Noise simulation

Gaëtan Lehmann

March 19, 2010

Abstract

Several kind of noise can be found in real images, mostly depending on the modality of acquisition. It is often useful to be able to simulate that noise, for example to test the behavior of an algorithm in the presence of a known amount of noise.

This contribution provides the filters to generate four kind of noise – additive gaussian, shot, speckle and salt and pepper – as well as a PSNR calculator.

Contents

1 Noise types

1.1 Additive gaussian noise

This is the most frequent kind of noise. It can be modeled as:

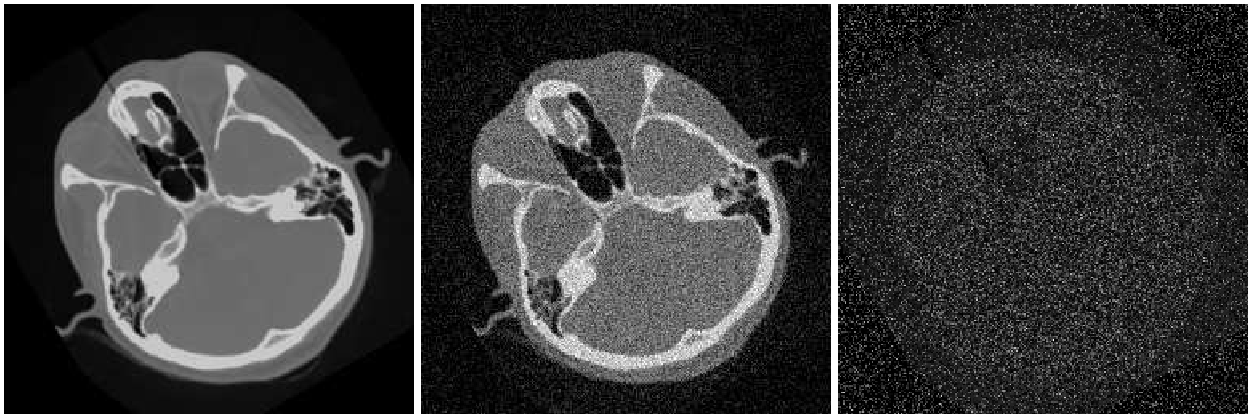
$$I = I_0 + N \tag{1}$$

where I is the observed image, I_0 is the non-noisy image and N is a normally distributed random variable of mean μ and variance σ^2. The noise is independant of the pixel intensities.

$$N \sim \mathcal{N}(\mu, \sigma^2) \tag{2}$$

μ is generally 0.

Additive gaussian noise can be simulated with `itk::AdditiveGaussianNoiseImageFilter`. The mean can be specified with `SetMean()` and the standard deviation with `SetStandardDeviation()`. The mean defaults to 0 and the standard deviation to 1.



(a) Input image (b) Noisy image (c) Generated noise

Figure 1: (a) the input image. (b) image altered with additive gaussian noise with $\mu = 0$ and $\sigma = 22.8$. (c) the generated noise extracted by computing the absolute difference between (a) and (b). Note that the noise is *almost* independant of the pixel intensities – the dark zones show less noise because of the clipping of the negative values applied to the pixel intensities during the simulation. The command line used was *./gauss ../images/cthead1.tif gauss.png 22.8.*

1.2 Shot noise

Shot noise, also called Poisson noise or photon noise can be modeled as:

$$I = N(I_0) \tag{3}$$

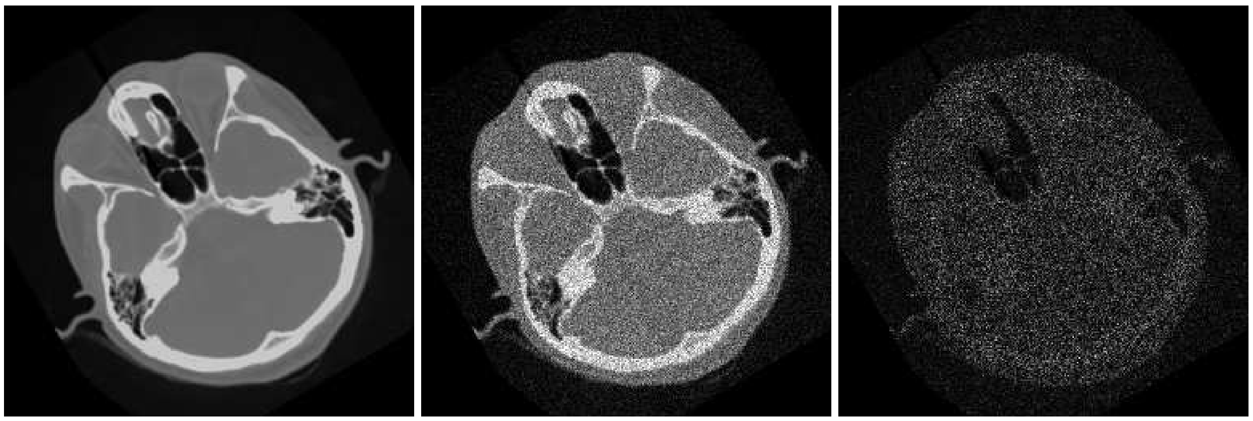
where $N(I_0)$ is a poisson distributed random variable of mean I_0. The noise is thus dependant on the pixel intensities in the image.

Shot noise can be simulated with `itk::ShotNoiseImageFilter`. The intensities in the image can be scaled by a user provided value to map the pixel value to the actual number of photon. The scaling can be seen as the inverse of the gain used during the acquisition. The noisy signal is then scaled back to its input intensity range.

$$I = \frac{N(I_0 \times s)}{s} \tag{4}$$

where s is the scale factor.

The scale factor can be set with `SetScale()`.



(a) Input image (b) Noisy image (c) Generated noise

Figure 2: (a) the input image. (b) image altered with shot noise with $s = 0.15$. (c) the generated noise extracted by computing the absolute difference between (a) and (b). Note that the noise is dependant of the pixel intensities – a strong signal leads to a strong noise. The command line used was *./shot ../images/ct-head1.tif shot.png 0.15*.

The poisson distributed variable is computed by using the code

Algorithm 1.1: POISSONDISTRIBUTEDVARIABLE(λ)

```
$k \leftarrow 0$
$p \leftarrow 1$
repeat
  $k \leftarrow k + 1$
  $p \leftarrow p * \mathrm{U}()$
until $p > e^{-\lambda}$
return $(k)$
```

where $U()$ provides a uniformly distributed random variable in the interval $[0,1]$.

This algorithm is very inefficient for large value of λ though. Fortunately, the poisson distribution can be accurately approximated by a Gaussian distribution λ of mean and variance λ when λ is large enough. This leads to this faster algorithm:

Algorithm 1.2: APPROXIMATEDPOISSONDISTRIBUTEDVARIABLE(λ)

```
if $\lambda \leq 50$
  then return (POISSONDISTRIBUTEDVARIABLE($\lambda$))
  else return $(\lambda + \sqrt{\lambda} \times \mathrm{N}())$
```

where $N()$ produce a normally distribution variable of mean 0 and variance 1.

1.3 Speckle noise

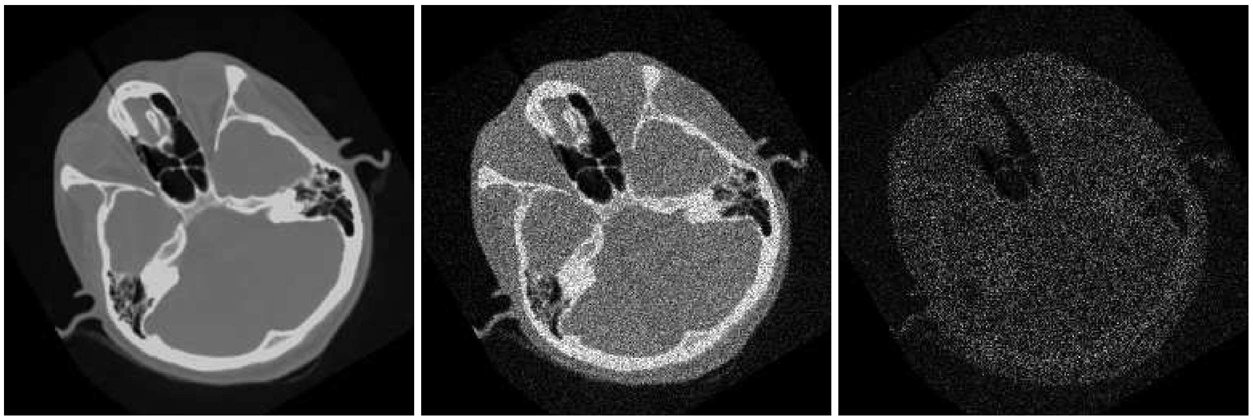
Speckle noise is also called multiplicative noise. It can be modeled as:

$$I = I_0 * G \tag{5}$$

where G is a is a gamma distributed random variable of mean 1 and variance proportional to the noise level.

$$G \sim \Gamma(\frac{1}{\sigma^2}, \sigma^2) \tag{6}$$

Speckle noise can be simulated with `itk::SpeckleNoiseImageFilter`. The standard deviation of the noise can be set with `SetStandardDeviation()` and defaults to 1.



(a) Input image (b) Noisy image (c) Generated noise

Figure 3: (a) the input image. (b) image altered with speckle noise with $\sigma = 0.24$. (c) the generated noise extracted by computing the absolute difference between (a) and (b). Note that the noise is dependant of the pixel intensities – a strong signal leads to a strong noise. The command line used was *./speckle ../images/cthead1.tif speckle.png 0.24*.

The gamma distributed random variable is a bit more difficult to compute than what is done in the previous cases.

Algorithm 1.3: GAMMADISTRIBUTEDVARIABLE(k,θ)

```
$\delta \leftarrow k$
$v_0 \leftarrow \frac{e}{e+\delta}$
repeat
    $v_1 \leftarrow U(), v_2 \leftarrow U(), v_3 \leftarrow U()$
    if $v_1 \leq v_0$
        then
            $\xi \leftarrow v_2^{1/\delta}$
            $\nu \leftarrow v_3 \xi^{\delta-1}$
        else
            $\xi \leftarrow 1 - ln v_2$
            $\nu \leftarrow v_3 e^{-\xi}$
until $\nu > e^{-\xi}\xi^{\delta-1}$
return $(\theta\left(\xi - \sum_{i=1}^{[k]} ln U()\right))$
```

where $U()$ produce a uniformaly distributed variable on the lower open range $(0,1]$, $[k]$ is the integral value of k and k is the decimal value of k.

1.4 Salt and pepper noise

Salt and pepper noise is a special kind of impulse noise where the value of the noise is either the maximum possible value in the image or its minimum. It can be modeled as:

$$I = \begin{cases} M, & \text{if } U < p/2 \\ m, & \text{if } U > 1 - p/2 \\ I_0, & \text{if } p/2 \geq U \leq 1 - p/2 \end{cases} \tag{7}$$

where p is the probability of apparition of the noise, U is a uniformally distributed random variable on the range $[0,1]$, M is the greatest possible pixel value and m the smallest possible pixel value.

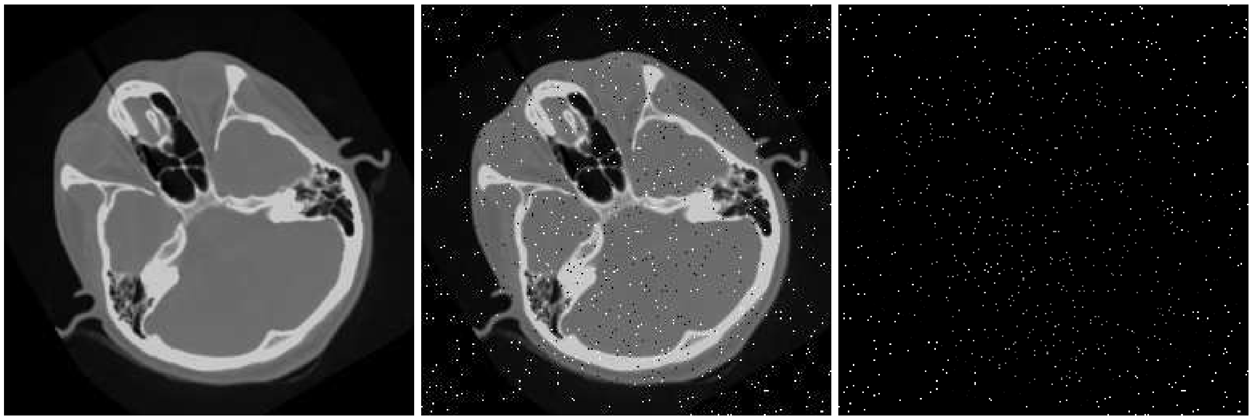
Salt and pepper noise can be simulated with `itk::SaltAndPepperNoiseImageFilter`. The probability of the noise can be set with `SetProbability()` and defaults to 0.01.

2 Peak signal-to-noise ratio

The peak signal-to-noise ratio (PSNR) is a measure of degradation of the image quality. It is computed as

$$PSNR = 10 \cdot log_{10}\left(\frac{M \cdot N}{\sum_{p \in D}(I_0[p] - I[p])^2}\right) \tag{8}$$

where D is the domain of definition of the image I_0 and I, N is the number of pixel in D, and M is the maximum possible value in I.



(a) Input image (b) Noisy image (c) Generated noise

Figure 4: (a) the input image. (b) image altered with salt and pepper noise with $p = 0.016$. (c) the generated noise extracted by computing the absolute difference between (a) and (b). Note that the noise is independant of the pixel intensities. The command line used was *./sp ../images/cthead1.tif sp.png 0.016*.

All the images degraded with noise in this article have a PSNR of 20.

3 Wrapping

All the new filters have been wrapped using WrapITK.

4 Development details

`itk::MersenneTwisterRandomVariateGenerator` has been modified to produce a thread safe implementation called `itk::ThreadSafeMersenneTwisterRandomVariateGenerator`. It should be possible to fix the non thread safety in the original class, but the implication on the testing framework and the backward compatibility are beyond the scope of this contribution.

The filters are *not* implemented as subclasses of `itk::UnaryFunctorImageFilter` because each thread must have its own random generator. They are implemented as subclasses of `itk::InPlaceImageFilter` to be able to run the noise addition in place. All the filters are multithreaded.

Due to the difficulty of testing a random behavior, no test is provided.

A development version is available in a darcs repository at `http://mima2.jouy.inra.fr/darcs/contrib-itk/noise`

5 Conclusion

6 Acknowledgments

Unlike most of my previous contributions, this work has *not* been supported by the french National Institute for Agricultural Research (INRA) – my employer. I'd like to thanks the INRA anyway, and more specifically the MIMA2 facility, to have provided the computer ressources used for this work.

7 References

Most of the informations used in this article are coming from wikipedia.

http://en.wikipedia.org/wiki/Additive_white_Gaussian_noise

http://en.wikipedia.org/wiki/Shot_noise

http://en.wikipedia.org/wiki/Poisson_distribution

http://en.wikipedia.org/wiki/Speckle_noise

http://en.wikipedia.org/wiki/Gamma_distribution#Generating_gamma-distributed_random_variables

http://www.ceremade.dauphine.fr/~peyre/numerical-tour/tours/denoising_data_dependent

http://en.wikipedia.org/wiki/PSNR