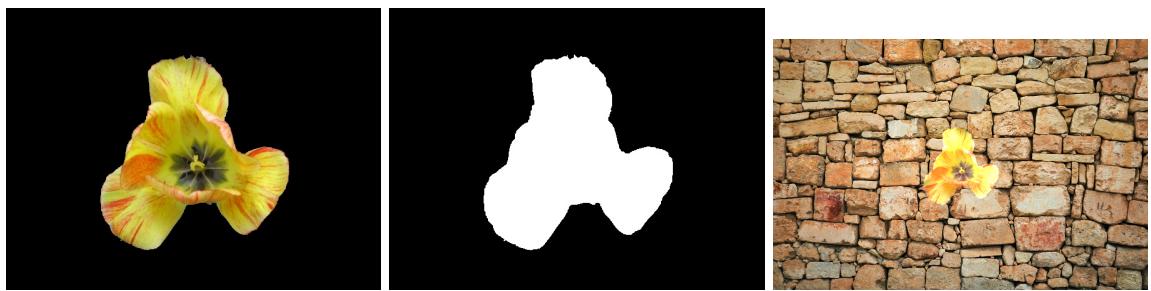


Figure 19: flower blended with a wall

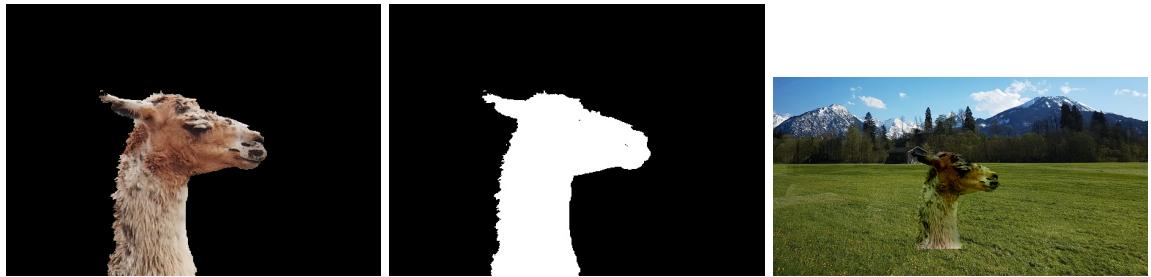


Figure 20: flower blended with a grassy mountain



Figure 21: memorial stone blended with a table

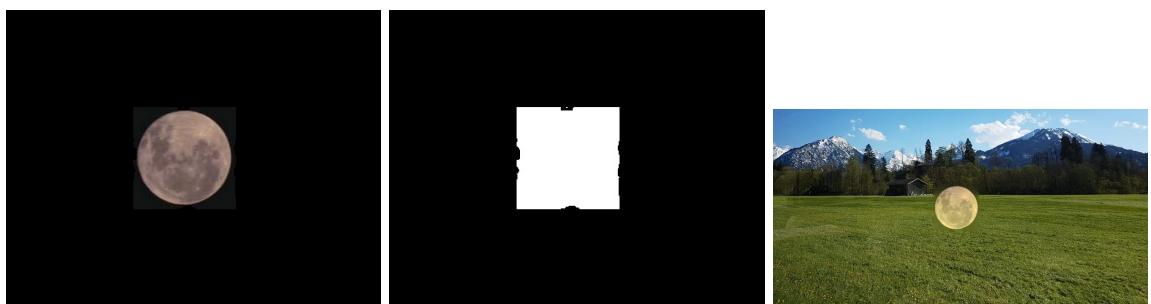


Figure 22: full moon blended with a grassy mountain

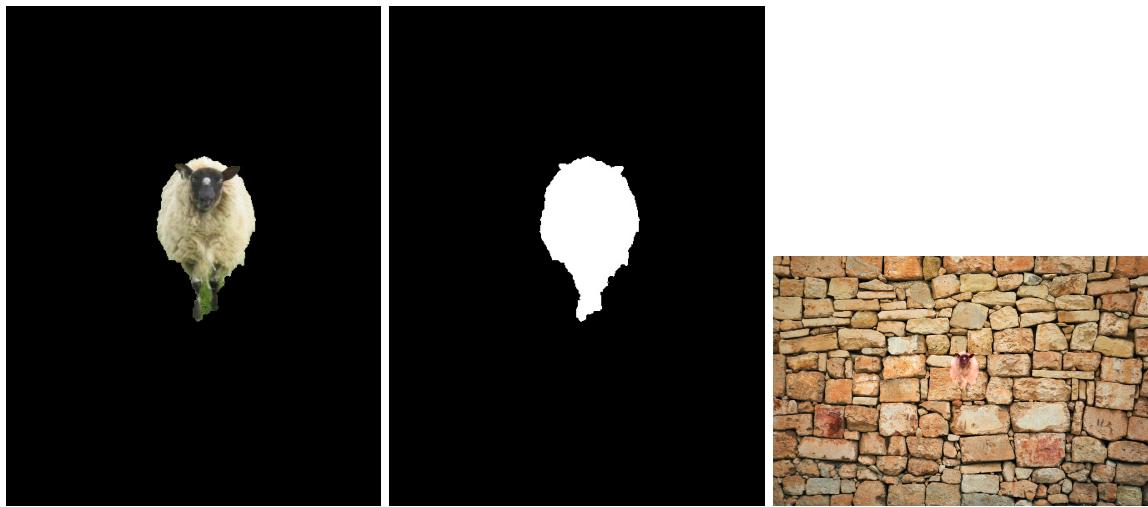


Figure 23: sheep blended with a wall

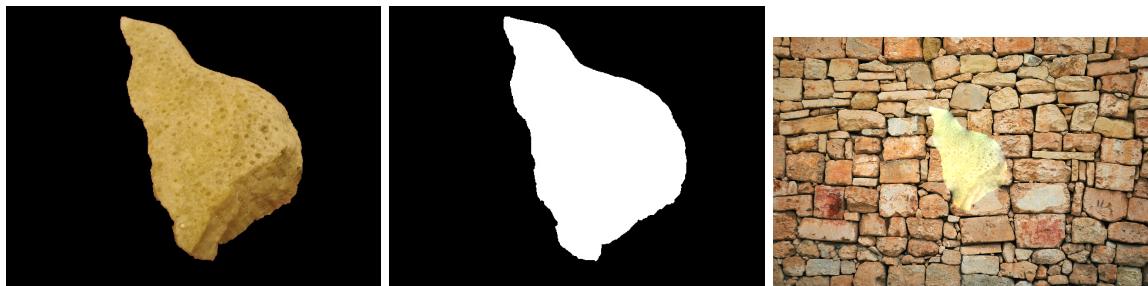


Figure 24: stone blended with a wall



Figure 25: teddy bear blended with a table

In the previous examples, we can see that we encountered two main problems, one of the colors fading, and the other of blurriness.

## 2.1 Fading colors problem

The fading colors can be seen in the teddy bear blended with the table. This problem is known with Poisson blending, and we talked about in class. Both the background is very bright and the original picture contains bright color pattern, so when we blend it with the gradients from the table, most of the details fade. In the following figure, we can see that when combined with a darker background the details stay a lot better. In class we saw that this problem can be avoided when working in the log domain.



Figure 26: teddy bear blended on bright vs dark background

## 2.2 Blurriness problem

The second problem we encountered is the blurriness problem. It looks like this problem occurs when the mask is not very tight around the desired foreground object. This results in a blurred area and unwanted colors from the source picture. In the following figures, we show a tight vs. loose mask of the same source pictures with the same background picture.



Figure 27: banana1 blended with a table with loose vs. tight mask



Figure 28: full-moon blended with a grassy with loose vs. tight mask

We can see that when the mask is tight we get both less blurry results, and the blending with the background picture looks much more natural.