Gradient domain high dynamic compression

Jeffrey Mahou

January 2022

1 Challenges

- The first challenge that I encountered concerned the border conditions.
  The article specifies that we use Neumann's conditions to solve the Poisson equation, i.e. the gradient at the border is 0. Hence, when computing the differences in the borders, I considered that the assumed values at -1 and $length + 1$ were a duplication of the values at 0 and length on both axes.
  I did the same when computing the gradient H, the gradients H_k and the divergence of G.
  Then I realized that in order to be coherent, I should not apply the same conditions for the divergence of G since this is a second order operator (G is already a gradient). Therefor, for the divergence, I used a 0-padding of the gradient G.

- The second challenge was about the definition of α. In the article, they say they set α to "0.1 times the average gradient magnitude". Hence, at first, I computed the average of the original gradient magnitude and multiplied it by 0.1 as they said.
  However, I noticed that the maximum of the subgradients' magnitude was different for each subgradient. Indeed, since in the computation we divided by 2^{k+1} (k being the layer of the gaussian pyramid), the magnitude of the subgradient was decreasing by a factor of 2 approximately at every layer.
  In order to correct this, I made α dependent of each subgradient's magnitude, treating every layer with the same weight.
  The coefficient I used for α was about 10 times lower than in the article, so maybe we use different definitions. An exploration of the differences could be interesting.

- The third point that was missing from the article is about the way of generating the gaussian pyramid. I supposed that each layer was divided by a factor of 2 because the definition of the gradients has a 2^k factor, but I didn't know how they did the smoothing part.
  I applied a gaussian filter implemented by scipy which can be tuned : "sigma" is the standard deviation of the gaussian and "truncate" tells the gaussian filter where to stop in the discrete space.

- the last problem that I encountered was about the resolution of the Poisson equation.
  They mention a method called "full multigrid algorithm" but I couldn't find the article they referred to. I did implement a multigrid method though by first running the equation with the fine grid, then downsampling by a factor of 2, continue to resolve the equation and finally interpolate to the original size and running again simulations on a fine grid.

I didn't apply the Gauss-Seidel smoothing operations as I thought that would not be relevant when coding in python. Indeed, the Gauss-Seidel operations imply to go through all the pixels individually, thus having 2 "for" loops. In python, "for" loops are very costly so even if Gauss-Seidel might improve convergence, I think that the cost would have been too high.

2 Results

The dynamic range seems to be reduced considerably since for the belgium house, the range (maximum luminescence divided by minimum luminescence) was 640000 and was reduced to 5 with the compression. For the fog image, the DR was reduced from 13000 to 3.5. For the synagogue, it went from 400 to 2.5

Here I present the gradient map, the attenuation function and the modified gradient for the Belgium house.

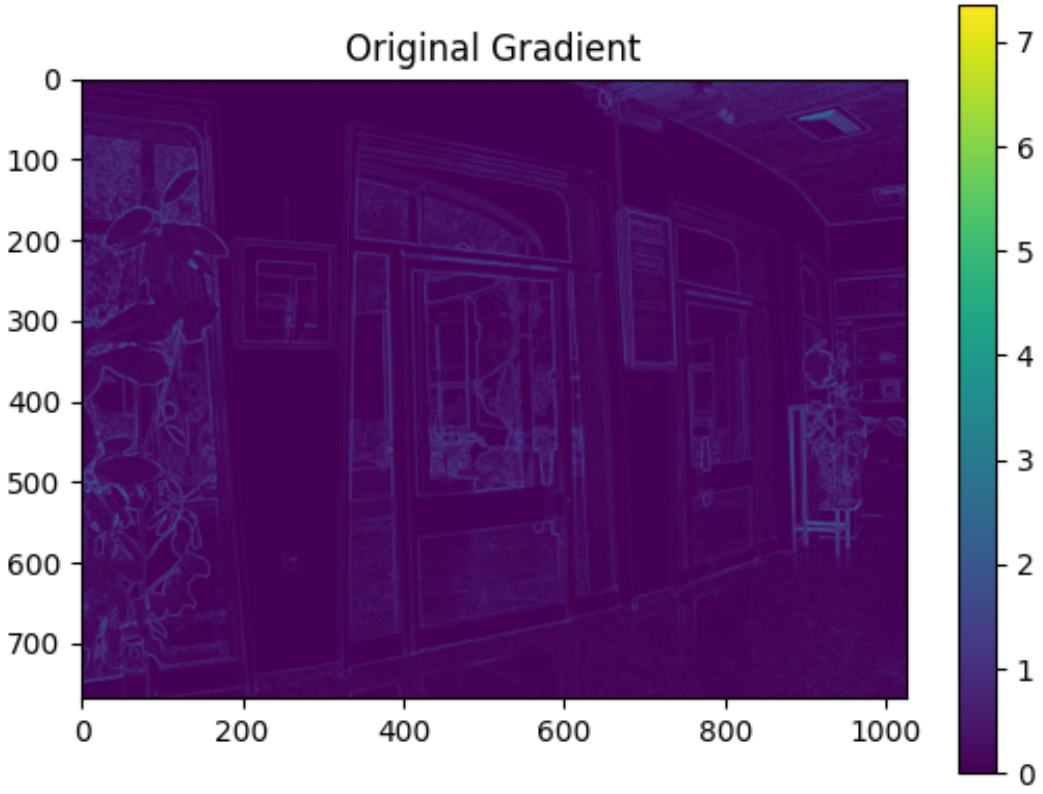

Figure 1: Original gradient of the belgium house

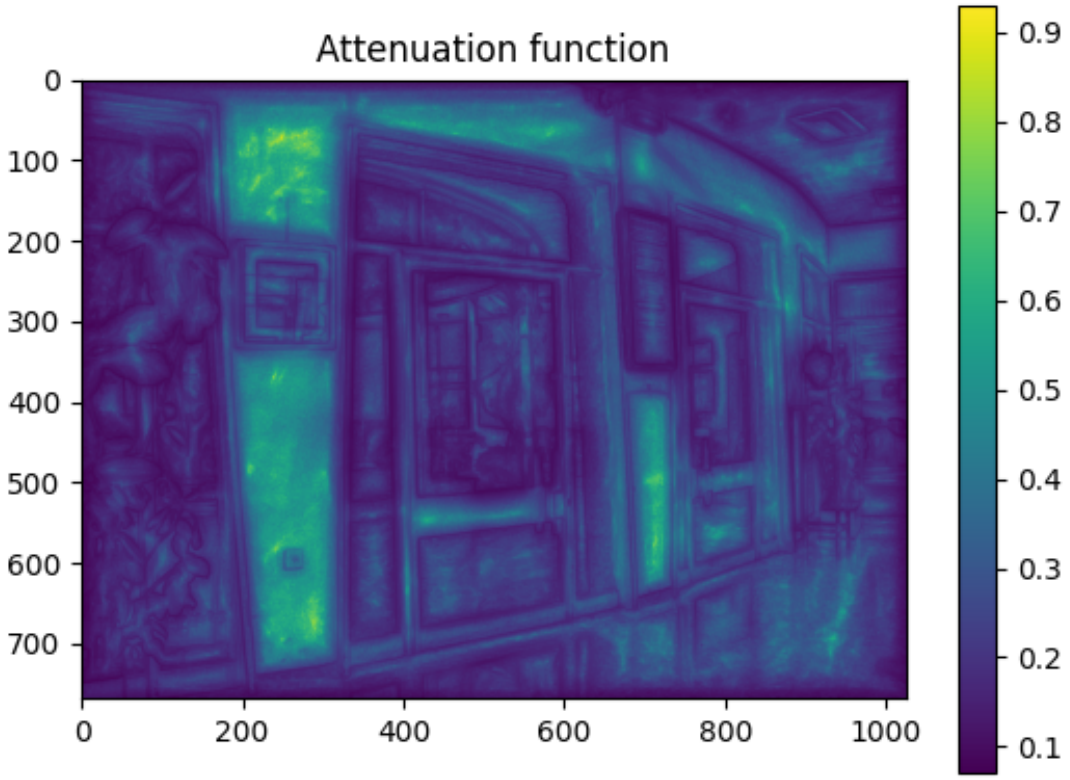

Figure 2: Attenuation function of the belgium house

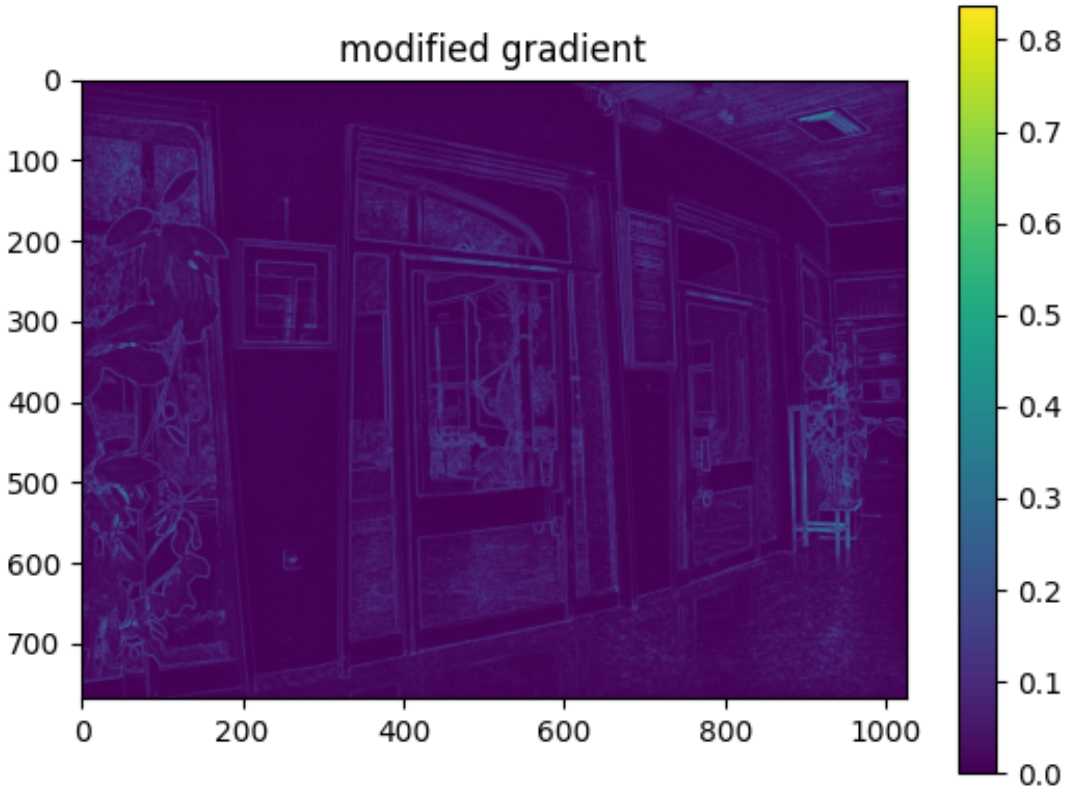

Figure 3: Modified gradient of the belgium house

We can see that the magnitude of the gradient is considerably reduced, going from a maximum of 7 to a maximum of 0.8

Now, here are some examples of the transformations for 3 different pictures.

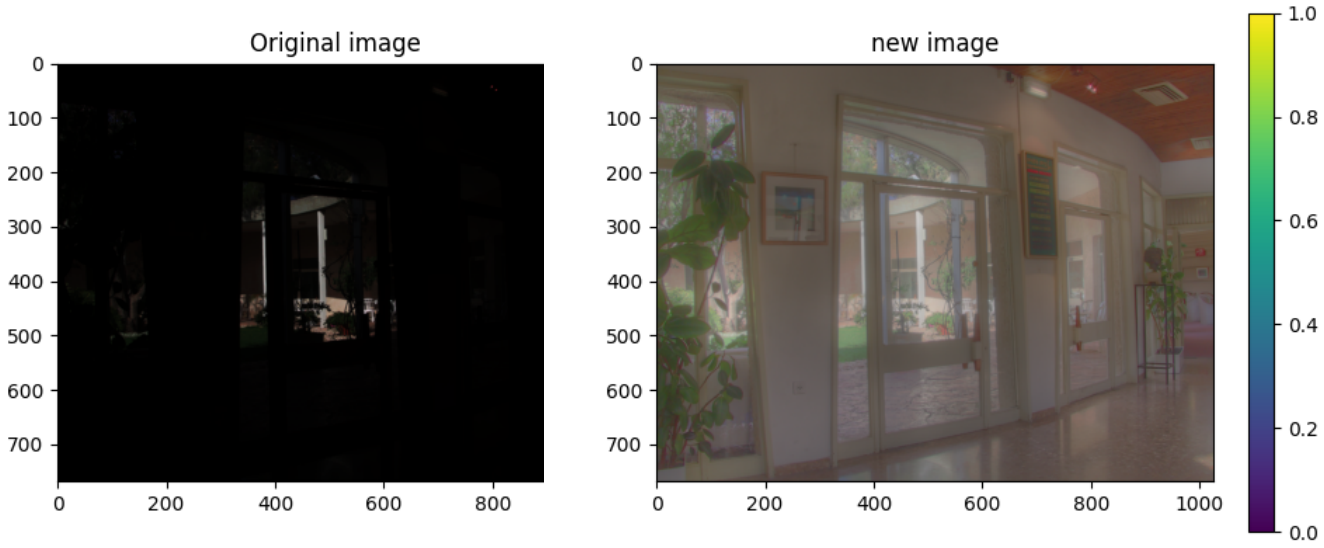

Figure 4: Belgium house before and after HDR compression

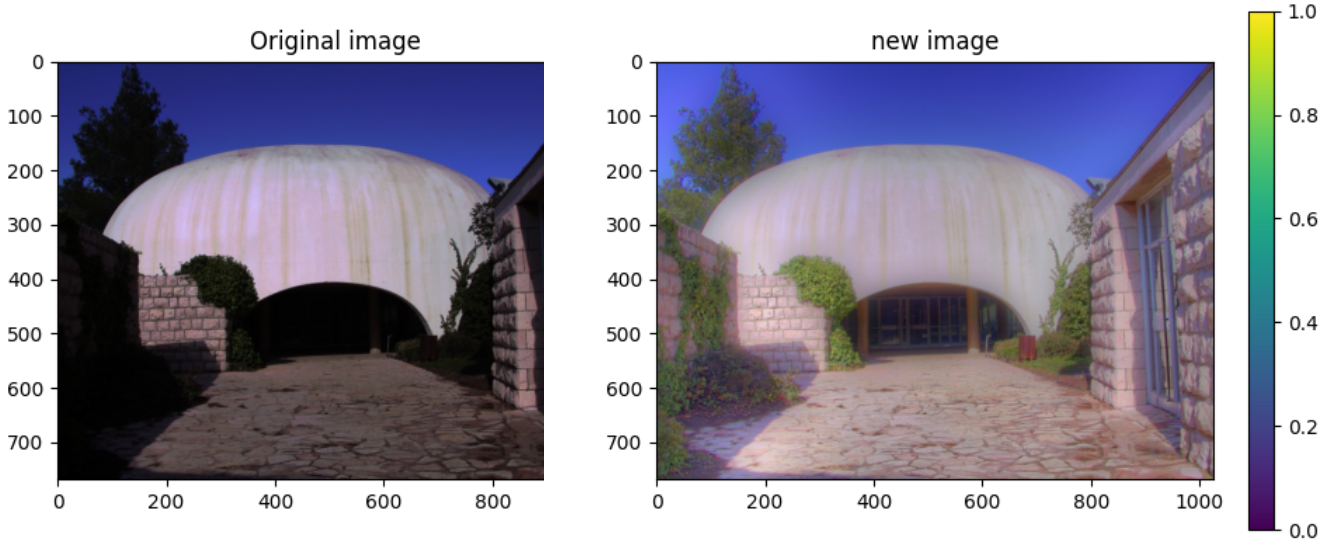

Figure 5: Synagogue before and after HDR compression

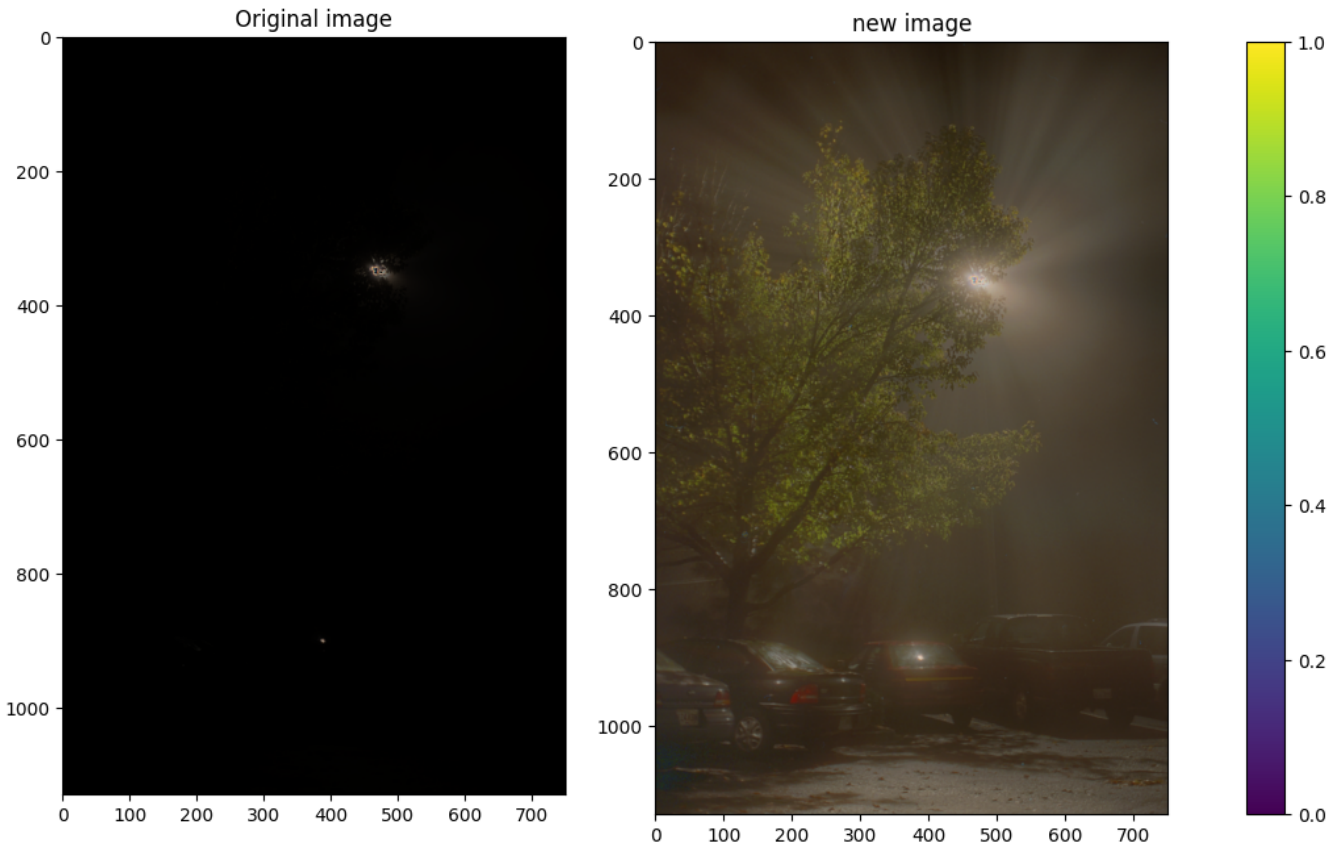

Figure 6: Fog Map before and after HDR compression

The results are quite satisfying as each of the parts of the image is lighted correctly and we can see the dark parts much better than in the original image. However, we notice a bit of an artificial looking of the image. Maybe the parameters can be tuned to have a more realistic display.

Finally, here are some results about the convergence of the algorithms on the belgium house image. It is hard to assess the exactness of the method as we don't have a ground truth reference objective.
We can nevertheless verify that the algorithm indeed converges to a certain point. In order to measure that, we compute the L2 norm between one iteration and the previous one and plot it in a log scale.

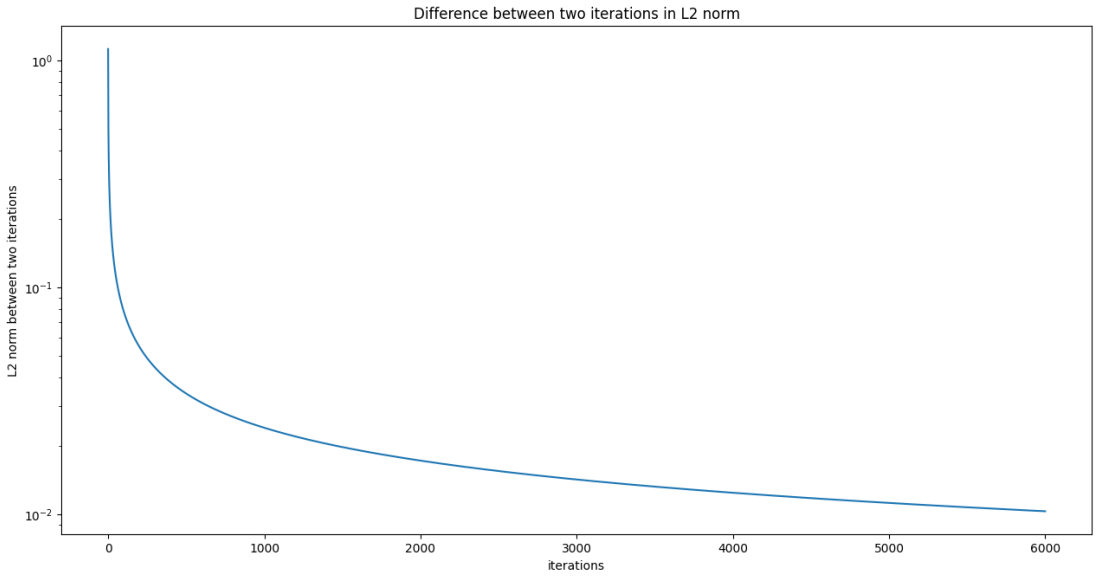

Figure 7: Convergence of the L2 loss for the 1 grid algorithm

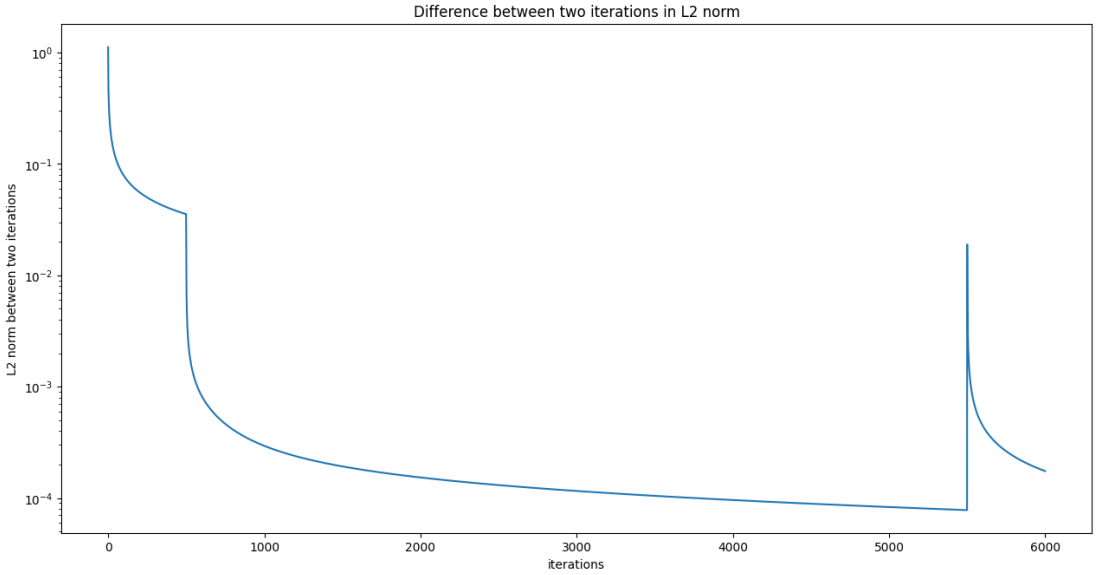

Figure 8: Convergence of the L2 norm for the multigrid algorithm

We can see that the 1 grid algorithm is much slower to converge as in 6000 iterations, it only reaches a loss of 10^{-2} while the multigrid algorithm reaches almost 10^{-4} so the gain is considerable. The jumps in the multigrid loss correspond to the switches between fine grid and coarse grid.

3 Further possibilities

- further tuning needs to be done on all the parameters mentioned as well as the number of iterations for the fine grid and the coarse grid.
- It would be interesting to apply Gauss-Seidel smoothing operations as described in the paper because it seems to be improving the convergence of the algorithm. However, I would suggest switching to C++ or using cython which is C++ coded in python because of the "for" loops that are very costly in python.
- Using more layers in the multigrid resolution (I only used 2) could also improve the results : We saw that using two layers improved by a lot the convergence, so we can imagine that adding more layers would again improve the results.
- Use a built-in solver for the Poisson equation, such as fenics. This was a bit hard to implement because fenics works with linux and I am on windows but it could be interested to see how a dedicated solver could treat this equation.
- In the article, they mention that a professional photographer can manually achieve a result of HDR compression by merging images of different exposure. It could be interesting to compute the difference (L2 norm for example) between the manual compression and the artificial one done by the algorithm.

4 Notes

The algorithm has a save function that enables you to save the images at different iterations and therefore see with your eyes the convergence of the algorithm.

To run the simulations, just run the python file, specifying the path to the image and the name of the hdr image (name should be in the form "belgium.hdr"). Further parameters can be given ; check it with "python image_HDR.py -h" command.