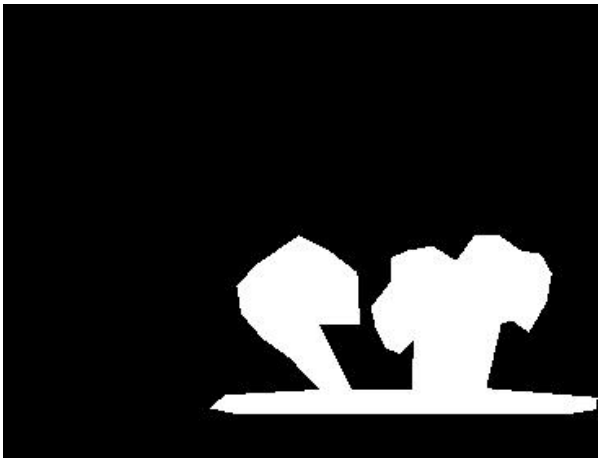
Object Removal by Exemplar-Based Inpainting

Implemented by Eli Bosworth (eboswort)

May, 2010

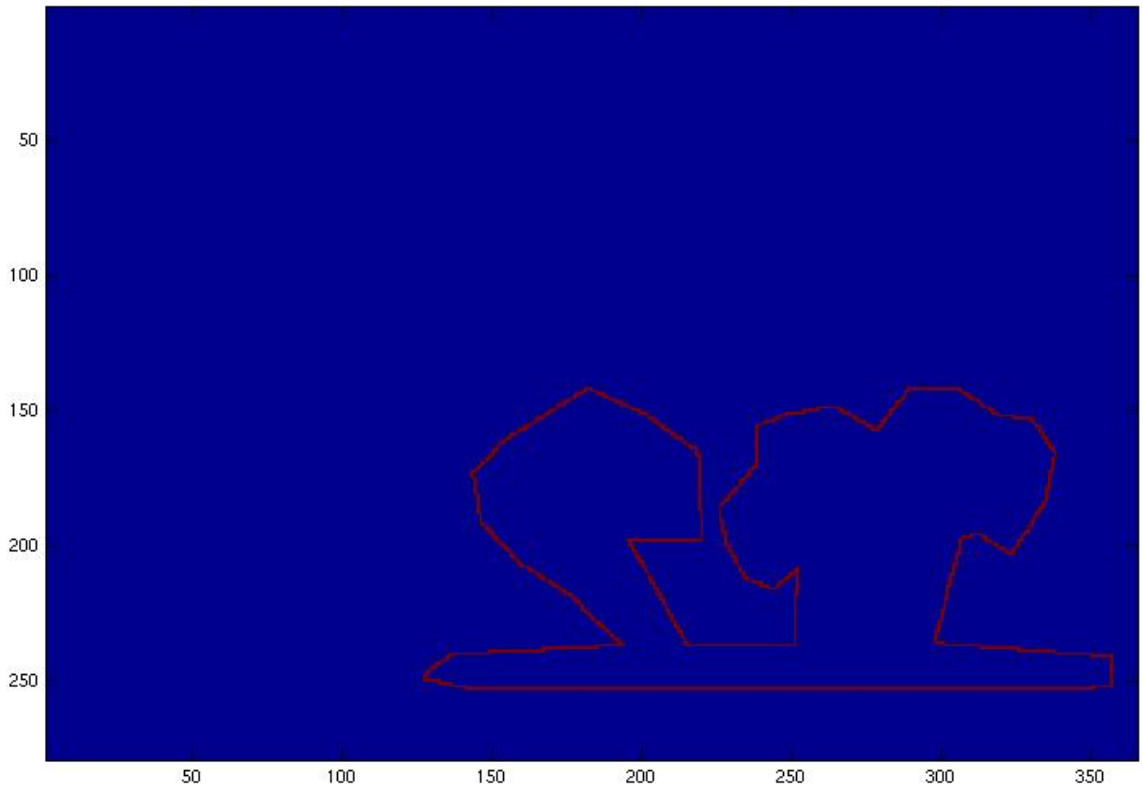
For my final project I implemented the algorithm described in Object Removal by Exemplar-Based Inpainting, by Criminsi et al. The idea behind the paper is to combine the advantages of two methods of filling regions of images, texture synthesis and inpainting. Texture synthesis copies pieces of texture from the other parts of the image and inpainting follows structural components of the image into the filling area. The advantage of texture synthesis is that the detail of the texture in the image is maintained. The advantage of inpainting is that the structure of the image is maintained. By using the structure of the image to decide where to place copied pieces of texture one can make sure to maintain the structure of the image while filling the unknown area with detailed texture.

The steps in detail for removing this desert island:

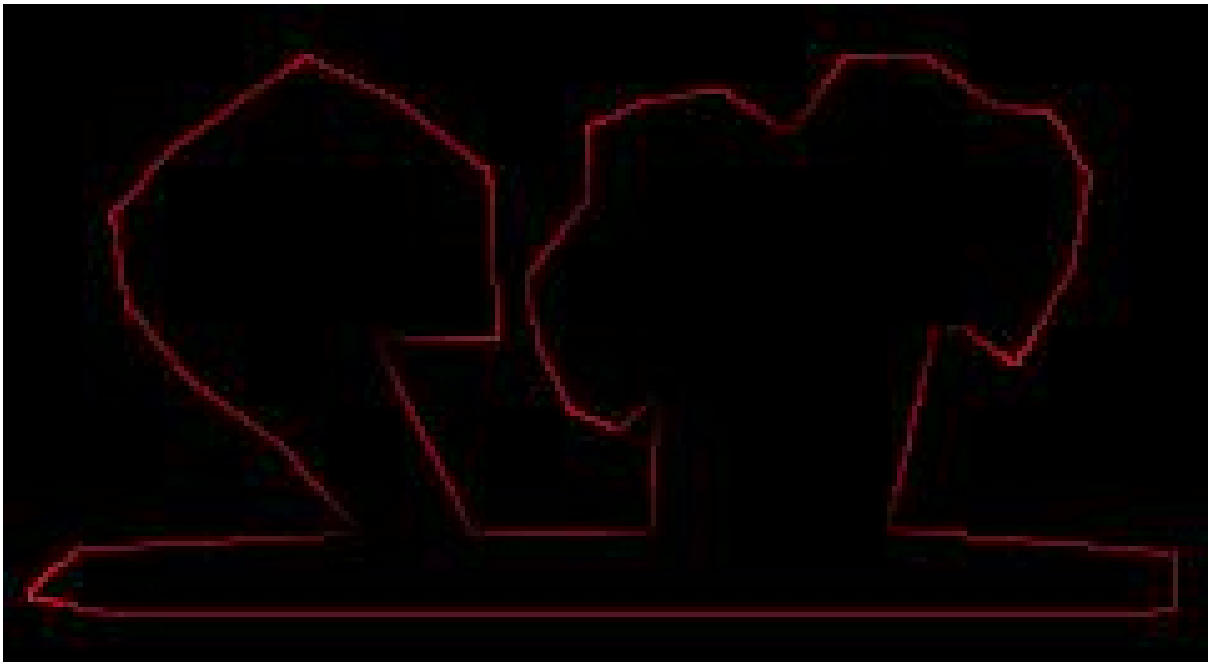
Find the border of the unknown area(s)

Simply extract the unknown area from an expanded version of itself. In order to avoid partial patches, the border is kept half the patch size away from the edge of the image



Find the confidence factor for each possible patch

Patches can be placed anywhere along the border. The confidence for a patch is the sum of the confidence values for each pixel it covers. At the start any pixel is the unknown area has a confidence value of 0 and all the pixels outside of that area have confidence of 1. Later those confidence values will be updated for areas that have been filled in



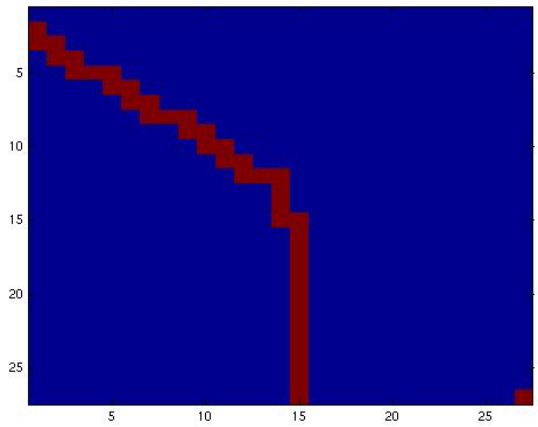
Find the structural factor for each possible patch

The goal here is to assign a value that represents where edges from the image intersect with the unknown area. First I convolved with a sobel filter

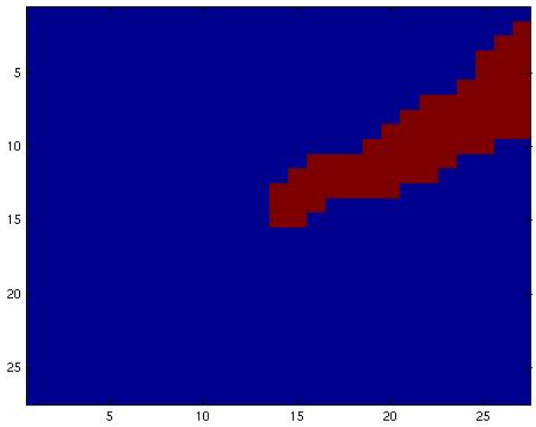
Then I remove values below a threshold and erode the remaining edges to hopefully remove texture and isolate structural edges within the image. Obviously I also remove edges that are inside the unknown area

Now for each patch a value that represents the angle between the incoming edge and the border of the unknown area must be produced. This is a hard problem becuase often the edges produced by the steps above are more complicated than simple lines. I used the matlab function polyfit to estimate the slope of both the border and the incoming edge. Here is an example result:

Polyfit estimates a slope of around -1.1 for this border



Polyfit estimates a slope of around 5/8 for this incoming edge



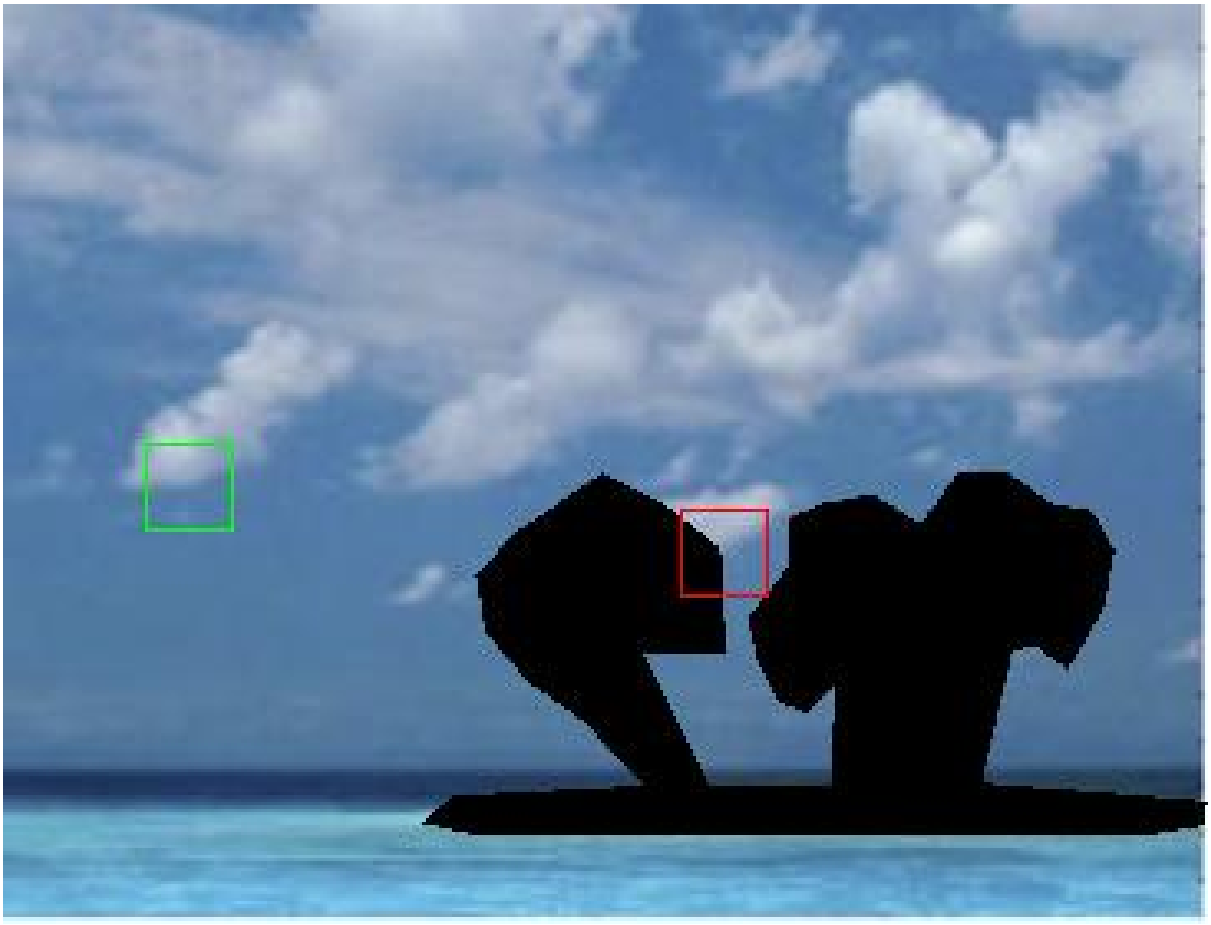
This is obviously not a very exact method of figuring out the angle between an incoming edge and the border, but it does pretty well. It can be thrown off by patches with more than one incoming edge, but usually in those cases it doesn't end up being a big problem since areas with some sort of incoming edge(s) still get done before areas without any incoming edges.

Pick the patch to fill

The patch priorities are created by simply multiplying the confidence and structural factors for every patch. The patch with the highest value is selected to be filled next

Pick the best match

Every patch in the image, outside the unknown area is considered as the possible best match for the selected patch. The patches are compared by sum of squared differences with the difference at each pixel weighted by its confidence value.

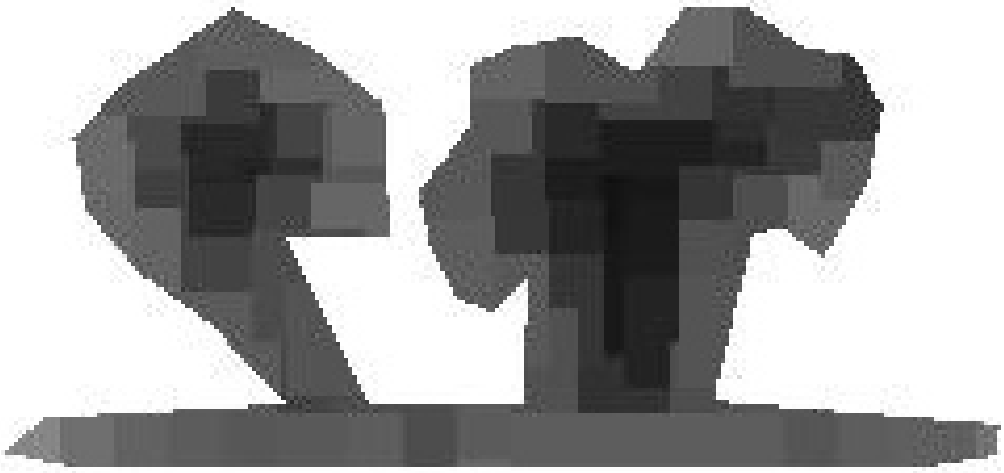


Add the patch to the image

The selected patch is added to the image using a poisson blend

Update the confidence Values

The area that was previously unknown now needs to be updated to have a confidence value greater than zero. The average confidence underneath the patch is assigned to the pixels that have low confidence values. This way confidence deteriorates as you move further into the unknown area and patches that overlapped more with confident areas reflect that their values are more trustworthy. Here is what the confidence matrix looks like after the process has finished:



Repeat

This process is repeated until all the pixels have positive confidence values

Notes and other stuff

It doesn't work for just any image

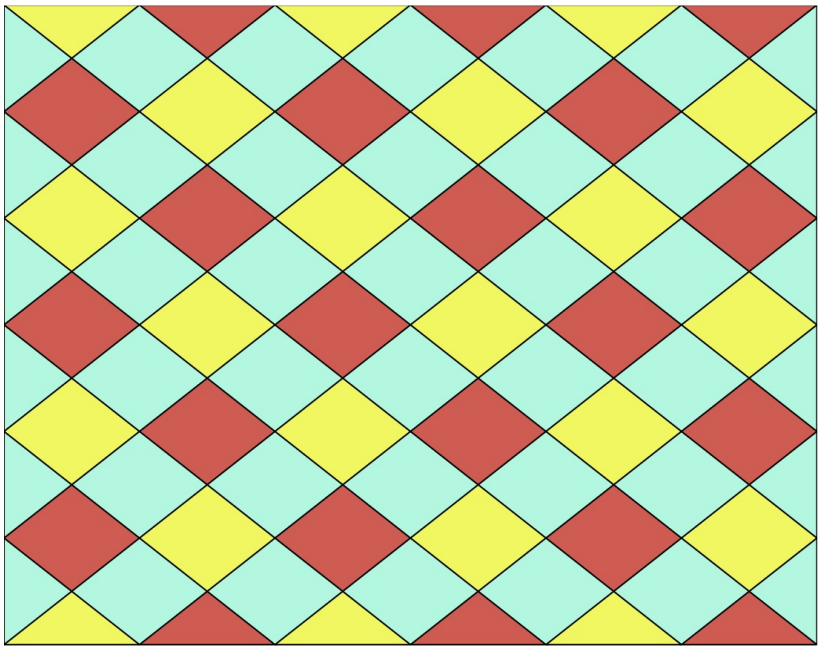
The first images I tried were from the paper and patterns that I had made. Those mostly worked very well, but finding images on google images that this algorithm can do a reasonable job on is not that easy. You really need a consitent background and it doesn't do so well with edges that intersect in the unknown area. It follows both edges in separately, but usually one makes it to the intersection first and doesn't allow for the other edge to meet up with it.

Patch size matters

Patch size matters a whole lot and it's not always easy to predict what patch size will be most effective. This is what the island removal looks like with a smaller patch size:

Artificial patterns are easy

This might be obvious but this algorithm does great with computer generated images. The clean edges and exact color equalities are very helpful. I tested in on images like this and it could reproduce them perfectly:



Kanizsa Triangle Video

Here is a video of the kaniza triangle getting filled in.

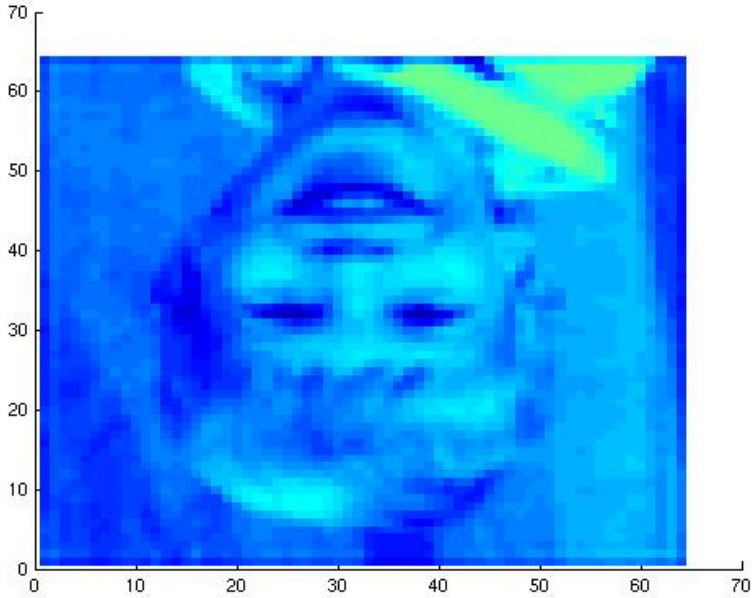
Click here

Reflecting patches

When experimenting with different images, I noticed that there were often situations where no patch really matched in it's natural orientation, but a patch could be found if a reflection of the image was considered. So I added code to check for patches in the image reflected horizontally, vertically and both. I quickly noticed that there are too many parts of a scene that have a top and bottom to reflect the image vertically, but I found that if only a horizontal reflection of the image was considered than it could improve the results.

Ghost story

I was sitting alone in my room late at night and matlab showed me this. I was just typing normally. I'm not sure what this means, but it was really scary. If I see this kid on a milkbox I am going to freak out



Some Results