TP 1 : Texture Synthesis

Matheo Dumont p1602557
You can find my source code here on my repository.

The repository is the same for all the TPs.

Implementation

I have implemented the *Efros-Leung Agorithm* in C++ after having tried in octave. The main reason is the time it takes to compute the algorithm with octave whereas in C++ it is significantly faster.

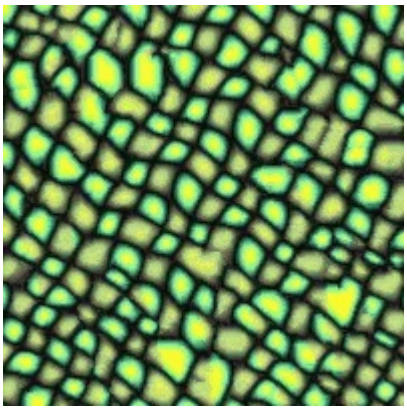
You can find the implementation at `src/tp1.cpp`. To compile it, being located at the root of the project:

```
mkdir build
cd build
cmake ..
```

Compile and run (in the build directory):

```
make -j$(nproc) && ./tp1
```

Here is the synthesis with the sample `data/synthese/text0.png`.
The size is (200, 200) with `n=9` and `epsilon=0.05`. It took 2 minutes and 17 sec.



Discussion

Tweaking the parameters is important to obtain images similar to te sample.

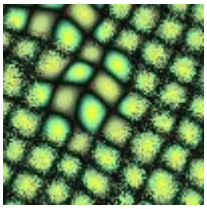
- `n` define the size of the patch that we use to compute the similarity between a portion of `I`, the synthesis image and `Ismp`, the sample we're trying to extend.
- `epsilon` define how similar a patch used to fill a pixel is to the closest patch of the portion of `I` we are looking.

With an appropriate `epsilon` and `n`, we get a result like the image above. Otherwise, with `n` too big, we compare big portion of images and this could lead to pixels changing abruptly, if the patterns on the sample are not big enough. With `n` to little, we loose the 'sense' of the image, logic of the pattern and it becomes chaos (and monstruous) ((200,200) `n=7` `eps=0.05`):

For `epsilon`, since i have normalize my distance between 0 and 1, 0 means always choosing the best fit and 1 is choosing completely randomly between all the patchs of `Ismp`. The closer we get to 1, the more noisy the image can be.

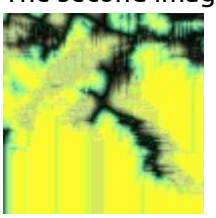
But those 2 parameters works together, even with a high `epsilon`, we can still get a plausible image because `n` allow to keep 'sense' to the image, but the inverse isn't true.

Also having both parameters high, or low gives bad results : The first image is `(100, 100)` with `n=21` and `epsilon=1`,



We can notice the first patch copy past, which is clean, and the noise all around.

The second images is the same but with `n=1` and `epsilon=0`,



Improvements

We could use the algorithm *Patch Match* to improve the research for closest patch, but having a high `epsilon` would increase time since we will have to compute more distances.