Computer Graphics – 1

Eldad Ligishal 316460815   Guy Paz Ben Itzhak 315328963

GrabCut Algorithm

We implemented the GrabCut algorithm according to the instructions given in the assignment.

First, we initiated a Kmeans object and then used its parameters (such as means and covariances) to initialize a GMM (Guassian Mixture) object. We used existing libraries for both objects.

After that we start a loop where in each loop we update the GMM using the new image pixel definitions and we calculate the relevant parameters for the object accordingly. We then use a prediction calculate the GMM's new parameters: means, weights and covariances.

We then go to building the graph and calculating the minimum cut within it. We build the n-links only once and use a separate graph to calculate the K parameter. In addition, we calculate the t-links, which their capacities we calculate with a probability function (taken from the articles). We add the edges and capacities to the graph and then calculate the mincut, which returns the vertices' partition.

The final step of the iteration is to update the mask's values – we update from probably foreground to probably background. We then check the energy's convergence: if the difference between the current and previous energies is less than 0.1% then we announce convergence and finish the run.

During the implementation we encountered several problems that we were able to overcome. Those included a wrong number of edged added to the graph, wrong update of GMMs in each iteration and not updating the mask correctly.

However, when running the code, we saw that our algorithm implementation did not work properly. We can see that the mask almost didn't change. During the run we saw that in each iteration there weren't many changes to the partition sides. It may be caused due to a incorrect building of the graph, or not updating the GMMs correctly.

We have tried repeatedly to fix several issues (which were discussed above) in order to make the algorithm work but didn't succeed. We think we might have a problem with updating the mask, we made efforts to solve it but couldn't resolve it.

We noticed that as the object is smaller, the cut gets better and it can be seen in images such as memorial, sheep and bush. It is probably because there is a lot more place to get closer into the object.

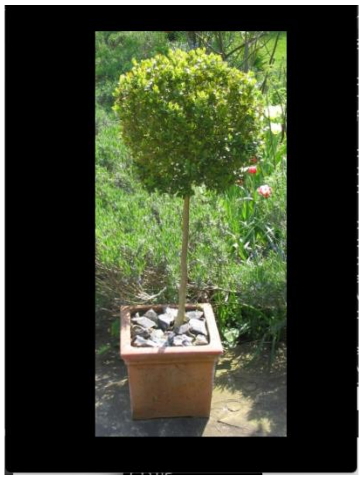
Even though, we tried to achieve some qualitative and quantitative evaluations of our results. They can be seen in the next page.

Img name	Accuracy	Jaccard
Banana1	0.43828450520833334	0.31323627738155885
Banana2	0.490263671875	0.3164949803579223
Book	0.57818359375	0.4671025846647338
Bush	0.697062962962963	0.3567457040619716
Cross	0.3783074074074074	0.37229069750048616
Flower	0.778074074074074	0.4659964886952027
Fullmoon	0.9664318034906271	0.6423800013773157
Grave	0.9097259259259259	0.5701234567901234
Llama	0.8165487092994541	0.4893228023987129
Memorial	0.8356814814814815	0.5292782039447858
Sheep	0.9420851851851851	0.47298709177311177
Stone2	0.7297526041666667	0.4728888888888889
Teddy	0.6996868143534575	0.4158291457286432
Banana1 (with 8 clusters)	0.43828450520833334	0.31323627738155885
Banana1 (with a blur)	0.43828450520833334	0.31323627738155885
Banana2 (with moon rectangle)	0.490263671875	0.3164949803579223

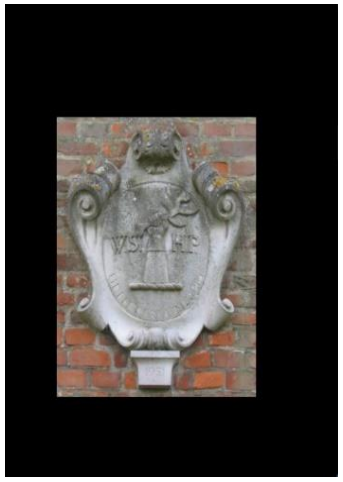
All implementations run with 2 clusters (except the last one), and all runtimes took between 10-20 seconds. Because of the problem in our implementation, it seems like there is no difference in accuracy of image if we change the number of clusters or add a blur to the image. The blur is supposed to affect the edges of the banana and since our cut is far from the edges, we get the same result. It seems though that in the book image it did a better job. Probably because of the size of the image.

We can also see that our classifications are only rectangulars, but we couldn't distinguish the reason for that.

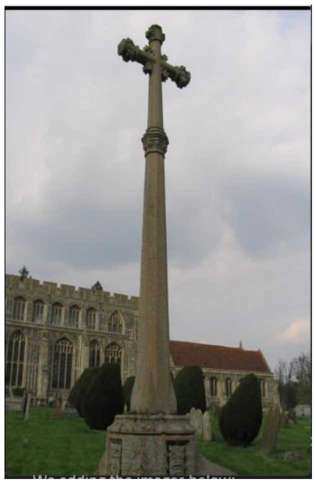
We are adding the images below:



Bush



memorial



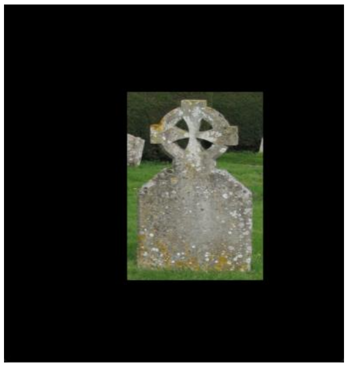
cross

Stone2

fullmoon

llama

Flower



grave

sheep

Banana2

banana1

book

Banana2 (with moon rectangle)

teddy

Poisson Blending Algorithm

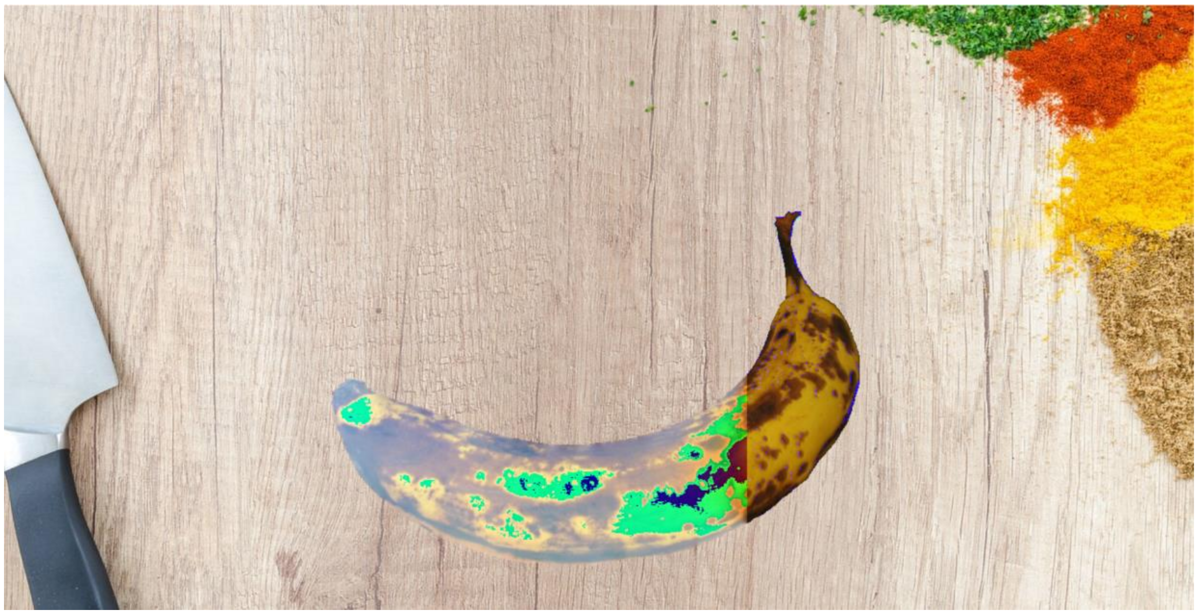
We implemented the Poisson Blending algorithm according to the presentation given in class.

The first step was to create a sparse matrix A by using coo matrices and then converting it to csr matrices. We adjusted the x vector to be in the size of the mask, since all of the other pixels in the blended image should be equal to the corresponding one in the target image and the only unknows may be inside the mask. For each black pixel in the mask, we made its corresponding row in A all zeros except one in the [i,i] index so that multiplying it would give us the corresponding target image pixel value (which we need to set up in the b vector). For all of the other pixels in the mask, if they are inside the shape (i.e. banana) we should just put -4 and ones in the corresponding places in the matrix' row to calculate the Laplacian and in the same index in the b vector we should put a calculation of Laplacian for the same pixel in the source image. we converted the pixels accordingly so the mask image should be at the center of the target image. For pixels in the sides of the shape we need to omit ones in the corresponding pixel's row in A and we need to substract those same values in the target image from the b vector.

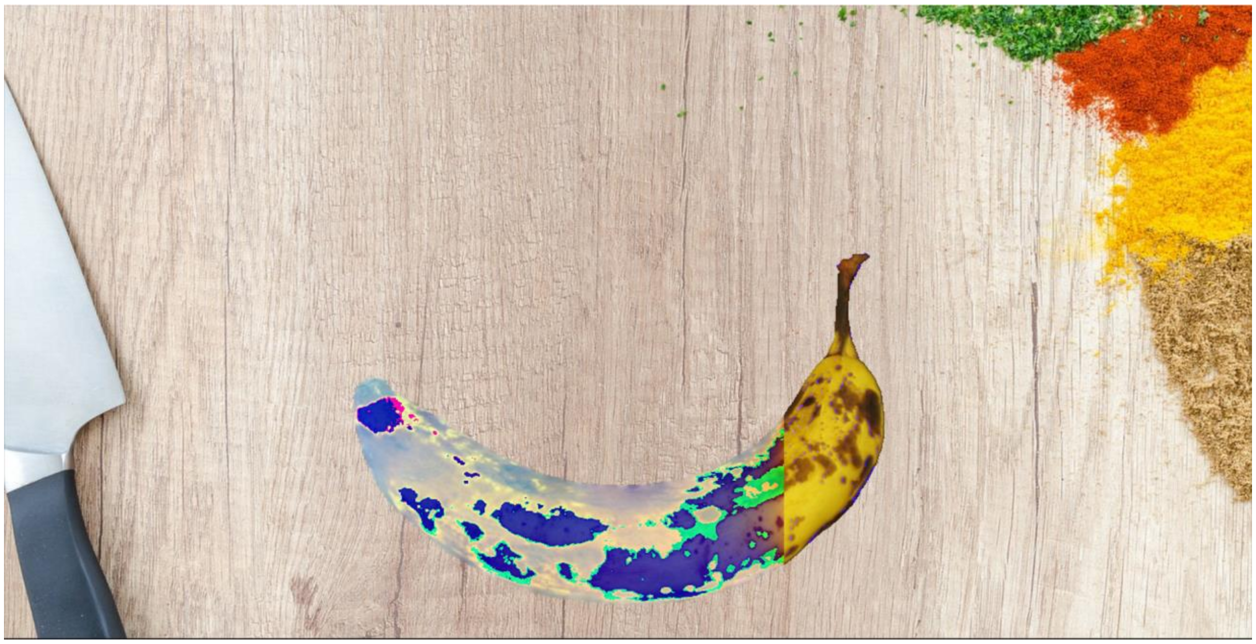
After that we have both A and b and we can calculate x using the spsolve method. Afterwards x's values are pasted to the appropriate pixels in the target image.

In our result of the banana1 image we see that some parts of the banana look pale and have some distortions. We might have some small issues with the Laplacian matrix that we weren't able to find that led to the distortions. As per the paleness, it might be caused by incorrect values in the b vectors.

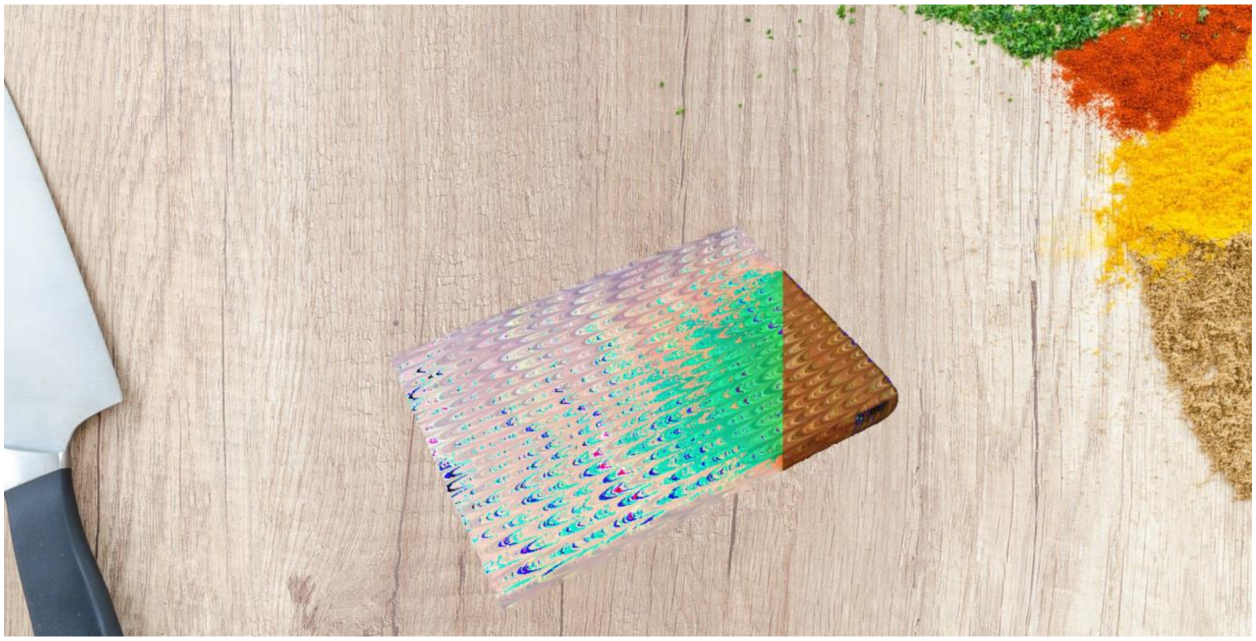
The following are images where Poisson Blending was applied:



Banana1 on table

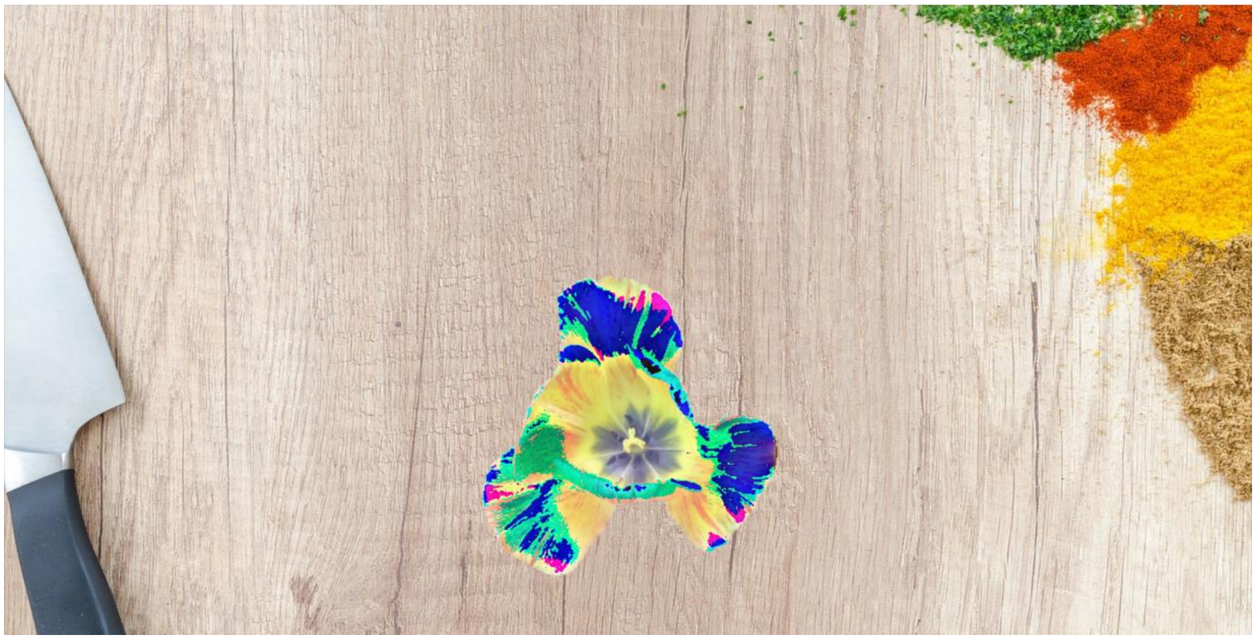


Banana2 on table



Book on table

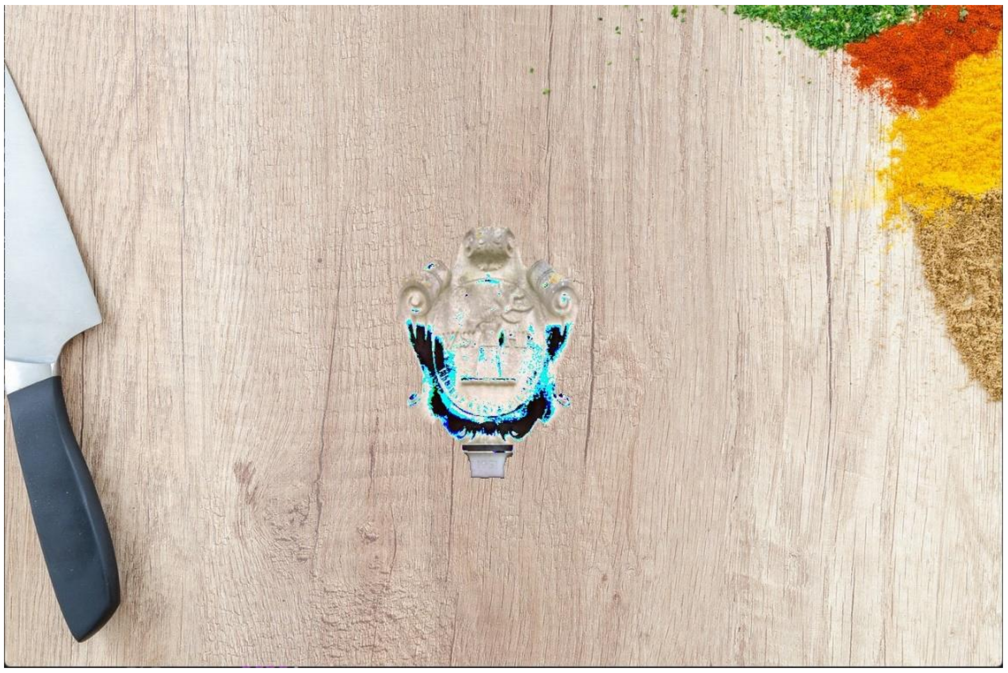
Bush on wall



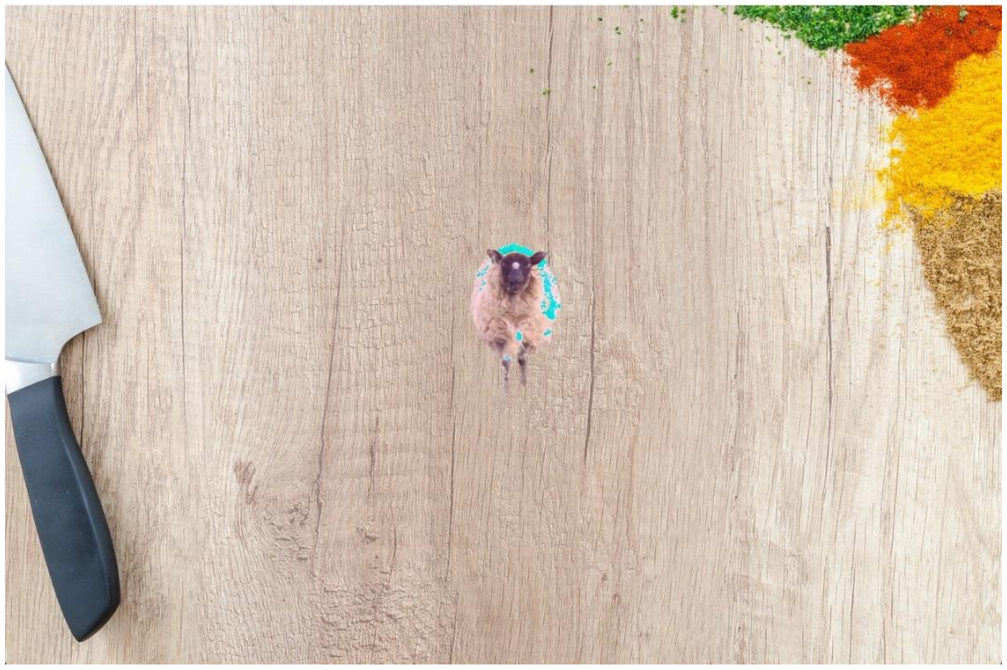
Flower on table

Moon on wall

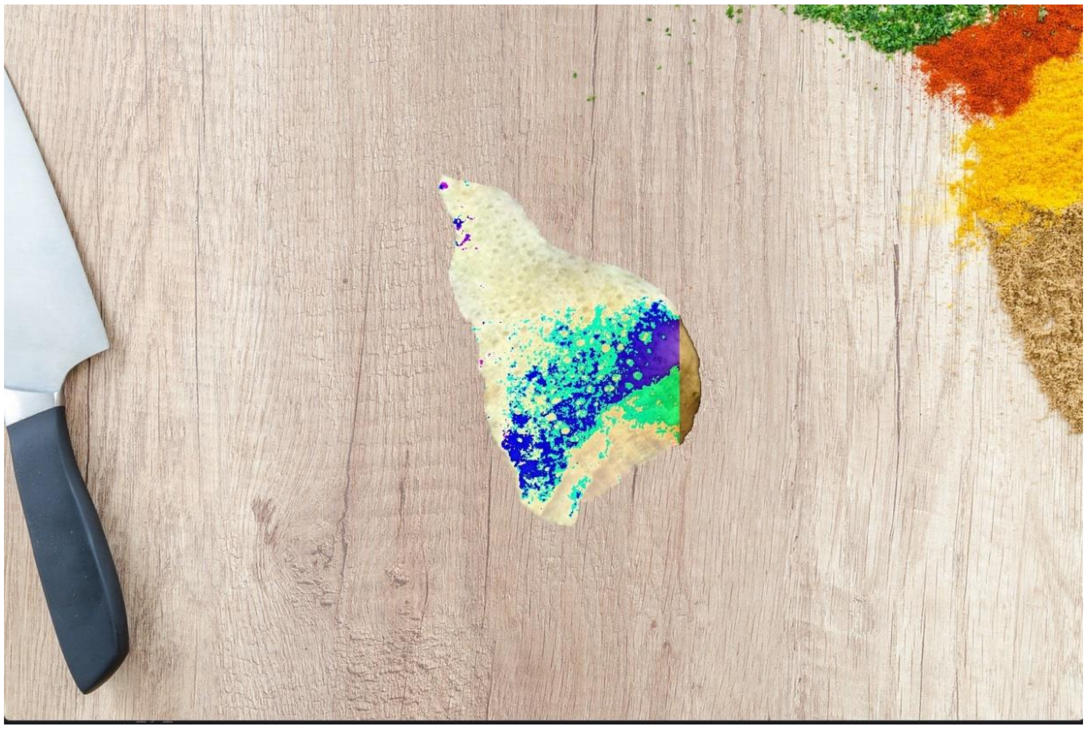
Grave on table



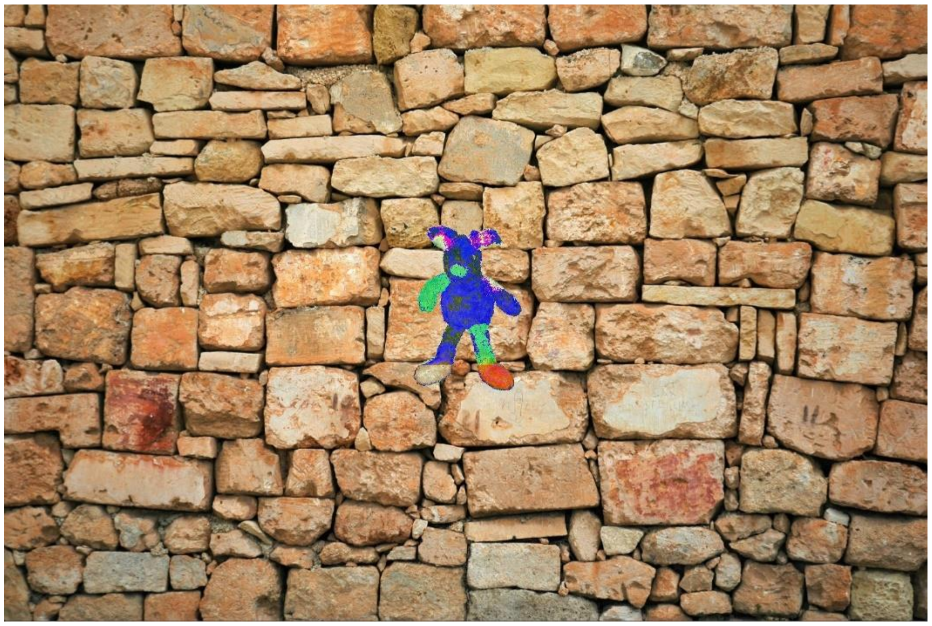
Memorial on table



Sheep on table

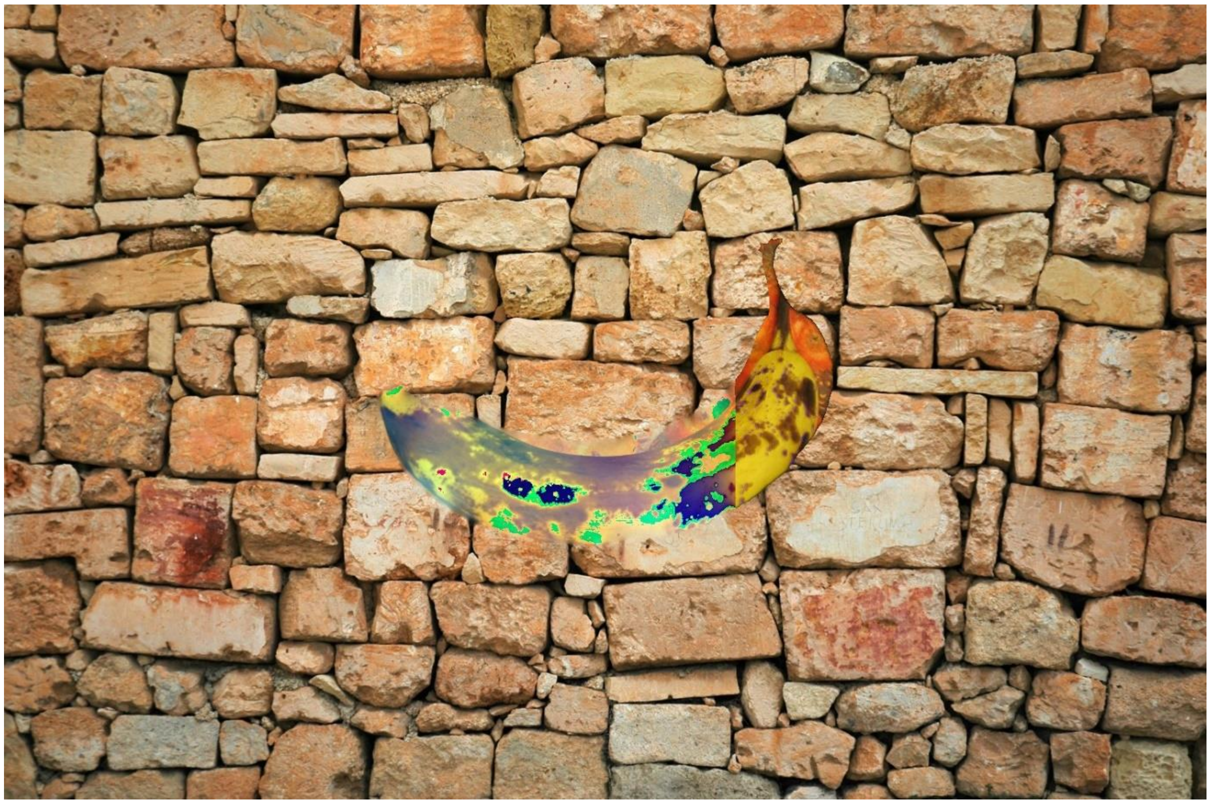


Stone2 on table

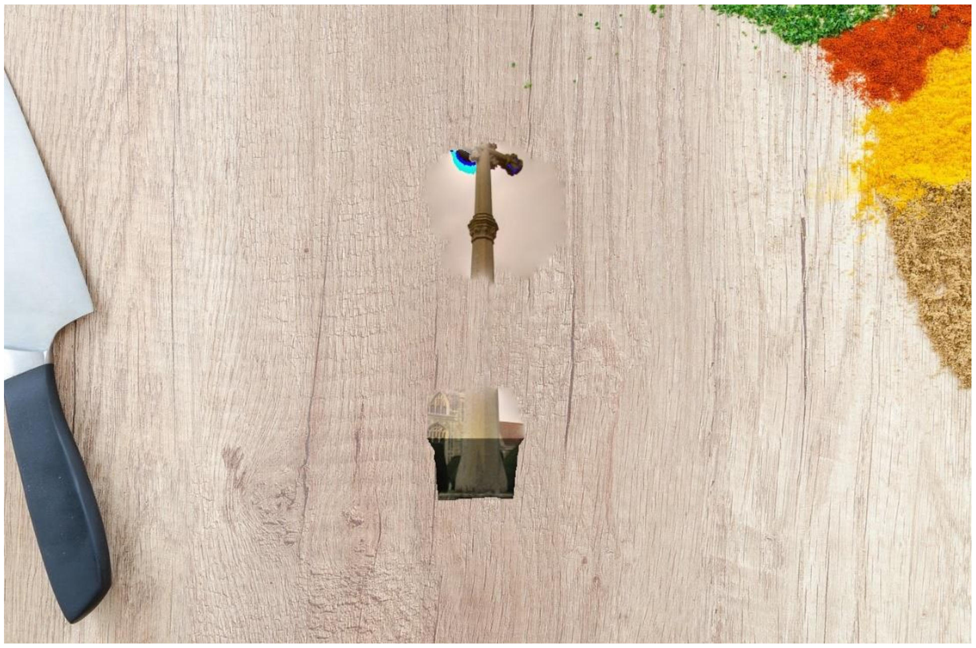


Teddy on Wall

The following 2 images have an untight mask:



Banana2 with the banana1 mask



Cross with bush mark

In general, we can see that most pictures were sliced well but were not colored well.

In the case of untight masks, we see that we got cut objects and some kind of a strange background that came with the objects from the source image.