Interpolation of an image (multi-threaded)

1. Source image enlargement:

* When interpolate pixels around the border, the area outside the image may need to be sampled. These could be solved largely in two ways. But I would like to place some of my analysis for this.

1st Method : When sampling the area outside the image, copy the pixel from the border.

2nd Method : First, pad some colors around the border first. (plus, the source image could be enlarged so the indexing may be more straightforward.) Then just interpolate over the padded area.

I chose the 2nd method for the reason stated below.

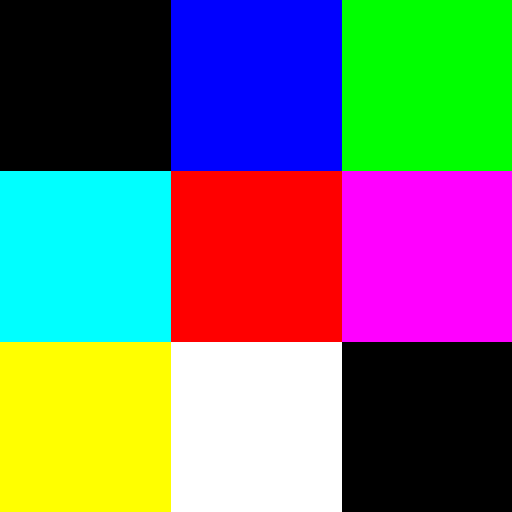
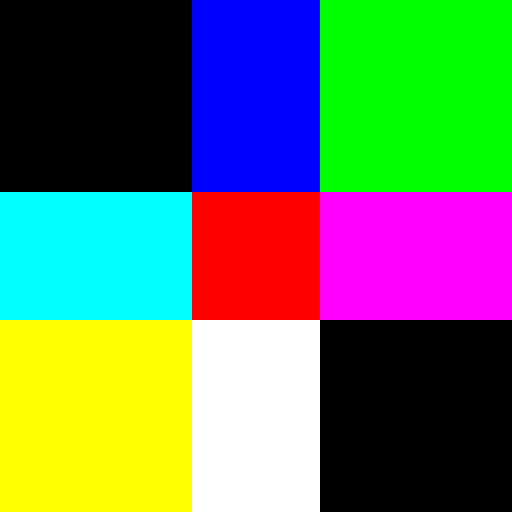
* Although the 2nd method adds extra computation, it removes a few if statements that is used over the destination image, which is usually larger.
* **(Theoretical) performance comparison** between different design choices.

1st method adds the computation :

2nd method adds the computation :

Typically, a destination image is larger than the source image. Therefore,

Thus, I chose the 2nd method.

**Figure 1 – Source image (left). Expanded source image (middle). Filtered image (right).**

1. Thread management:

* Std::thread – I used STL library thread.
* I implemented asynchronous join of threads myself.
  + Because STL thread lacks asynchronous join of threads, I coded it in ThreadManager class myself.
    - You may notice one global variable for that. I almost never use it especially in a threaded environment. However, implementing asynchronous joining may involve communication between threads. One global might not matter for this scope of projects. So I went with that.