IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2017–07–18.
Submitted on 2016–10–20, accepted on 2017–03–29.
ISSN 2105–1232 © 2017 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2017.192

Realistic Film Grain Rendering

Alasdair Newson[1], Noura Faraj[2], Julie Delon[3], Bruno Galerne[4]

[1] Télécom ParisTech, France (anewson@telecom-paristech.fr)
[2] MAP5, Université Paris Descartes, France (noura.faraj@parisdescartes.fr)
[3] MAP5, Université Paris Descartes, France (julie.delon@parisdescartes.fr)
[4] MAP5, Université Paris Descartes, France (bruno.galerne@parisdescartes.fr)

Communicated by Mauricio Delbracio *Demo edited by* Noura Faraj and Nelson Monzón

Abstract

Film grain is the unique texture which results from the silver halide based analog photographic process. Film emulsions are made up of microscopic photo-sensitive silver grains, and the fluctuating density of these grains leads to what is known as film grain. This texture is valued by photographers and film directors for its artistic value. We present two implementations of a film grain rendering algorithm based on a physically realistic film grain model. The rendering algorithm uses a Monte Carlo simulation to determine the value of each output rendered pixel. A significant advantage of using this model is that the images can be rendered at any resolution, so that arbitrary zoom factors are possible, even to the point where the individual grains can be observed. We provide a method to choose the best implementation automatically, with respect to execution time.

Source Code

The C++ code for this work, as well as an online demo, are available from the web page of the article[1].

Keywords: film grain; texture synthesis; stochastic geometry

1 Introduction

Film grain is an essential phenomenon which contributes to the visual quality of analog images. Many prominent film directors still shoot their movies in analog film in order to get this unique look, and amateur photographers often add this film grain to their digital images a posteriori to attenuate the undesirable "digital" visual aspect. Therefore, a realistic film grain synthesis algorithm is of clear importance for professionals and amateurs alike. Newson et al. [13] proposed a film grain model based on the physical photographic process. This model employs a Boolean model from the stochastic geometry literature to model the film grain. The advantage of the model is that it is defined

[1] https://doi.org/10.5201/ipol.2017.192

in a continuous domain and the grain can therefore be rendered, via a Monte Carlo simulation, at any desired resolution.

We describe the implementation of this algorithm, for which we provide precise details. We describe two distinct implementations of the algorithm which perform differently in terms of execution time depending upon the input parameters. We perform a theoretical and empirical analysis of the time complexities of each implementation and propose an empirical solution to choose the fastest one automatically.

2 The Photographic Process and Previous Work

A photographic film emulsion is made of gelatin in which photosensitive silver halide crystals are suspended. The photographic process is carried out in two steps: film grain sensitization and development. Sensitization refers to exposing the silver halide crystals to incoming photons for a certain amount of time. When a photon hits a crystal it can, via a reduction reaction, create a tiny amount of solid silver on the crystal. The crystals which undergo this change are made "developable", that is to say they may be turned completely solid, and thus opaque, by a chemical developer. These two steps produce a negative image. The physical density of the developed grains corresponds to the gray-level of the final output image. Since the grain blocks light, a photographic image is in fact a binary function which is equal to 0 in the areas covered by the grains, and equal to 1 otherwise.

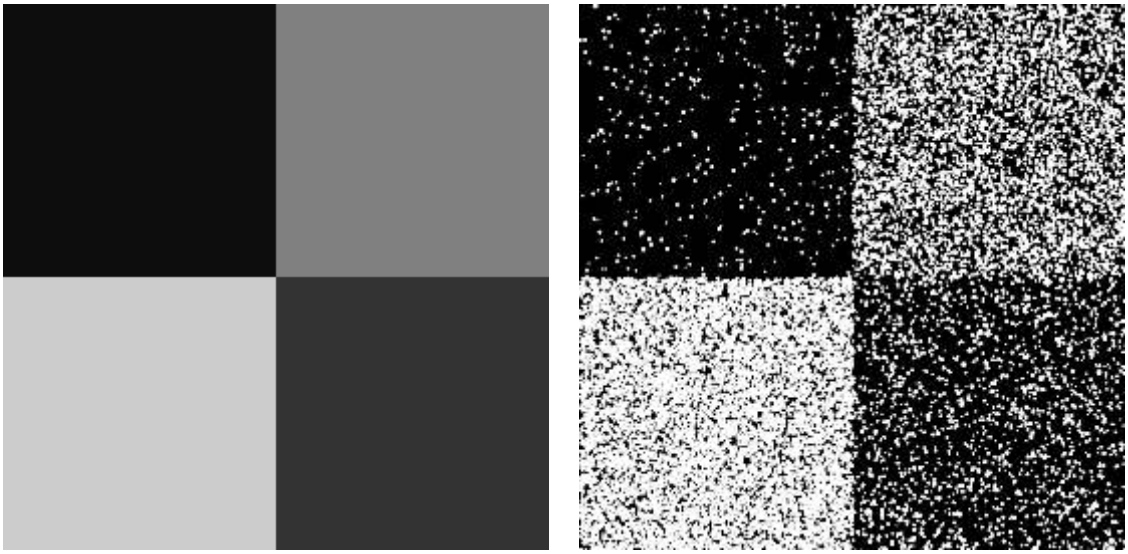
The final positive image takes form on photographic (photosensitive) paper. To do this, light is shone through the negative film onto the photographic paper. During this step, the image is typically enlarged by a factor of about ten times. After the exposure of the photographic paper, a *positive* representation of the original image has been recorded. As a simplification, we shall consider that the photographic paper is a continuous recording material, even if the photographic paper can contain its own "grain". Thus we only consider the grain produced by the original, negative, film.

2.1 Previous Work

The silver-halide photographic process has been extensively studied since the beginning of the twentieth century. Gurney and Mott [7] proposed a comprehensive physical model of the process, which we have presented above. Nutting [14] was the first to study the statistical properties of the "random dot model" for film grain, which is strongly linked to the model proposed in [13]. Initially, most of the work in this area was dedicated to understanding the statistical properties of the grain, such as the standard deviation of the optical density of the emulsion, which was named the "granularity" of an emulsion. Another similar quantity is that of Selwyn granularity [20], which is basically the granularity defined in such a fashion that it is independent of the size of the aperture in which the granularity is measured. A good summary of these common notions may be found in the paper of Bayer [2] in the context of the random dot model. Much of the subsequent analog literature is concerned with proposing mathematical models [3, 10, 19] which imitate the perceived effect of grain "clumping". This clumping, or agglomeration, of grains is an essential visual feature of graininess.

In more modern work, the main goal is grain *synthesis*. The most popular method seems to be to use real scanned examples of film grain. Film grain synthesis products such as DxO's "FilmPack"[2] and Grubba Software's "TrueGrain"[3] tools take this approach. More precisely, a single grain image is saved for each film type. Similarly, Schallauer and Mörzinger [16] copy grain from scanned analog images and synthesize a new grain image from these examples, which they then apply to the image in an additive fashion. A similar approach is used in the Film Emulation feature of the G'MIC free

[2]DxO Film Pack 5, 2016. http://www.dxo.com/us/photography/photo-software/dxo-filmpack
[3]Grubbasoftware TrueGrain, 2015. http://grubbasoftware.com/



Input pixels (close-up) Inhomogeneous Boolean model

Figure 1: **Illustration of inhomogeneous Boolean model**. The local intensity $\lambda(y)$ of the inhomogeneous Boolean model is chosen to respect the input pixel gray-levels.

software[4] using the random phase texture algorithm [5] to synthesize large grain textures from small stored samples. The Heeger-Bergen texture synthesis [8] algorithm is used by Bae et al. [1] on a constant gray-level area in an example image to produce film grain. Stephenson and Saunders [17] filter white noise in the Fourier domain. Yan et al. [22] propose an additive film grain model with signal-dependent noise. A drawback is that their approach supposes that film grain noise is spatially uncorrelated, which is clearly unrealistic. Oh et al. [15] propose an auto-regressive model for film grain removal and synthesis. They point out that spatial correlation is crucial for producing realistic film grain. However, they consider that an input grainy image is available, and that the characteristics of the grain may be extracted.

The main disadvantage of these approaches is that the models are not linked to the physical photographic process. There are several advantages of proposing a physically-based model [13]. Firstly, physically meaningful parameters can be tuned to produce different graininess. Secondly, there is no "blending" process to add the grain; the image itself results from the grain. Finally, it is possible to render images at any resolution, which is not the case for example-based algorithms which scan grain examples at a fixed resolution.

3 Stochastic Film Grain Model and Rendering Algorithm

The goal of the film grain rendering is to imitate the analog photographic process as closely as possible, in order to produce a realistic grainy image. Newson et al. [13] employ an inhomogeneous Boolean model [4] to achieve this goal. Here, we analyse this algorithm in depth and provide precise implementation details so that it is easily reproducible. We start by presenting the model used to represent film grain: the Boolean model.

3.1 The Boolean Film Grain Model

The Boolean model is a random set denoted by Z, which is the union of a sequence of randomly distributed disks whose centers $\{x_i\}$ are uniformly distributed in the plane with a Poisson process.

[4]GREYC's Magic for Image Computing (G'MIC). http://gmic.eu/

Note that the Boolean model is in fact much more general than this, but we present it in the case of disks as used in [13]. The radii $\{r_i\}$ of the disks are identically and independently distributed (i.i.d.), with mean and variance which are independent of the positions $\{x_i\}$. Thus, the model is in fact a continuous binary representation of the input image: it is equal to 1 at any point covered by a disk, and 0 otherwise. This Boolean model is employed to recreate the physical reality of film grain. The density of the disks is chosen to reflect the local gray-level of the image. In other words, if we view the model from very "far away" it should average out to maintain the gray-level of the input digital image. The density is governed by the parameter λ of the Poisson process. Let $\mathbb{1}_Z(y)$ denote the indicator function of Z, equal to 1 when y is covered by Z and 0 otherwise. The probability of a point y being covered by a disk in a model with constant λ is

$$\mathbb{P}(\mathbb{1}_Z(y) = 1) = 1 - \exp(-\lambda \mathbb{E}\left[\pi r_1^2\right]), \tag{1}$$

where $\mathbb{E}[\pi r_1^2]$ is the average area covered by a disk [18]. An illustration of some Boolean models for different average gray-levels is shown in Figure 1.

As mentioned above, the parameter λ is chosen to respect the local image gray-level. Let $u : \{0, \dots, m-1\} \times \{0, \dots, n-1\} \subset \mathbb{N}^2 \to [0, u_{max}]$ be this discrete input image. Firstly, we normalize the input image u to the interval $[0, 1)$ by defining $\tilde{u}(y) = \frac{u(y)}{u_{max}+\varepsilon}$, where ε represent a small parameter, and u_{max} is the maximal possible gray-level value for a given image format. We restrict the image to $[0, 1)$, since the Boolean model with $\mathbb{P}(\mathbb{1}_Z(y) = 1) = 1$ would lead to an infinite intensity parameter λ. In order to respect the gray-level for a given pixel, following Equation (1), the parameter λ is set to

$$\lambda(y) = \frac{1}{\mathbb{E}[\pi r_1^2]} \log\left(\frac{1}{1 - \tilde{u}(\lfloor y \rfloor)}\right). \tag{2}$$

This defines a piece-wise constant function $\lambda(\cdot)$ on the continuous image domain $[0, m) \times [0, n)$. Note that we use the convention that a unit square of $\mathbb{R}^2$ corresponds to an input pixel. In particular, the grain radii r_i are expressed with respect to this convention.

A Boolean model with a variable λ is called an *inhomogeneous* Boolean model. In practical terms, one simulates the inhomogeneous Boolean model by drawing, for each pixel (i, j), the number of grains Q from a Poisson distribution of parameter $\lambda(i, j)$ given by Equation (2). Then the Q centers x_i of the grains are drawn from the distribution $\mathcal{U}([i, i+1) \times [j, j+1))$. Finally, a radius r_i is drawn for each center x_i. Newson et al [13] propose the choice of either constant radii or radii following a log-normal distribution [11]. Throughout the rest of this paper, we refer to the mean and variance of this grain radius distribution with μ_r and σ_r^2.

To sum up, this inhomogeneous Boolean model represents a binary film grain whose density respects the gray-level intensity of the input digital image. We note that grain shapes other than disks can easily be included in the Boolean model, such as triangular grains. However, this does not significantly impact the visual result, so we do not discuss it here.

3.2 The Filtered Boolean Model

The continuous inhomogenous Boolean model is binary, but a filtering step produces the output gray-levels. This filtering models the processes which an analog image undergoes before being observed by a human (photo enlargement, human vision). This filtering step is also very important in producing the "grainy" effect in the output image. In the grain rendering algorithm, Newson et al. [13] model this step with a single Gaussian kernel ϕ of variance σ^2. For any continuous point p (expressed in the input coordinate system), the filtered Boolean model evaluated at that point is given by

$$v(p) = \phi * \mathbb{1}_Z(p) = \int_{\mathbb{R}^2} \phi(t) \mathbb{1}_Z(p - t) dt. \tag{3}$$

In theory, this allows us to create an output image on any arbitrary discrete grid. However, we shall consider that the aspect ratio of the input image is maintained, so that there is a zoom factor $s \in (0, +\infty)$ such that the output image size is (sm, sn). In practical terms, this means that to convert from the input to output coordinate systems, we simply multiply the input coordinates by s.

A final, important, issue in the algorithm is how to evaluate the continuous convolution of the filter with the random function $\mathbb{1}_Z$, which is not a trivial task. The proposed solution is to use a Monte Carlo method to approximate the integral needed for this convolution. The Monte Carlo approach consists in evaluating $\mathbb{1}_Z$ for a sequence of N i.i.d. points following a Gaussian distribution of mean y and of variance σ^2. Therefore, the output image at a given point y, expressed in terms of the *output grid* ($y \in \{0, \dots, ms-1\} \times \{0, \dots, ns-1\}$), is defined as

$$v(y) = \frac{1}{N} \sum_{k=1}^{N} \mathbb{1}_Z(\frac{y - \xi_k}{s}), \tag{4}$$

with $\xi_k \sim \mathcal{N}(0, \sigma^2 I_2)$, where I_2 represents the identity matrix of size 2×2. Note that the filtering parameter σ^2 is expressed in terms of the *output* pixel size. This is natural, since the filtering is related to the *observed* image, and not the underlying model. We divide by s in Equation (4) since $\mathbb{1}_Z$ is defined with respect to the *input* grid. As N increases, according to the law of large numbers one has

$$\frac{1}{N} \sum_{k=1}^{N} \mathbb{1}_Z(\frac{y - \xi_k}{s}) \xrightarrow[N \to +\infty]{} \mathbb{E}[\mathbb{1}_Z(\frac{y - \xi_1}{s})] = \int_{\mathbb{R}^2} \mathbb{1}_Z(\frac{y - t}{s}) \phi(\frac{t}{s}) dt = v(y), \tag{5}$$

where ϕ is the probability density function of the Gaussian distribution $\mathcal{N}(0, \sigma^2 I_2)$. In practice, the same random offsets ξ_k are used for each pixel. This avoids drawing random numbers excessively, which can slow the algorithm down.

4 Algorithmic Details and Implementations

Several important algorithmic choices must be made when implementing the film grain rendering approach presented in Section 3. Our algorithm essentially boils down to evaluating $v(y) = \frac{1}{N} \sum_{k=1}^{N} \mathbb{1}_Z(\frac{y - \xi_k}{s})$ for each output pixel y. This may also be viewed as averaging a series of binary images v_k which are in turn produced by evaluating the function $\mathbb{1}_Z$ over the output grid shifted by the ξ_k's. The exact manner in which this operation is done has considerable practical consequences.

Conceptually, the most straightforward way would certainly be to sample all of the grains, and then render them on the output image. However, this can require prohibitively large memory storage. For example, if we suppose a grain radius $r = \frac{1}{40}$, with a high resolution image (2048×2048) with constant gray-level values of 128 everywhere, 35 GB of memory is needed to store the grain positions and radii with single precision floating point.

Therefore, two different implementations of the rendering algorithm are proposed. Both of these approaches avoid storing the grain information. The first, which is referred to as the"grain-wise" algorithm, samples a grain and then determines its effect on each Monte Carlo iteration. For this, we store N temporary binary images, each of which correspond to one Monte Carlo iteration, and average them to produce the final output image. The second, which is referred to as the "pixel-wise" algorithm, renders each output pixel by locally generating the Poisson process $\{x_i, r_i\}$ in a reproducible manner using pseudo-random number generation. We propose two different approaches since each one has different advantages depending upon the grain parameters used. In what follows, we explain these algorithms in detail, and we then discuss how to choose the most efficient one depending on the grain distribution parameters.

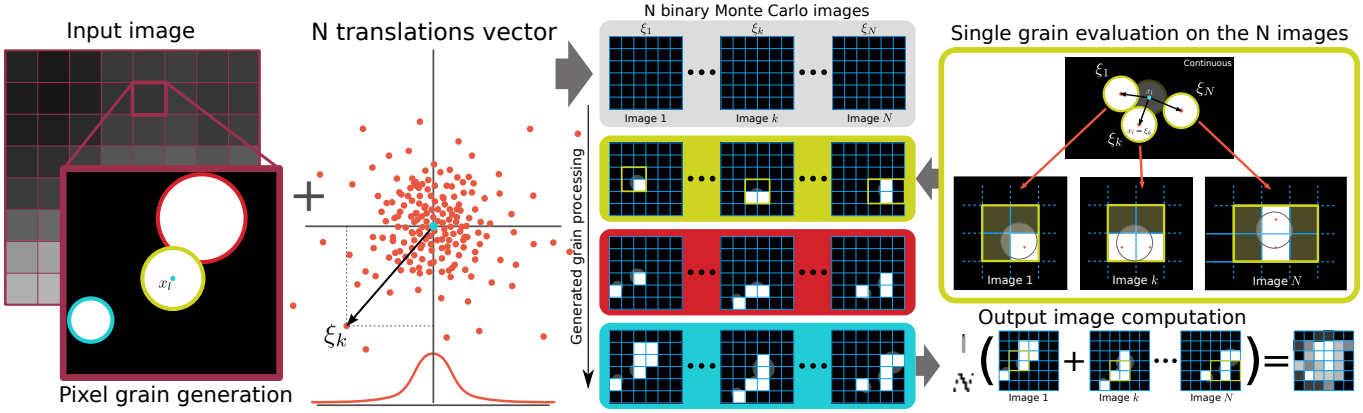


Figure 2: **Illustration of the grain-wise algorithm**. First, we sample the N Monte Carlo vectors ξ_k (illustrated with red dots). We create N binary images (the collection of blue grids in the middle), of the same size as the output image. These images record the effect of each sampled grain. For each input pixel, we sample the grains of our Boolean model which are centered inside the considered pixel. The final result is the average of these N binary images.

4.1 Grain-Wise Algorithm

This approach samples each grain sequentially. Once a grain is sampled, its effect on each Monte Carlo iteration is computed, that is to say whether the grain covers the point $\frac{y+\xi_k}{s}$, for $k = 1, \dots, N$. The grain's information is then discarded. This "influence" is stored in the form of a sequence of N *binary* images v_k, $k \in \{1 \dots N\}$ (see the blue grids in Figure 2). Each image represents one Monte Carlo iteration, and a pixel of one of these binary images is set to 1 if *at least one* grain covers the center of the pixel. A Monte Carlo iteration consists in evaluating the Boolean model on the randomly shifted grids $(\xi_k + \{0, \dots, ms-1\} \times \{0, \dots, ns-1\})\frac{1}{s}$, with $\xi_k \sim \mathcal{N}(0, \sigma^2 I_2)$. Therefore, for each grain drawn, and for each Monte Carlo shift ξ_k, we determine all the output pixels $y \in \{0, \dots, ms-1\} \times \{0, \dots, ns-1\}$ in the image v_k which verify $\mathbb{1}_Z(\frac{y-\xi_k}{s}) = 1$, in other words, for the grain q, we evaluate

$$||\frac{y-\xi_k}{s} - x_q|| \leq r_q. \tag{6}$$

In reality, we only need to check the pixels $y \in \mathcal{B}(x_q + \frac{\xi_k}{s}, r_q)$. The output image v is given by the average of all the images

$$v(y) = \frac{1}{N} \sum_{k=1}^{N} v_k(y). \tag{7}$$

This algorithm is fully described in Algorithm 1, and is illustrated in Figure 2.

4.2 Pixel-Wise Algorithm

The second algorithm proposed in [13] is referred to as the "pixel-wise" approach. This method processes the *output* pixels independently, and relies on repeatable pseudo-random number (PRN) generation to access any grain's information at will. Using a partition of $\mathbb{R}^2$ into disjoint cells, it is possible to generate the grain information in any cell in a repeatable fashion, meaning that their information does not have to be stored in memory. By using the coordinates of the cell and a local PRN generator [21, 9] the number of grains whose centers belong to this cell (and those grains' information) can be generated. Our implementation uses the PRN generation employed for procedural texture generation by Galerne et al. [6].

The pixel-wise algorithm basically asks the following question: for each $y \in \{0, \dots, ms-1\} \times \{0, \dots, ns-1\}$, what is the value of $v(y)$? For each y, we must average the value of $\mathbb{1}_Z(\frac{y-\xi_k}{s})$ over all

Algorithm 1 The proposed "grain-wise" film grain rendering algorithm. The loop colored in blue is parallelized.

Data: $u : \{0, 1, \dots m-1\} \times \{0, 1, \dots, n-1\} \to [0, u_{max}]$: input image
Parameters:
$\mathcal{D}(\mu_r, \sigma_r^2)$: distribution of grain radii
s: output zoom
σ: standard deviation of the Gaussian low-pass filter
N: number of iterations in the Monte Carlo method
Result: v: Synthesized, film grain image

Set up N binary images of size $ms \times ns$ and draw N random offsets:
for $k = 1$ **to** N **do**
    $v_k = 0$
    $\xi_k \leftarrow \mathcal{N}(0, \sigma^2 I_2)$

foreach $(i, j) \in \{0, \dots, m-1\} \times \{0, \dots, n-1\}$ **do**
    $\tilde{u}(i, j) = \frac{u(i,j)}{u_{max}+\varepsilon}$
    $\lambda = \frac{1}{\pi(\mu_r^2+\sigma_r^2)} \log \frac{1}{(1-\tilde{u}(i,j))}$
    $Q \leftarrow \text{Poisson}(\lambda)$
    Sample $x_{q=1\dots Q}$ from $\mathcal{U}\left([i, i+1) \times [j, j+1)\right)$
    Sample grain radii $r_{q=1\dots Q} \sim \mathcal{D}(\mu_r, \sigma_r^2)$
    for $k = 1$ **to** N **do**
        for $\ell = 1$ **to** Q **do**
            $t = x_\ell + \frac{1}{s}\xi_k$
            foreach $y \in \{0, \dots, sm-1\} \times \{0, \dots, sn-1\}$ *s. t.* $\|t - \frac{y}{s}\|_2 \leq r_\ell$ **do**
                $v_k(y) = 1$

foreach $y \in \{0, \dots, sm-1\} \times \{0, \dots, sn-1\}$ **do**
    $v(y) = 0$
    for $k = 1$ **to** N **do**
        $v(y) = v(y) + v_k(y)$
    $v(y) = \frac{1}{N}\, v(y)$

return(v)

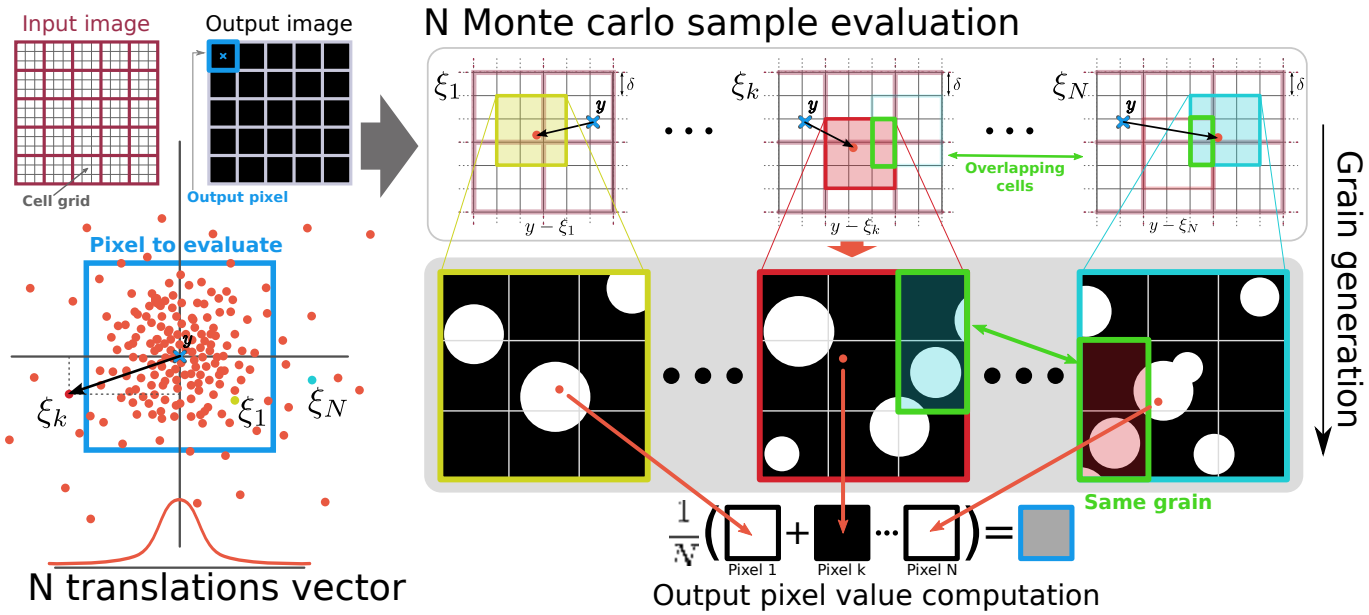


Figure 3: **Illustration of the pixel-wise algorithm**. At the beginning of the algorithm, we draw the N Monte Carlo offset vectors $\{\xi_k\}$ (red points on the left). The algorithm then evaluates the gray-level for each output pixel (in blue). This is done by averaging the evaluations of $\mathbb{1}_Z(y-\xi_k)$ for $k = 1 \dots N$.

the ξ_k's (see Equation (4)). This boils down to determining whether each shifted *output* pixel $y - \xi_k$ is covered by at least one grain of the model Z. Crucially, using the cell partition introduced above, we can access the grains in any given cell. For a constant grain radius, we only need to inspect those cells which intersect the ball of radius r centered on the point $y - \xi_k$. Therefore, we generate the grains in the cells $\left\{ \left\lfloor \frac{\frac{y-\xi_k}{s} - r_m}{\delta} \right\rfloor, \dots, \left\lfloor \frac{\frac{y-\xi_k}{s} + r_m}{\delta} \right\rfloor \right\}$. If we find a grain which covers the position $\frac{y-\xi_i}{s}$, we increment the output pixel $v(y)$ by one. The final output pixel is therefore given by $\frac{1}{N} v(y)$. This is repeated for each pixel y, however this can be done completely in parallel.

From a practical point of view, we now have three grids: the initial input grid $\{0, \dots, m-1\} \times \{0, \dots, n-1\}$ (the red grid in Figure 3), the cell grid $\{0, \delta, \dots, m-1\} \times \{0, \delta, \dots n-1\}$ (in gray as a subdivision of the input grid) and the output grid $\{0, \frac{1}{s}, \dots, m-1\} \times \{0, \frac{1}{s}, \dots n-1\}$ (illustrated with the blue pixel on the left). The first one defines the Boolean model, the second one defines the output pixels and the last one is used for the pseudo-random number generation. The cell size δ is defined such that the cell grid is a subdivision of the input grid, in other words $\delta = \frac{1}{n}$ for some integer n. For variable radii, which can take theoretically unbounded values, a maximum possible radius r_m is specified, so that $r_i \leq r_m, \forall i \in \mathbb{N}$.

The pixel-wise approach is described in detail in the pseudo-code of Algorithm 2 and Algorithm 3, and is illustrated in Figure 3.

4.3 Comparison and Choice of Algorithm

We have presented two different algorithmic approaches for evaluating the same stochastic model used in the film grain rendering of [13]. These two approaches have very different advantages and disadvantages, meaning that they behave very differently in terms of execution when different parameters are used (μ_r, σ_r, etc.). We now look at the time complexities of the two algorithms. Consider an output zoom factor of $s = 1$ and an image $\tilde{u} = \frac{1}{2}$ everywhere. In the case of the grain-wise algorithm, for each input pixel, on average $\lambda = \frac{\log(2)}{\pi(\mu_r^2 + \sigma_r^2)}$ grains are processed. For each grain, we must evaluate its effect on N intermediate images, with each evaluation requiring four Euclidean

Algorithm 2 The proposed "pixel-wise" film grain rendering algorithm. The loop colored in blue is parallelized.

Data: $u : \{0, 1, \dots m-1\} \times \{0, 1, \dots, n-1\} \to [0, u_{max}]$: input image
Parameters:
$\mathcal{D}(\mu_r, \sigma_r^2)$: distribution of grain radii
r_m: maximum radius allowed
s: output zoom
σ: standard deviation of the Gaussian low-pass filter
N: number of iterations in the Monte Carlo method
Result: v: Image rendered with film grain

$\delta = \frac{1}{\lceil \frac{1}{\mu_r} \rceil}$

for $k = 1$ **to** N **do**
  $\xi_k \leftarrow \mathcal{N}(0, \sigma^2 I_2)$

foreach $y \in \{0, \dots, sm-1\} \times \{0, \dots, sn-1\}$ **do**
  $v(y) = 0$
  for $k = 1$ **to** N **do**
    $(x_g, y_g) = \frac{1}{s}(y + \xi_k)$
    Evaluate $\mathbb{1}_Z(\frac{1}{s}(y + \xi_k))$ *:*
    $v(y) = v(y) +$ EvaluateLocalBooleanIndicator(x_g, y_g) (see Algorithm 3)
  $v(y) = \frac{1}{N} v(y)$
return(v)

Algorithm 3 EvaluateLocalBooleanIndicator: Evaluation of the Boolean model at a point (x, y)

Data: u: input image
(x, y): point to evaluate
Parameters:
$\mathcal{D}(\mu_r, \sigma_r^2)$: distribution of grain radii
δ: cell size
Result: Binary result, whether (x, y) is covered by a ball
foreach $(i_\delta, j_\delta) \in \{\lfloor \frac{x-r_m}{\delta} \rfloor, \dots, \lfloor \frac{x+r_m}{\delta} \rfloor\} \times \{\lfloor \frac{y-r_m}{\delta} \rfloor, \dots, \lfloor \frac{y+r_m}{\delta} \rfloor\}$ **do**
  $\tilde{u} = \frac{u(\delta.i_\delta, \delta.j_\delta)}{u_{max} + \varepsilon}$
  $\lambda = \frac{1}{\pi(\mu_r^2 + \sigma_r^2)} \log \frac{1}{(1-\tilde{u})}$
  $Q \leftarrow$ Poisson(λ)

  for $\ell = 1$ **to** Q **do**
    Draw the centre of the grain using the cell coordinate (i_δ, j_δ)
    $x \leftarrow \delta(i_\delta, j_\delta) + \mathcal{U}([0, \delta) \times [0, \delta))$
    Draw the grain radius using the cell coordinate (i_δ, j_δ)
    $r = \min(\mathcal{D}(\mu_r, \sigma_r^2), r_m)$
    if $||(x, y) - x||_2 < r$ **then**
      return(1)
return(0)

distance calculations. Therefore, the complexity of the grain-wise algorithm is

$$C_0 = mnN\frac{4\log(2)}{\pi(\mu_r^2+\sigma_r^2)}. \tag{8}$$

In the case of the pixel-wise algorithm, for each output pixel we evaluate N points $y-\xi_i$ for the Monte Carlo simulation. For each $y-\xi_i$, we must visit $(2\lfloor\frac{p_{1-\alpha}}{\mu_r}\rfloor+1)^2$ cells which may contain grains overlapping the point, where $p_{1-\alpha}$ is the $(1-\alpha)$th quantile of the grain radius distribution. If we suppose that $\delta=\mu_r$ (which is very often the case), then, on average there are $\frac{\mu_r^2\log 2}{\pi(\mu_r^2+\sigma_r^2)}$ grains in each cell, and we carry out a Euclidean distance calculation for each grain. Therefore, the complexity of the pixel-wise algorithm is

$$C_1 = mnN(2\lfloor\frac{p_{1-\alpha}}{\mu_r}\rfloor+1)^2\frac{\mu_r^2\log(2)}{\pi(\mu_r^2+\sigma_r^2)}. \tag{9}$$

According to this analysis, we have $C_0 < C_1$ (the grain-wise algorithm is faster) when

$$\frac{2}{\mu_r} < 2\lfloor\frac{p_{1-\alpha}}{\mu_r}\rfloor+1. \tag{10}$$

This is true when the ratio $\frac{p_{1-\alpha}}{\mu_r}$ is relatively large. In the degenerate case when $\sigma_r=0$, we have $p_{1-\alpha}=\mu_r$, so that the pixel-wise algorithm clearly has a lower time complexity. However, when μ_r is large, the grain-wise algorithm quickly becomes faster. This is intuitively correct, since the larger the grains are, the less grains the grain-wise algorithm has to process, whereas the computational load of the pixel-wise algorithm stays the same. To summarize, the complexity comparison depends on the ratio $\frac{p_{1-\alpha}}{\mu_r}$. Let us analyze this ratio

$$1-\alpha = \mathbb{P}\left(r_i \le p_{1-\alpha}\right) = \mathbb{P}\left(r_i^2 \le p_{1-\alpha}^2\right) = 1-\mathbb{P}\left(r_i^2 \ge p_{1-\alpha}^2\right). \tag{11}$$

Using Markov's inequality we have

$$\begin{aligned}\mathbb{P}\left(r_i^2 \ge p_{1-\alpha}^2\right) &\le \frac{\mathbb{E}[r_i^2]}{p_{1-\alpha}^2} \\ &\le \frac{\sigma_r^2+\mu_r^2}{p_{1-\alpha}^2}.\end{aligned} \tag{12}$$

Finally, we have

$$\begin{aligned}1-\alpha &\ge 1-\frac{\sigma_r^2+\mu_r^2}{p_{1-\alpha}^2} \\ \frac{p_{1-\alpha}}{\mu_r} &\le \sqrt{\frac{\frac{\sigma_r^2}{\mu_r^2}+1}{\alpha}}.\end{aligned} \tag{13}$$

We observe that, for a fixed α, the ratio $\frac{p_{1-\alpha}}{\mu_r}$ depends on the ratio $\frac{\sigma_r^2}{\mu_r^2}$. Thus, the complexity of the pixel-wise algorithm increases with a larger ratio $\frac{\sigma_r}{\mu_r}$.

Numerical simulations indicate that the pixel-wise algorithm has a lower complexity in the cases where $\sigma_r < \mu_r$. However, the timings for the actual algorithms show that the grain-wise algorithm becomes faster at a significantly lower value of σ_r than theoretically predicted. This is due to several

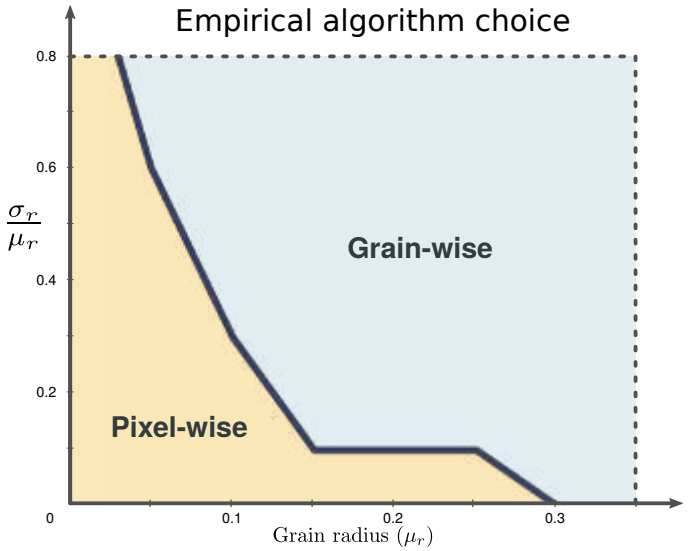


Figure 4: **Illustration of the automatic choice between the grain-wise and pixel-wise algorithms**. The two parameter domains in which each algorithm is applied are shown by two different colors. Generally speaking, the lower the value of σ_r, the faster the pixel-wise approach is.

subtleties which are not easy to take into account. The first point concerns grain generation. In the grain-wise algorithm, *all* grains are *necessarily* generated. In the pixel-wise approach, once a point $y - \xi_k$ is covered by at least one grain, then the algorithm continues on to the next Monte Carlo iteration. Thus, in very light areas of an image, where there are many grains, the grain-wise algorithm will be much slower than the pixel-wise approach. Conversely, if the average μ_r and standard deviation σ_r of the grains are very large, then the grain-wise algorithm will be faster than the pixel-wise approach. Indeed, there are less grains to generate, which favors the grain-wise approach, but the pixel-wise approach needs to scan more pixels due to an increased value of r_m. Another factor influencing the time complexities is the amount of random number generation work required. Indeed, in the grain-wise approach, the quantity Q (the number of grains) is generated once for each input pixel. In the pixel-wise approach, a Q is generated for each cell of size δ. This significantly changes the work required by either approach.

To reflect the intricate nature of these time complexities, we base our choice of algorithm on empirical evidence taken from a series of experiments carried out on various images. In these experiments, we compared the execution times of the grain-wise and pixel-wise algorithms on a series of twenty images. For each image, and for both algorithms, we varied the parameters μ_r and $\frac{\sigma_r}{\mu_r}$ to study the performance of the algorithms. In our experiments, μ_r varied in the set $\{0.03, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$, and $\frac{\sigma_r}{\mu_r}$ varied from 0.1 to 0.9 with steps of 0.1. Figure 4 shows the results of these experiments and the parameter domains in which one algorithm is faster than the other. In the source code provided, the algorithm choice is taken automatically according to this empirical study.

4.4 Parallelization of Algorithms and Accelerations

Both the grain-wise and pixel-wise algorithm can be parallelized with great ease. In the case of the grain-wise algorithm, we parallelize over the input pixels. In the case of the pixel-wise algorithm, we parallelize over the output pixels. The loops which we parallelize are highlighted in blue in Algorithms 1 and 2. In our code, publicly available on the IPOL website, we parallelize the algorithm in the described manner using the OpenMP library, if it is available on the machine which compiles and executes the code.

We have also carried out a thorough analysis of our code using the valgrind tool in order to

further decrease execution times and obtain optimal performance. In particular, in the case of the pixel-wise algorithm, there are several quantities which may be pre-computed to avoid unnecessary computations. Since the pixel-wise approach potentially scans many cells, we would like to avoid recomputing the local values of λ, which requires the computation of a log. Therefore, we determine λ for each possible gray-level value, of which there are 256 in general. Finally, we avoid re-drawing the Gaussian offset vectors ξ_k for each output pixel by calculating them once at the beginning of the algorithm, and using the same for all the output pixels.

4.5 Algorithm Parameters

The variables and parameters of the grain rendering algorithm are summarized in Table 1. The algorithm contains three main tunable parameters which may be changed by the user to achieve grains of different visual aspects. The first two are the average grain radius, μ_r and the grain standard deviation σ_r. Increasing μ_r and σ_r increases the graininess of the film grain. These parameters are in units relative to the input image grid, that is to say we consider the input image grid to define a unit square. The third is the zoom factor which determines the output image size. This can be increased (or decreased) in order to render the film grain model at different resolutions. As in the case of the grain radius parameters, the output resolution is relative to the input image. Additionally, it is possible to render only a part of the image, which is useful when we wish to test extremely large zooms. Therefore, in our source code, we also allow the parameters (x_A, y_A), (x_B, y_B), m_{out} and n_{out} to be modified. The first two couples of real numbers specify the top left and bottom right limits of the input image where we wish to render the film grain, and m_{out} and n_{out} represent the number of rows and columns in the output image.

The rest of the parameters can be considered to be fixed, or best chosen automatically. The number of samples N in the Monte Carlo simulation can be reduced to increase speed, however this is at the cost of accuracy. The choice between the grain-wise and pixel-wise algorithm is automatically determined by the code, however it can also be fixed by the user. The automatic choice is taken in order to reduce execution time. We take the decision based on empirical experiments comparing the execution times of the grain-wise and pixel-wise algorithms. The standard deviation σ of the Gaussian filter is set by default to 0.8 output pixels. This can be modified to produce a more or less smoothed image, although there are theoretically sound reasons to leave it at the default value [12]. Finally, when we consider variable radii, we set r_m (the maximum radius allowed in the pixel-wise algorithm) to the $(1-\alpha)$th quantile of the radius distribution, with α very small. In practice, we take $1-\alpha = 0.999$.

5 Results

We now show some visual results of our film grain rendering algorithm. We have implemented our code in the C++ programming language.

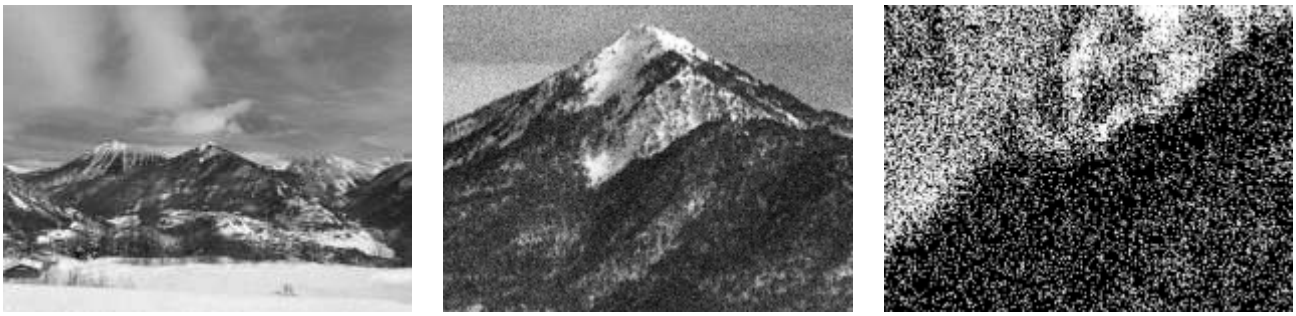
Figure 5 shows some grain rendering results on two images. We illustrate the algorithm's capacity to provide arbitrary zooming on the image, so that the individual grains may be observed. We used a constant grain radius of $\mu_r = 0.1$ pixels in these examples.

5.1 Grain Variability

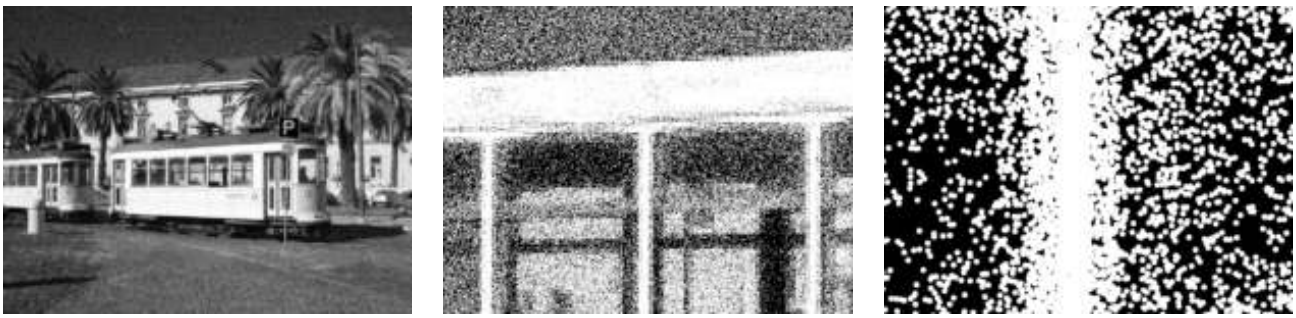
It is important to demonstrate that the film grain model contains parameters which the user can tune to vary the visual aspect of the grain, and therefore imitate different types of grain. The two main parameters which determine this aspect are the average grain radius μ_r and the standard deviation σ_r

Symbol	Interpretation
$u \in \mathbb{R}_{m,n}$	Input image
$v \in \mathbb{R}_{sm,sn}$	Output image
$s > 0$	Output image zoom
$\mu_r > 0$	Average grain radius (in input pixels)
$\frac{\sigma_r}{\mu_r} \in [0, 1)$	Standard deviation of grain radius with respect to average radius (in input pixels)
$\sigma > 0$	Standard deviation of the Gaussian filter (in output pixels)
$N \in \mathbb{N}^*$	Number of iterations in the Monte Carlo simulation
$\epsilon \in \mathbb{R}$	Small parameter, for normalization of image to $[0, 1)$, 0.1 gray-levels
$r_m = p_{1-\alpha}$	Maximum radius allowed r_m, where $p_{1-\alpha}$ is the $(1-\alpha)$th quantile of the radius distribution. $\alpha = 0.001$

Table 1: **Algorithm execution times**. In this Table, we list the different variables and parameters and their meaning and interpretation.

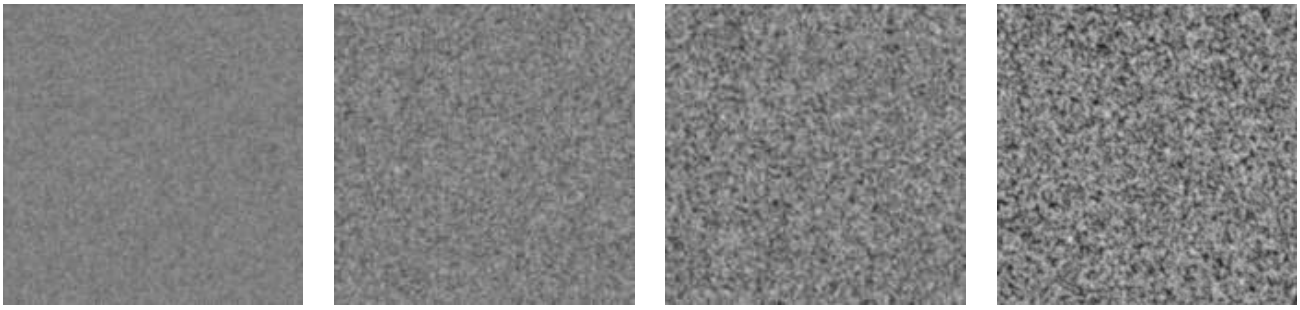


Full resolution (2048×1536) Zoom $5\times$ Zoom $30\times$

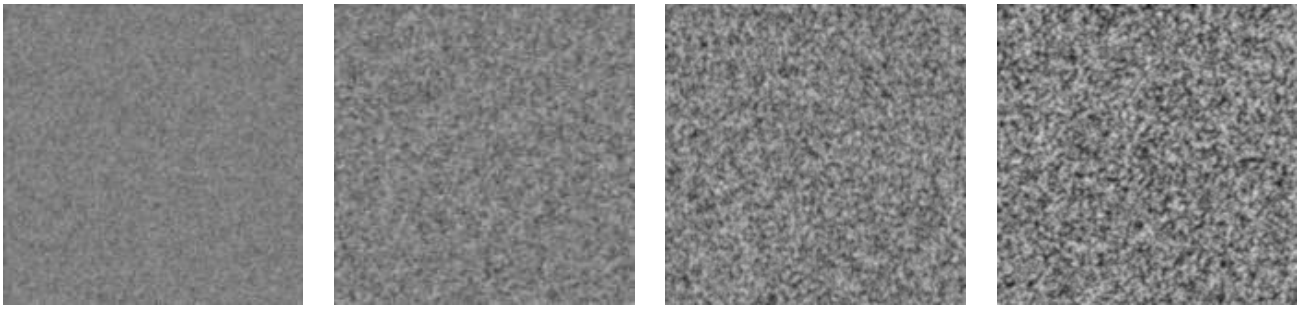


Full resolution (604×453) Zoom $5\times$ Zoom $30\times$

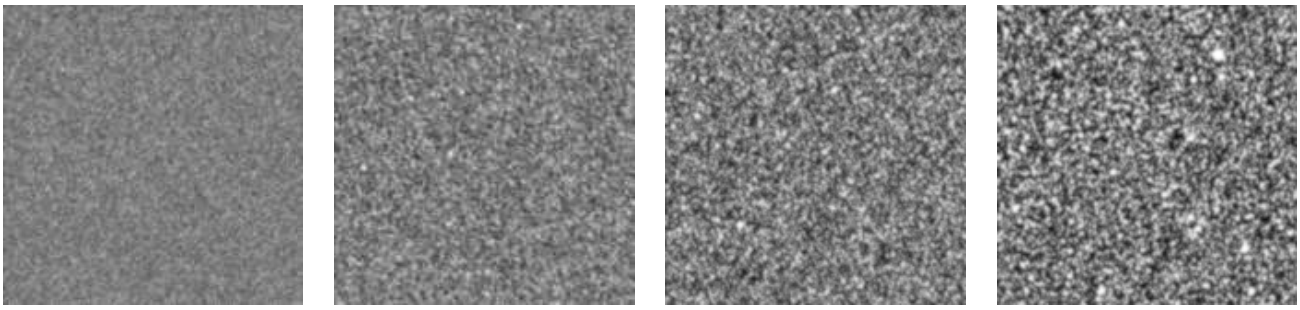
Figure 5: **Illustration of film grain rendering at any resolution**. The use of a continuous film grain model means that we are able to render the image at any desired resolution. We specify the input image size to explain the difference in the size of the disks in the zoomed images.



$\mu_r = 0.025,\ \sigma_r = 0$ $\mu_r = 0.05,\ \sigma_r = 0$ $\mu_r = 0.075,\ \sigma_r = 0$ $\mu_r = 0.1,\ \sigma_r = 0$



$\mu_r = 0.025,\ \sigma_r = \frac{1}{4}\mu_r$ $\mu_r = 0.05,\ \sigma_r = \frac{1}{4}\mu_r$ $\mu_r = 0.075,\ \sigma_r = \frac{1}{4}\mu_r$ $\mu_r = 0.1,\ \sigma_r = \frac{1}{4}\mu_r$



$\mu_r = 0.025,\ \sigma_r = \frac{1}{2}\mu_r$ $\mu_r = 0.05,\ \sigma_r = \frac{1}{2}\mu_r$ $\mu_r = 0.075,\ \sigma_r = \frac{1}{2}\mu_r$ $\mu_r = 0.1,\ \sigma_r = \frac{1}{2}\mu_r$

Figure 6: **Film grain texture with varying parameters.** In this Figure, we show the effect of varying grain size on the results of our grain synthesis. We vary the average size of the grains as well as the standard deviation of a log-normal grain distribution. It can be seen that using either constant or random grain sizes has a significant impact on the rendering results.

$\mu_r = 0.1$

$\mu_r = 0.3$

$\mu_r = 0.1$, close-up

$\mu_r = 0.3$, close-up

Figure 7: **Increasing film grain radius**. We show the effect of increasing the film grain radius on the rendering result.

Figure 8: **Colour film grain on a modern image**. This example shows that film grain can be used for artistic purposes to give images a vintage look. On the left part of the image is the digital input, on the right is the image with color grain rendering.

of this radius. As expected, increasing these parameters accentuates the"graininess" of the rendered result, however, there is also a significant visual difference between constant grain and variable grain. A series of experiments illustrating this may be seen in Figure 6. In Figure 7, we show the visual artistic effect achieved by increasing the (constant) grain radius. The radius parameter can therefore be tuned in order to create a more or less grainy image, depending on the look required by the user.

5.2 Color Photography

We may also wish to produce images with color film grain. Color film emulsions are made of several layers of normal silver-halide crystal emulsions which are chemically sensitized to different light wavelengths. It turns out that these crystals are naturally sensitive to blue light, therefore the color emulsion consists of a top layer of normal grains, followed by a yellow filter (to block blue light) and two following layers sensitive to green and red light, respectively. Since the grains only react to incoming photons of a certain color, it is a good approximation to simulate the film grain in each color channel independently. In summary, color film grain can be obtained by applying grain independently in each color channel. In Figure 8, we show a more modern image which has had color grain added to it. This shows that recent images can be rendered to give them a more vintage feel.

5.3 "Dithering" Effect

An interesting effect of adding film grain to a digital image is that certain compression artefacts can be attenuated. This is linked to a phenomenon known as dithering, where noise is added to quantized signals in order to reduce this quantization. Since the model of the image which we create is of a stochastic nature, this effect is visible on our results. An illustration of this effect is shown in Figure 9. The blocking artefacts due to the jpg compression are drastically reduced.

5.4 Algorithm Limitations

There are several limitations of the presented film grain rendering approach. The first is linked to the model used. Indeed, while the parameters μ_r and σ_r can be tuned to produce results with more or less graininess, there are certainly some film emulsion types which are difficult to emulate. An

Input Film grain rendering

Figure 9: **Illustration of the "dithering" effect.** On the left, a closeup on an image which has been highly compressed with jpg compression. On the right, the result of our film grain rendering. The blocking artefacts due to the jpg compression are clearly visible.

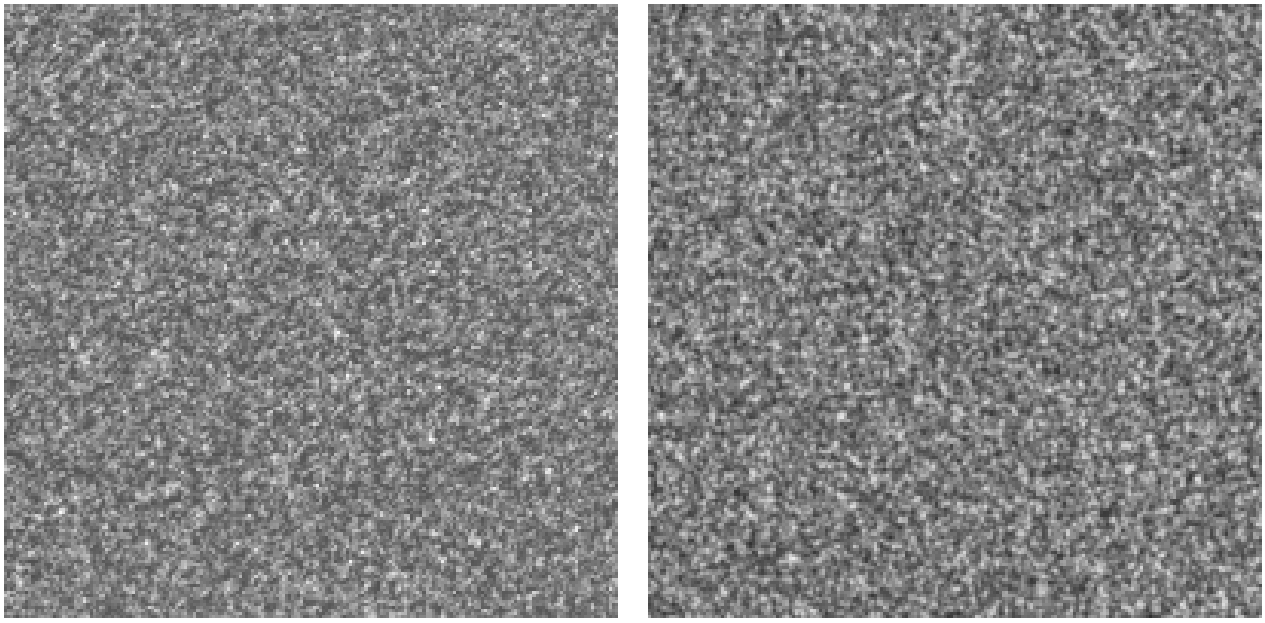
example of this limitation can be seen in Figure 10. Another issue with this model is film grain in dark areas. Indeed, in such areas, there may be only a few grains, which may result in an unrealistic visual effect. Finally, the algorithm is far from being real-time, which is necessary for use in image and film post-production. For example, in the case of a 512×512 constant image (equal to 128 everywhere), the parallelized grain-wise and pixel-wise algorithms take, respectively, 40 seconds and 2.4 seconds. It is crucial to address this point if the approach presented in this paper is to be widely used.

6 Conclusion

We have presented a film grain rendering algorithm based on the discrete evaluation of a filtered Boolean model using a Monte Carlo simulation. We have shown the film grain rendering results of the algorithm with black and white or color film grain for any digital image at any desired resolution. We have presented and analyzed two different implementations of this algorithm, which present different strengths depending upon the situation. We have provided a method to choose between the two, based on an analysis of the execution times when varying the grain parameters. We also demonstrate the visual variability of the grain when the parameters are tuned. The code of this implementation is publicly available at the IPOL website.

Acknowledgements

This work has been partially funded by the French Research Agency (ANR) under grant no. ANR-14-CE27-001 (MIRIAM). The authors thank Dick Dickerson and Silvia Zawadzki for their invaluable expertise and for the fruitful discussions we had about analog film.



"FilmPack" tool of DxO | Result of the Boolean model

Figure 10: **Limitation of the film grain model when imitating certain film emulsion types**. We used the "FilmPack" tool of DxO which uses scanned film grain examples. This example is taken from the "Ilford Delta 3200" emulsion type. The Boolean model does not provide the possibility to imitate the qualitative tendency of the Ilford emulsion to produce "white" grains rather than dark ones.

Image Credits

All images by Noura Faraj except:

 Creative Commons

References

[1] S. Bae, S. Paris, and F Durand, *Two-scale tone management for photographic look*, in ACM Transactions on Graphics, vol. 25, 2006. http://dx.doi.org/10.1145/1179352.1141935.

[2] B. E. Bayer, *Relation Between Granularity and Density for a Random-Dot Model*, Journal of the Optical Society of America, 54 (1964), pp. 1485–1490. http://dx.doi.org/10.1364/josa.54.001485.

[3] P. E. Castro, J. H. B. Kemperman, and E. A. Trabka, *Alternating renewal model of photographic granularity*, Journal of the Optical Society of America, 63 (1973), pp. 820–825. http://dx.doi.org/10.1364/josa.63.000820.

[4] S. N. Chiu, Dietrich Stoyan, W. S. Kendall, and J. Mecke, *Stochastic geometry and its applications*, John Wiley & Sons, third ed., 2013.

[5] B. Galerne, Y. Gousseau, and J.-M. Morel, *Random phase textures: Theory and synthesis*, IEEE Transactions on Image Processing, 20 (2011), pp. 257 – 267. http://dx.doi.org/10.1109/TIP.2010.2052822.

[6] B. Galerne, A. Leclaire, and L. Moisan, *Texton noise*, Tech. Report 2016-09, MAP5, 2016. http://www.math-info.univ-paris5.fr/~bgalerne/texton_noise/.

[7] R.W. Gurney, *The theory of the photolysis of silver bromide and the photographic latent image*, Proceedings of the Royal Society of London, 164 (1938), pp. 151–167. http://dx.doi.org/10.1098/rspa.1938.0011.

[8] D. J. Heeger and J. R. Bergen, *Pyramid-based texture analysis/synthesis*, in Proceedings of the 22nd annual conference on Computer Graphics and Interactive Techniques, 1995, pp. 229–238. http://dx.doi.org/10.1145/218380.218446.

[9] A. Lagae, S. Lefebvre, G. Drettakis, and P. Dutré, *Procedural noise using sparse Gabor convolution*, SIGGRAPH '09, 28 (2009). https://doi.org/10.1145/1576246.1531360.

[10] W. H. Lawton, E. A. Trabka, and D. R. Wilder, *Crowded Emulsions: Granularity Theory for Multilayers*, Journal of the Optical Society of America, 62 (1972), pp. 659–667. http://dx.doi.org/10.1364/josa.62.000659.

[11] R. Livingston, *The Theory of the Photographic Process. By C. E. Kenneth Mees*, Journal of Physical Chemistry, 49 (1945), p. 509. http://dx.doi.org/10.1021/j150443a017.

[12] J.-M. Morel and G Yu, *Is sift scale invariant?*, Inverse Problems and Imaging, 5 (2011), pp. 115–136.

[13] A Newson, B. Galerne, and J. Delon, *Stochastic modelling and realistic rendering of film grain*, tech. report, Laboratoire MAP5, Université Paris Descartes, France, 2016.

[14] P. G. Nutting, *On the absorption of light in heterogeneous media*, Philosophical Magazine, 26 (1913), pp. 423–426. http://dx.doi.org/10.1080/14786441308634988.

[15] B. T. Oh, S.-M. Lei, and C.-C. Kuo, *Advanced Film Grain Noise Extraction and Synthesis for High-Definition Video Coding*, IEEE Transactions on Circuits and Systems for Video Technology, 19 (2009), pp. 1717–1729. http://dx.doi.org/10.1109/tcsvt.2009.2026974.

[16] P. Schallauer and R. Mörzinger, *Film grain synthesis and its application to re-graining*, in Proceedings of the SPIE, vol. 6059, 2006, pp. 60590Z–60590Z–7. http://dx.doi.org/10.1117/12.650694.

[17] I. Stephenson and A. Saunders, *Simulating film grain using the noise-power spectrum*, in Eurographics UK Theory and Practice of Computer Graphics, 2007. http://dx.doi.org/10.2312/LocalChapterEvents/TPCG/TPCG07/069-072.

[18] Stoyan, *Stochastic geometry and its applications*, vol. 2, Wiley New York, 1987.

[19] K. Tanaka and S. Uchida, *Extended random-dot model*, Journal of the Optical Society of America, 73 (1983), pp. 1312–1319. http://dx.doi.org/10.1364/josa.73.001312.

[20] G. Wernicke, *Silver-Halide Recording Materials for Holography and Their Processing*, Zeitschrift für Physikalische Chemie, 187 (1994), pp. 322–323. http://dx.doi.org/10.1524/zpch.1994.187.part_2.322.

[21] S. Worley, *A cellular texture basis function*, in SIGGRAPH '96, ACM, 1996, pp. 291–294. https://doi.org/10.1145/237170.237267.

[22] J. C. K. Yan, *Statistical methods for film grain noise removal and generation*, master's thesis, University of Toronto, 1997.