Analysis of Texture Optimization for Example-based Synthesis

Uddeshya, Devansh, Bhavya

ABSTRACT

This is our implementation of the research work by *Vivek Kwatra et.al* on Texture Optimization for Example-based Synthesis. The Algorithm presented here defines *energy* of images and and uses *Iteratively reweighted least squares* to perform *EM optimization*.

What is the objective?

Given a sample (usually small) image of a texture, generate a larger image replicating the texture in the given image. One may be tempted to think that resizing the sample image or copy pasting the sample image might work. But both the methods have problems. For example copy-pasting the sample image will have discontinuities at the boundaries, resizing will degrade the quality and texture obtained will not look the same as the final image has larger elements than before.
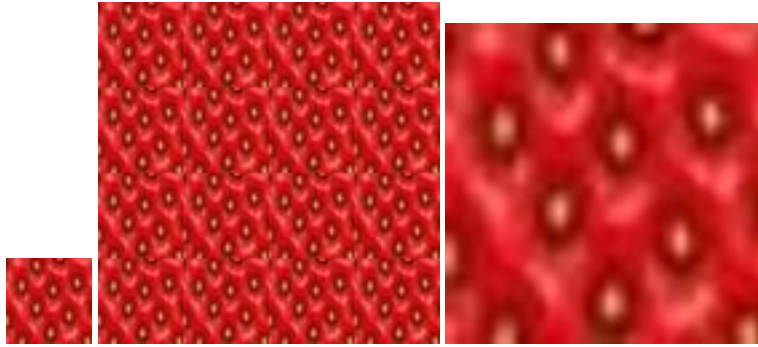

Figure 1. sample input texture, repeated and zoomed

What does the Algorithm do?

Given a patch of an image and a reference image, energy is defined for the given patch with respect to reference image as *sum of squared difference between the patch and its nearest neighbor in the reference image*. The energy of the image with respect to reference image is defined as the *sum of energies of patches of the image of some fixed dimension running with some fixed step size*.

The energy of the whole image act as the objective function and it is this function the algorithm is trying to optimize. This is just one of the simplest ways to define energy, one may add additional terms to this objective function, or take weighted some as per the need. Adding additional terms to the objective function however usually increase the cost and time of computation, hence in the interest of time we have avoided additional terms but it is worth noticing that additional terms can serve various purposes such as style transfer, flow synthesis, etc. In this implementation, instead of taking the square of the absolute difference, we used a power less than 1 to make the algorithm more robust (as pointed out in the paper).

We start of with some guess estimate of the image of the dimension that we are trying to obtain, we iteratively refine our estimate by minimizing the energy function (i.e finding nearest neighbor in sample image of various patch and updating our guess). Every round of refinement of our estimate will also change the nearest neighbor in the sample image for the next round. The refinement of our guess can be seen as the **E step of the EM optimization** and the updates in the nearest neighbor can be seen as the **M step**.

Results and explanations

Some of the images that we have generated are as follows. The synthesis happens in multiple levels (***multi-level***). We start synthesizing texture on a smaller resolution, the result at this resolution is resized to the higher resolution and used as the initial
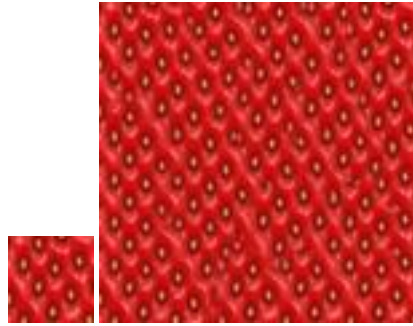
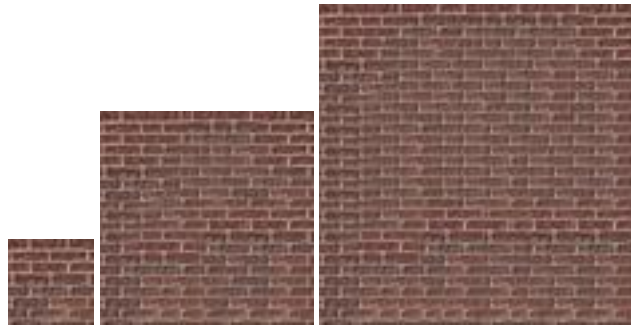**Figure 2.** sample input texture and the synthesized output

Figure 3. sample input texture, level 2 and level 3 outputs

estimate of the next level generation, this is continued until we reach the desired resolution.

The synthesis at each level also happens in multiple passes. In the initial pass the size of the patch is kept high, and as the number of passes increase the patch size decreases, this is done to smoothen out the artifacts and attain finer details after every pass. The energy plots as shown in the figure hit plateau in some of the initial passes at initial level, the explanation for the same is that the output dimensions of the images we are trying to synthesize are not so much bigger than the input sample texture, as a result we will have intermediate phases in the run where the pixels in the output images have a particular average values which doesn't changes for every iteration. Also, it may appear that the in every pass the energy goes to zero (in fact, those values are very small in comparison to other values as can be seen in matlab (or in other language) console, matlab renders them to zero)
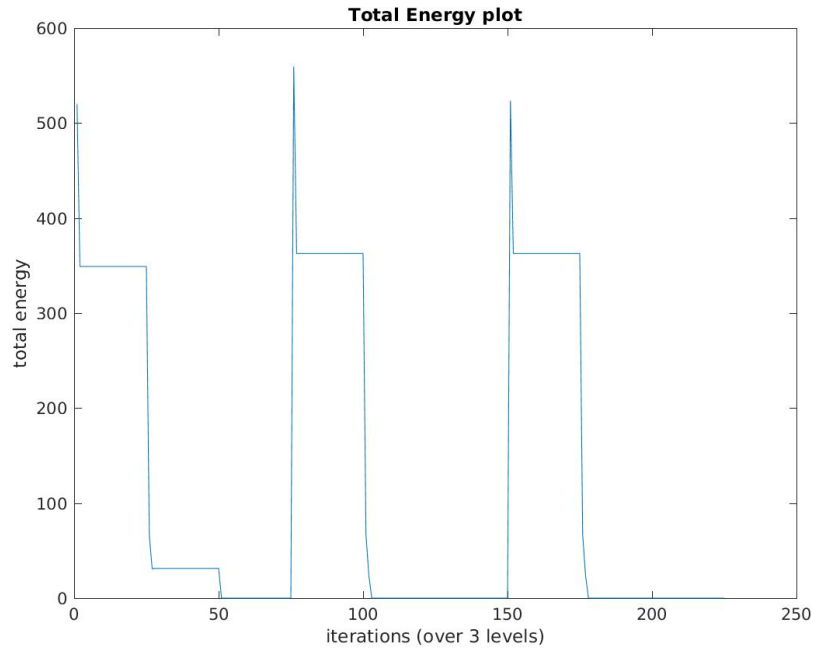
Figure 4. total energy plot for figure 2

Another observation which may seem counter intuitive at first is that if one plots the total energy, it's not really decreasing (over multiple levels, peak values may even be increasing), this so happens because as level increases so does the dimension of estimate image and hence number of patches also increases and total energy may go up or down or remain same. To account for this we have to perform some sort of normalization, here we have normalized every level by dividing with the estimate image dimension of that level.
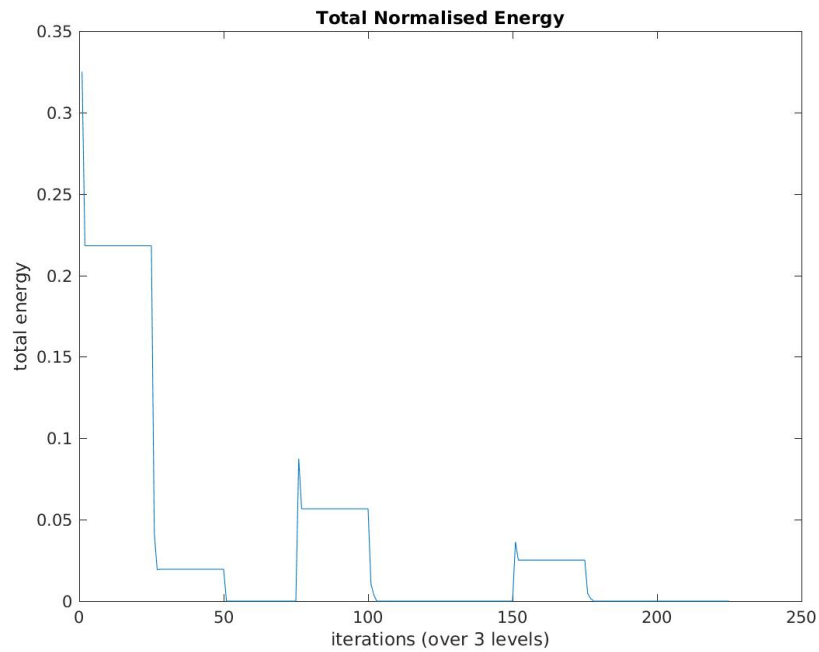

Figure 5. normalized total energy plot for figure 2

Limitations and Future works

The algorithm finds *nearest neighbors* of many patches in the sample texture image, which is relatively an expensive operation, as profiling of the code also reveals the bottleneck. We have used the vectorized code for nearest neighbor search, the main bottleneck being the fact that nearest neighbor search happens sequentially for each of the patches in a pass.

Profile Summary
Generated 22-Nov-2017 22:38:43 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
testSynt	1	1283.893 s	0.004 s	
synthesise	1	1283.864 s	0.091 s	
updateEX	225	1282.738 s	6.436 s	
NNPatchSearchX	154275	1276.302 s	1276.302 s	
imshow	9	0.981 s	0.239 s	
initSize	9	0.437 s	0.019 s	
movegui	9	0.376 s	0.265 s	
newplot	17	0.152 s	0.027 s	
findall	26	0.140 s	0.128 s	
basicImageDisplay	9	0.085 s	0.047 s	
newplot>ObserveFigureNextPlot	17	0.075 s	0.003 s	
clf	8	0.072 s	0.010 s	
newplot>ObserveAxesNextPlot	17	0.050 s	0.005 s	
graphics/private/clo	25	0.049 s	0.049 s	

Figure 6. profile of the implementation in one of the runs

For a given pass the nearest neighbor search of all the patches can be found independently of each other and hence a possible optimization is to translate the code into **C language** and use **pthread** library to create a thread for every row of patches. In the interest of time we did not implement that. The output generated here are of small dimension to save time.

Additional terms in the objective function can lead to solution of style transfer, but it would add to the above cost of computation hence we leave it for future work.

Contributions

Code development and the profiling was done by uddeshya. Formulation, verification and reporting done by all of us.

References

- Texture Optimization for Example-based Synthesis
- Style-Transfer via Texture-Synthesis