Studying and solving visual artifacts occurring when procedural texturing with paradoxical requirements

Vincent Tavernier

Grenoble INP - Ensimag

École nationale supérieure d'informatique et de mathématiques appliquées

3rd year - Specialization ISI

Studying and solving visual artifacts occurring when procedural texturing with paradoxical requirements

Vincent TAVERNIER

June 21st, 2017

Masters research project performed at

MAVERICK Team – Inria/LJK

Institut National de Recherche en Informatique et en Automatique
Laboratoire Jean Kuntzmann

Under the supervision of

Fabrice NEYRET
Joelle THOLLOT
Romain VERGNE

Defended before a jury composed of

James Crowley, President
Dominique Vaufreydaz
Renaud Blanch
Franck Hétroy-Wheeler

MAVERICK Team – Inria/LJK
655 Avenue de l'Europe
38330 Montbonnot-Saint-Martin

Contents

List of Figures

Abstract

Textures are images widely used by computer graphics artists to add visual detail to their work. Textures may come from different sources, such as pictures of real-world surfaces, manually created images using graphics editors, or algorithmic processes. "Procedural texturing" refers to the creation of textures using algorithmic processes.

Procedural textures offer many advantages, including the ability to manipulate their appearance through parameters. Many applications rely on changing those parameters to evolve the look of those textures over time or space. This often introduces requirements contradictory with the structure of the unaltered texture, often resulting in visible rendering artifacts. As an example, to animate a lava flow the rendered texture should be an effective representation of the simulated flow, but features such as rocks floating over should not be distorted, nor brutally appear or disappear thus disrupting the illusion. Informally, we want our lava texture to "change, but stay the same". This example is an instance of the consistency problem that arises when changing parameters of a texture, resulting in noticeable artifacts in the rendered result.

In this project, we seek to classify these artifacts depending on their causes and their effects on textures, but also how we can objectively detect and explain their presence, and so predict their occurrence. Analytical and statistical analysis of procedural texturing processes will be performed, in order to find the relation with the corresponding artifacts.

The core contribution lies on the search for objective measures and characterizations of texture artifacts, independently of the texturing process. The resulting method should ideally be process-independent, in order to provide a generic framework for building better-looking animated procedural textures through the true comprehension of rendering artifacts.

Résumé

Les textures sont des images largement utilisées par les artistes infographistes pour ajouter des détails dans leurs rendus. Ces textures peuvent provenir de différentes sources, telles que des images de surfaces réelles, des images créées manuellement dans des logiciels, ou des processus algorithmiques. Ces dernières sont appelées "textures procédurales".

Les textures procédurales offrent beaucoup d'avantages, notamment la possibilité de définir leur apparence à partir de paramètres. Dans de nombreuses situations, ces paramètres évoluent au cours du temps ou dans l'espace. Cela introduit cependant des contraintes contradictoires à l'apparence de la texture originale, introduisant ainsi des artefacts visibles. Par exemple, pour animer un flot de lave, la texture rendue devrait suivre le flot simulé, mais les formes telles que des rochers flottants à la surface ne devraient pas être distordus, ni apparaître ou disparaître brutalement, ce qui casserait l'illusion. Informellement, cette texture de lave doit "changer, mais rester la même". Cet exemple est une instance du problème de la cohérence temporelle lié à l'évolution de paramètres d'une texture, ce qui introduit des artefacts dans le rendu final.

Au cours de ce projet, nous essayons de classer les différents artefacts selon les causes qui les produisent et leurs effets sur les textures. Nous cherchons aussi des méthodes pour décrire objectivement et détecter leur présence, et même prédire leur apparition. Des méthodes analytiques mais aussi statistiques des procédés de texture vont être mis en œuvre, afin de trouver les liens entre artefacts et descripteurs.

Notre principale contribution repose sur la recherche de mesures objectives ainsi que la caractérisation d'artefacts de texture, indépendamment du procédé de génération. La méthode qui en découle devrait idéalement être indépendante du procédé, afin de poser les bases d'une méthodologie pour construire de meilleures textures procédurales, grâce à la compréhension des artefacts.

Acknowledgements

I would first like to thank my internship advisors, Fabrice Neyret, for his valuable insight on procedural texturing and comprehension of artifacts, Joelle Thollot, who never failed keeping this project on schedule, while sharing her knowledge about non-photorealistic rendering, and Romain Vergne, who successfully steered my research about perception the right way while proofreading this work.

I would also like to thank the entire MAVERICK team for their warm welcome, as I felt integrated since the very first days, and it would be a pleasure to keep working along them, maybe for a PhD.

I would like to thank Inria and the Laboratoire Jean Kuntzmann for providing me with the necessary resources to carry out this work.

Finally, I would like to thank James Crowley for allowing me to defend this research project alongside other students of the MoSIG master, even though I am following an ISI specialization at Ensimag.

– *Vincent Tavernier*

1 Introduction

1.1 Introduction to texturing

Images can have various forms depending on the data they contain. They can have a certain number of dimensions, depending on how many coordinates are needed to designate a pixel. 2D images are the most common, but 1D images are sometimes used. 3D and higher have uses in specific rendering techniques such as solid texturing.

A texture is a n-dimensional image used to add controllable details to computer graphics renderings in various applications, such as texture mapping (fig. 1.2a), 2D area visualizations (fig. 1.2b), fluid simulation renderings (fig. 1.2c), compositing (fig. 1.2d), solid rendering (fig. 1.2e) and material rendering (fig. 1.2f).

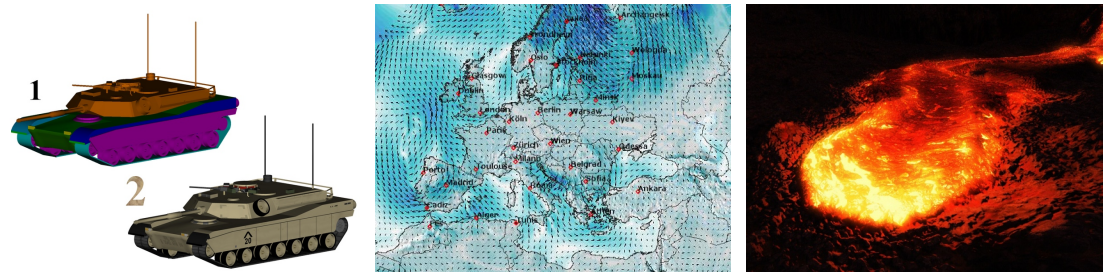
Except for texture mapping applications, textures usually exhibit a self-similar structure, giving them a homogeneous look. This property can be used to texture surfaces larger than the texture itself through tiling, among other means. fig. 1.1 illustrates this property with different wall textures.

These textures may come from different sources, depending on the requirements:

- Graphics editor, either based on photographs or created from scratch. This can be used to create tileable textures or mapped textures.
- Algorithmic processes, based on procedural noise primitives (Perlin, Worley, Gabor), and described in terms of shading programs, such as shadetrees.

(a) Concrete **(b)** Bricks **(c)** Damaged wall

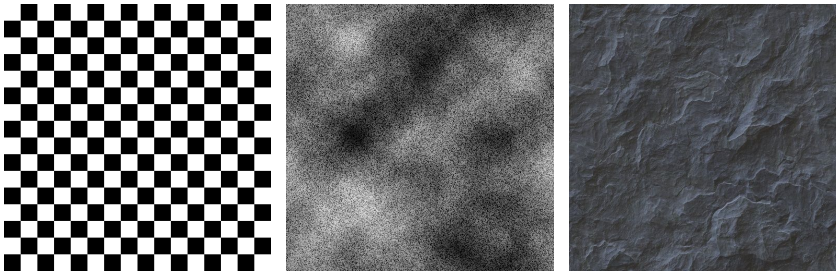
Figure 1.1: Differences between textures and images. Images (a) and (b) qualify as textures because of their homogeneous structure, while (c) does not.



(a) Mapped texture **(b)** 2D visualization **(c)** Fluid simulation

(d) Compositing **(e)** Solid rendering **(f)** Rendered material

Figure 1.2: Applications of textures



(a) Regular **(b)** Stochastic **(c)** Rough

Figure 1.3: Regular texture (a) vs. stochastic texture (b). Rough material rendered using a stochastic texture as a randomness source (c).

	Procedural	Image
Resolution	Infinite	Fixed: $N \times M$ pixels
Memory	Negligible	$N \times M \times size\,(pixel)$
Parameters	Few	Many during edition, none during usage

Table 1.1: Summary of procedural texture advantages over image-based textures

Procedurally generated textures exhibit many advantages over simple images, as summarized by table 1.1. The ability to render a texture at any resolution and over any surface size considerably reduces the amount of work required to author a texture: instead of specifying the value of each pixel for all resolutions and over all the texture space, the artist only has to choose the right parameters for the algorithm to produce the desired appearance. The resulting procedural texture can be rendered on-the-fly at the needed resolution and size. This allows adding very fine details to textures, a critical part of realistic rendering.

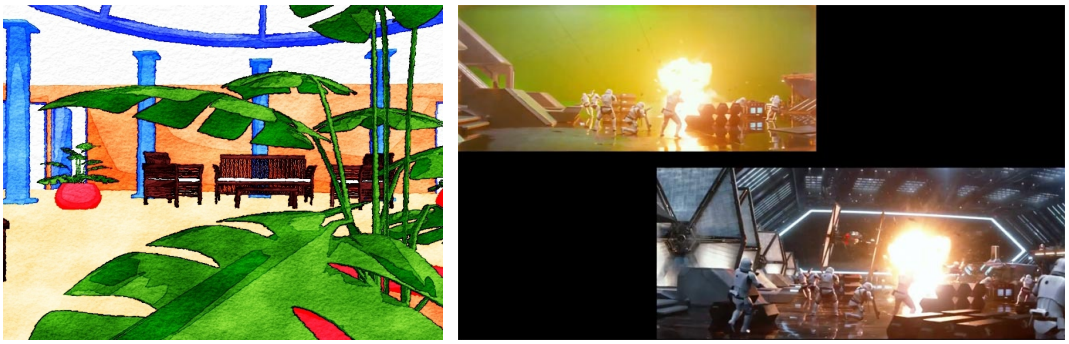
The processes used when generating textures may produce images ranging from highly regular images (fig. 1.3a), to images with no apparent regularity (ie. stochastic textures, fig. 1.3b). Stochastic textures are often used as a coherent source of randomness, for example to add roughness to a material surface, as in fig. 1.3c. Moreover, many procedural noise primitives exist to generate stochastic textures exhibiting the required characteristics.

We are interested here in procedurally generated stochastic textures, because of their statistical properties that make them suitable for a number of applications, as well as their widespread use as building blocks for more complex textures.

Complex textures can be created from procedural primitives using many different methods. These methods may involve compositing textures together, a process driven by the artist describing how layers of a textures should be combined. These combinations are given in terms of "blending modes", which specify the mathematical operations to be performed to compute the resulting values. Examples of blending modes include additive, subtractive and multiplicative, which match the basic mathematical operations, although more complex modes may be used to introduce specific effects, such as the "screen" blending mode[1]. Some of the major uses for compositing include non-photorealistic rendering and various video special effects (see fig. 1.4).

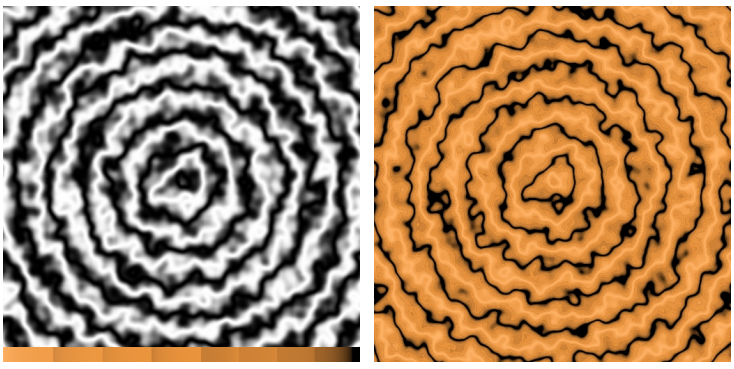
Another method for creating complex textures from procedural primitives involves adding more parameters to the base algorithm behind the primitive. This is a sort of refinement of the base texture, giving more control to the artist but potentially greatly increasing the complexity of the tuning phase at authoring time. For example, if a step in the texturing algorithm uses a simple function to compute pixel values, allowing the artist to change the definition of this function would result in more freedom. In fact, this is what happens when the output of a texture is transformed using a LUT (look-up table), or "color map". A color map defines a mapping (ie. a function) which converts intensity values into colors. Choosing these LUTs is usually referred to as "sculpting" the material, the way a sculptor carves the base material to add detail to his work. fig. 1.5 shows how a LUT can be used to create a procedural wood texture.

[1] The screen blending mode is defined as being the result of "projecting" the layers onto a screen, as it would look on a traditional diapositive projector.



(a) Non-photorealistic rendering **(b)** Video special effects

Figure 1.4: Uses of blending modes. In non-photorealistic rendering (a), colors can be tuned depending on a pigment density to achieve watercolor effects. This tuning is effectively a blending mode. Source: [Bou+06]. In video special effects (b), artists use blending modes to compose traditional footage (top) and digital footage together to form the final image (bottom) (image from Star Wars: The Force Awakens).



(a) Circular noise texture and LUT **(b)** Resulting wood texture

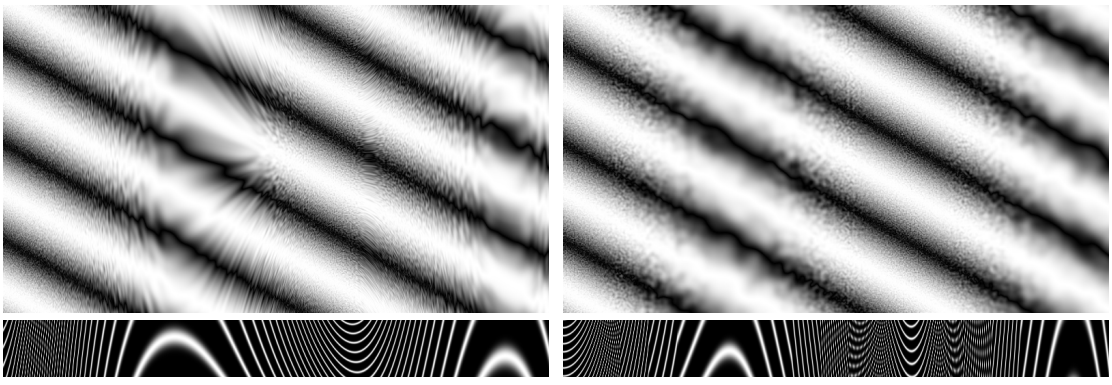
Figure 1.5: Use of a LUT in a wood texture. The value of a procedural noise (a) is used to look up a color in the LUT texture (a, inset), in order to produce a wood texture (b). We can see the discontinuities in the LUT introduce detail in the resulting texture.

Although blending and parameter evolutions seem straightforward, trivial techniques used to implement these methods often result in unwanted results in renderings, called artifacts. They are hard to remove by nature, as they come from the procedural texturing process itself, over which an artist has no control except for the input parameters.

1.2 Visual artifacts

We can distinguish two main classes of artifacts. The first class regroups artifacts that arise from continuously varying parameters of a procedural texture using continuous functions. The second class involves artifacts resulting from blending multiple texture layers.

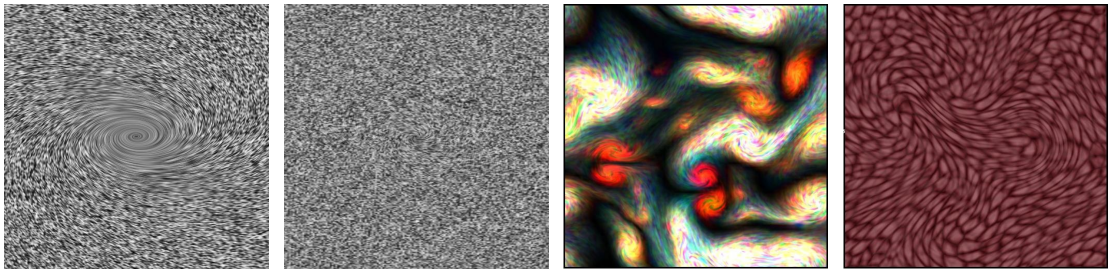
The appearance of a procedural texture is defined by the choice of its input parameters. In order to offer finer control to the artist, these parameters can be set up to vary over the texture space or over time. As discontinuities in textures are usually noticeable, parameters are often chosen to be continuous functions of their input (texture coordinates, time, etc.). However, this continuity is contradictory with the requirements for the texture to keep its original appearance. In fig. 1.6, the scale of a marble pattern changes depending on the texture coordinates. Stretching artifacts occur because of the noise scaling parameter continuity with respect to the texture coordinates. Introducing discontinuities in the scale parameter relation to the texture coordinates helps resolve the problem, at the cost of lower local contrast.



(a) Continuous scaling **(b)** Non-continuous scaling

Figure 1.6: Stretching artifact in a marble texture. Note how contours appear near scale changes in the continuous scaling version (left), while introduced discontinuities remove this artifact (right). Bottom images show contour lines of the function of texture coordinates that drives the noise scale. Source: http://bit.ly/st-MlGGW1

Issues with continuous dependence of procedural texture parameters on other continuous quantities is also a source of stretching in naive texture advection schemes, as noted by Neyret [Ney03]. Texture advection is a process of distorting a texture in order to represent a flow, usually a result of a physically-based fluid simulation. However, an artist rendering a fluid animation would want to keep the local appearance of the texture, which is contradictory with the stretching imposed by the flow (see fig. 1.7). In order to reduce stretching, while keeping the motion induced by the flow, over-stretched textures can be progressively replaced with non-stretched textures.



(a) Advected noise **(b)** Advected textures

Figure 1.7: Advected textures. Naive advection results in distortion, while stretched layer replacement does not (a). Stretching can be controlled to achieve desired results on specific textures (b). Source: [Ney03]

However, such texture advection schemes require blending, which usually results in a loss of contrast [Yu+11] or in ghosting artifacts (see figure 1.8). These artifacts can be found on almost all complex textures involving multiple layers, as blending is one of the basic primitives for building procedural textures. Blending modes (ie. functions combining pixel values between layers) have a simple mathematical description, which enable a statistical modeling of blending artifacts.

(a) Loss of contrast **(b)** Ghosting

Figure 1.8: Typical blending artifacts. Texture with artifact (left) without artifact (right). Sources: (a) [Yu+11] (b) [Ney03]

Mathematical modeling of texturing artifacts is thus required to make the necessary changes in rendering algorithms. Different modeling strategies are suited to different artifacts although the methodology is the same. Some artifacts are

also more visible than others, which may play a role when no perfect solution is available.

1.3 Contributions

Texturing artifacts vary in nature and perceivability, therefore we chose in this work to focus on the search for objective and perceptually sound measures of artifacts that affect texture properties.

We chose to do this through a canonical case of a procedural texture with paradoxical requirements: a stationary white noise zooming texture. The well defined properties of white noise enable us to relate visible artifacts to mathematical descriptors, and provide solutions to those artifacts, when applicable. These solutions will be evaluated relatively to the reference texture (a white noise) in order to validate them and ensure they do not create more problems than they solve.

We also discuss the perception of these artifacts, as their impact can vary greatly, some of which may completely change the resulting texture, while others may be almost invisible. The human perceptual system will thus be taken into account, in order to determine if the chosen descriptors are relevant.

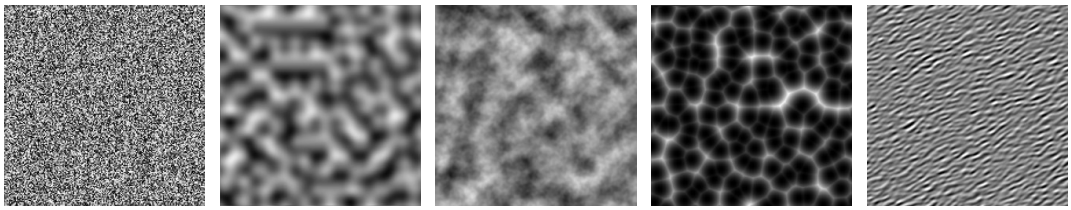
This basic procedural texturing case is interesting in the sense that the initial choice of white noise does not restrict the scope of this study. Indeed, more complex noises can be decomposed into simpler components through transforms such as Fourier analysis. Such components can then be analyzed following the same methodology as introduced in this work.

1.4 Related work

As this work focuses on procedural texturing using stochastic noise algorithms, we are interested in procedural noise primitives and their properties, as well as known artifacts resulting from procedural texturing. This includes how these artifacts can be described and fixed, but also how observers perceive them.

1.4.1 Procedural noises and stochastic textures

Procedural texturing heavily relies on stochastic procedural primitives, but one needs to apply specific techniques to create noises with the required properties, such as controlled spectrum, specific histogram, anisotropy, etc. Indeed, pseudo-random generators can be used to generate Gaussian white noise, but its direct usage for texturing is rather limited: its power spectrum and histograms can only be flat, and it is isotropic, thus offering poor control potential to the artist. It can



(a) White noise **(b)** Perlin noise **(c)** Fractal Perlin noise **(d)** Cell noise **(e)** Gabor noise

Figure 1.9: Common procedural noise primitives

however be used to build more complex stochastic processes, called procedural noises or noise primitives. Refer to fig. 1.9 for illustrations of noise primitives mentioned in the next part.

Perlin [Per85] introduces a noise function suitable for texture modeling because of its statistical invariance and controlled frequency bandwidth. In this original work, the noise function is used to tune the colors and normals of 3D objects. The stochastic yet controllable aspect of this "Perlin noise" gives a random touch to the renderings, over which the artist still has control. Perlin also introduces fractalized noise, which is a combination of Perlin noise layers at different scales. The resulting texture has a larger but still limited frequency bandwidth. Many improvements have been made to Perlin noise, in order to reduce aliasing and detail loss, as in Wavelet noise [CD05].

Another noise primitive was introduced by Worley [Wor96], either called "cell noise" or "Voronoise", due to its resemblance to Voronoi diagrams. Indeed, this primitive describes a texture which is controlled by the placement of seed points, the noise value being given by the distance to the closest point. Applying a color map to the output of this primitive allows creating a wide range of materials that exhibit such stochastic cellular patterns.

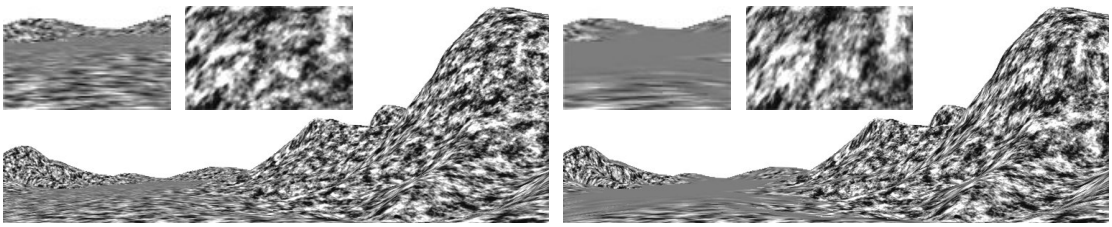
However, these noise primitives offer very little control, thus making the design of noise patterns a hard task for the artist. This problem is partially solved by Lagae et al. [Lag+09], who introduce a procedural noise primitive based on sparse Gabor convolution. The artist is responsible for defining the frequency spectrum of the texture, which is then convolved with a Gabor kernel in order to synthesize the resulting texture. Although this method offers full spectral control of the texture, the frequency spectrum is often seen as unintuitive as a design tool. This issue is addressed by Galerne et al. [Gal+12] by deriving the frequency spectrum from an example texture to reproduce.

Noise primitives other than white noise offer greater expressive potential for artists seeking to add controllable randomness to their work. Though their mathematical properties are harder to describe than in the white noise case, they can still be decomposed into simpler components whose properties are similar to white noise

instances. Any work done based on white noise properties should be applicable to these components, and further combinations of those components.

1.4.2 Texturing artifacts fixing methodology

One of the most studied artifacts in computer graphics is aliasing. The similarities between signal processing and digital imaging provides a solid mathematical framework for the study of this class of artifacts. Mitchell and Netravali [MN88] propose a thorough analysis of aliasing artifacts, from the different sources of aliasing to the properties of reconstruction filters for fixing those artifacts. The different filters are compared in a user study in order to find the best compromise between aliasing and blurriness, to preserve the sharpness of the original image. Mitchell and Netravali also note that due to their sampled nature, procedural texturing algorithms are easily subject to aliasing when the signal is reconstructed, as non band-limited noise primitives will often exceed the Nyquist frequency. The problem of stochastic texturing aliasing has been more specifically studied by Goldberg, Zwicker, and Durand [GZD08], in the context of texture mapping. They propose a filtering method based on a Fourier decomposition of the texture, to adapt the filter anisotropy to the texture actual display transformation (see fig. 1.10).



(a) Anisotropic noise

(b) Isotropic filtering, no distortion compensation

Figure 1.10: Anisotropic noise filtering avoids detail loss artifacts introduced in isotropic filtering, and also allows for distortion compensation [GZD08].

Other artifacts can be described without considering frequency spectrums. Indeed, texture properties can be described using common statistical descriptors, such as the mean, representing the image luminance, and the variance, representing the image contrast. Those descriptors can prove the presence of artifacts, such as a texture being too bright or too dim, or a change in contrast that drastically alters the look of the resulting texture. These descriptor values may also be used to fix the corresponding artifacts, as noted by Yu et al. [Yu+11]. In this work, blending is used to advect a texture along a flow, ie. distorting texture patches in order to match the optical flow of the resulting texture with the input flow. Blending multiple texture patches together keeps the original texture mean, as long as the blending weights sum up to 1, although the contrast is lowered. Dividing the

resulting pixel intensity values by their standard deviation restores the contrast of the original texture. See figure 1.11 for an example of this method.

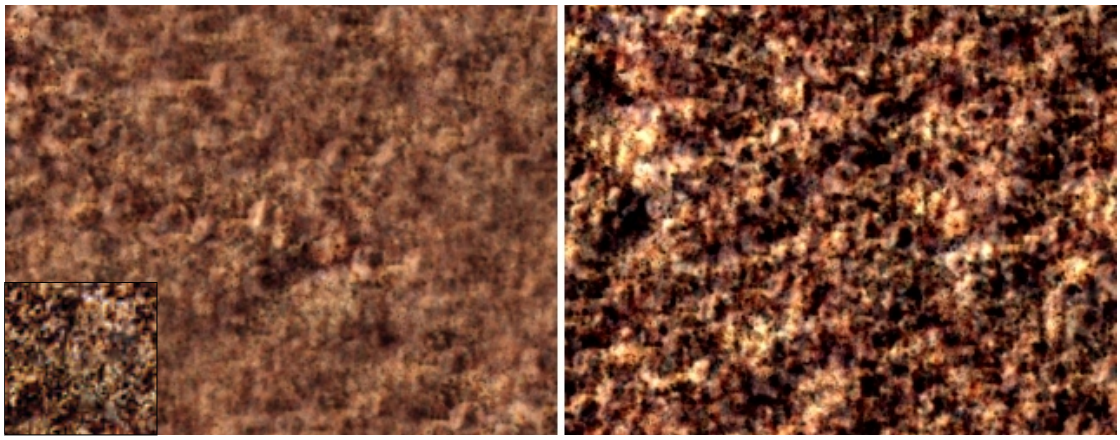


Figure 1.11: (Left) Naive sprite blending (Right) Contrast-corrected blending (Inset) Original sprite. Source: http://bit.ly/st-4dcSDr

However, blending artifacts are present in situations other than texture advection, for example in the tri-planar texture mapping process. This technique is used when rendering complex objects with non-trivial texture mapping, such as uneven terrain, trees, rocks, etc. Instead of defining an explicit texture mapping, three textures are projected onto the object from three different axes, and blended together depending on the orientation of the current rendering point. A point facing the X axis will only be rendered using the texture for the X axis, while a point forming 45° angles with all three axes will be rendered using all three textures, with 1/3 blending weights. Ghosting and contrast loss artifacts are common (see fig. 1.12), but artists usually find a way around these artifacts, either by:

- Editing the geometry so there is no large surfaces forming 45° angles with primary axes.
- Changing the blending weights in order to reduce the surface space suffering from blending artifacts, at the cost of higher distortion.
- Adding more textures at 45° angles to reduce blended areas.
- Extending textures to use depth-map blending, a process where the "depth" of the texture is used to determine which blended texture is in front of the other. Smoothing is required to prevent depth-fighting when moving the surface.

Artifacts have received varying levels of attention, from mathematically modeled aliasing filters to unsolved issues in texture mapping processes. Artists thus spend a lot of time fixing artifacts inherent to procedural texturing algorithms, as without a modified algorithm they cannot do more than hide the artifacts away from the view point. In this work we try to refine the methodology for the study of

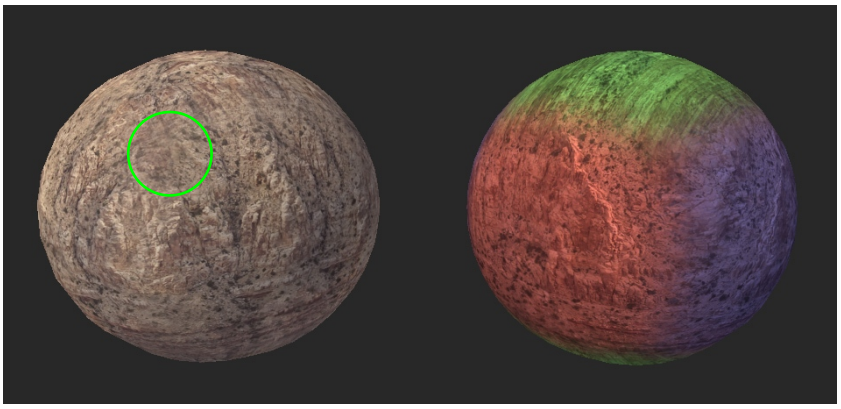


Figure 1.12: Tri-planar mapping on a sphere. The same texture is used for all three axes (red, green and blue on the right), but we can see the contrast being lower at 45° angles (green circle). Source: http://bit.ly/triplanar

blending and continuous parameter evolution artifacts, in the context of stochastic procedural texturing.

1.4.3 Perception of texturing artifacts

Although artifacts may be described objectively, their perception can vary greatly depending on the rendering situation. Indeed, the human perceptual system, being highly evolved, is able to infer a lot of high-order information, such as contour, shape, movement and so on. These higher-order properties are studied by Bénard, Bousseau, and Thollot [BBT11] when comparing the different solutions to the temporal consistency problem in stylized animations. Criteria for the comparison include the presence of common artifacts:

- Popping: the sudden appearance or disappearance of texture features.
- Regeneration issues: texture advection time regeneration periods causing continuity issues.
- Flickering: texture features rapidly blinking because of a condition being near of its limit value.
- Secondary motion: perceptible motion introduced by texturing that was not present before, and usually not wanted.
- Shower door and sliding: when the texture and the scene movements do not match in the resulting image, texture features can be perceived as "sliding over" the underlying scene.

It is interesting to note that most of the methods presented in this work involve some trade-offs, mainly due to the contradictory requirements introduced by non-photorealistic rendering of animations. However, these trade-offs may be acceptable depending on the use case, most notably because of visual masking.

Visual masking occurs when a visual pattern is made perceptually invisible because of interferences with another pattern. Ferwerda et al. [Fer+97] introduce a computational model of visual masking for the human perceptual model using psychophysical data. This work also presents an application of this model to predict the visibility of a faceting artifact depending on the visual properties of a noise texture (see fig. 1.13). We can see that the contrast, frequency spectrum and orientation of the texture clearly hides the underlying faceting artifact.

The impact of the human perceptual system has also been studied in the context of fractalized textures, a useful primitive in the design of non-photorealistic rendering algorithms. Bénard, Thollot, and Sillion [BTS09] present a ranking of common procedural textures depending on their ability to be fractalized while maintaining their original appearance. This ranking is based on a psychophysical user study, meaning the introduced "distortion score" is based on how the textures are perceived. Highly-regular textures such as checker boards, grids, etc. greatly suffer from a fractalization process, while stochastic textures are perceived as very similar to their non-fractalized versions (see fig. 1.14). Users also reported changes in contrast, sharpness and scale of textures with a poor fractalization potential.

Deficiencies in the human perceptual system have been known for years, and have been exploited in the design of lossy image compression algorithms [Sal+15]. By removing information which cannot be perceived by the human perceptual system (known as "psycho visual redundancies" in images), one is able to compress images without changing their appearance in a noticeable way. The same redundancies, with a supplementary time dimension, are exploited in lossy video compression algorithms.

Although some artifacts have clearly defined solutions, ie. introducing the fix in the algorithm completely removes the artifacts without any side effects, others may involve a trade-off with other texture properties. For example, restoring contrast in a texture may perceptually change its appearance. Having a perceptual insight on the presence of texturing artifacts may help chose the best compromise, if applicable.

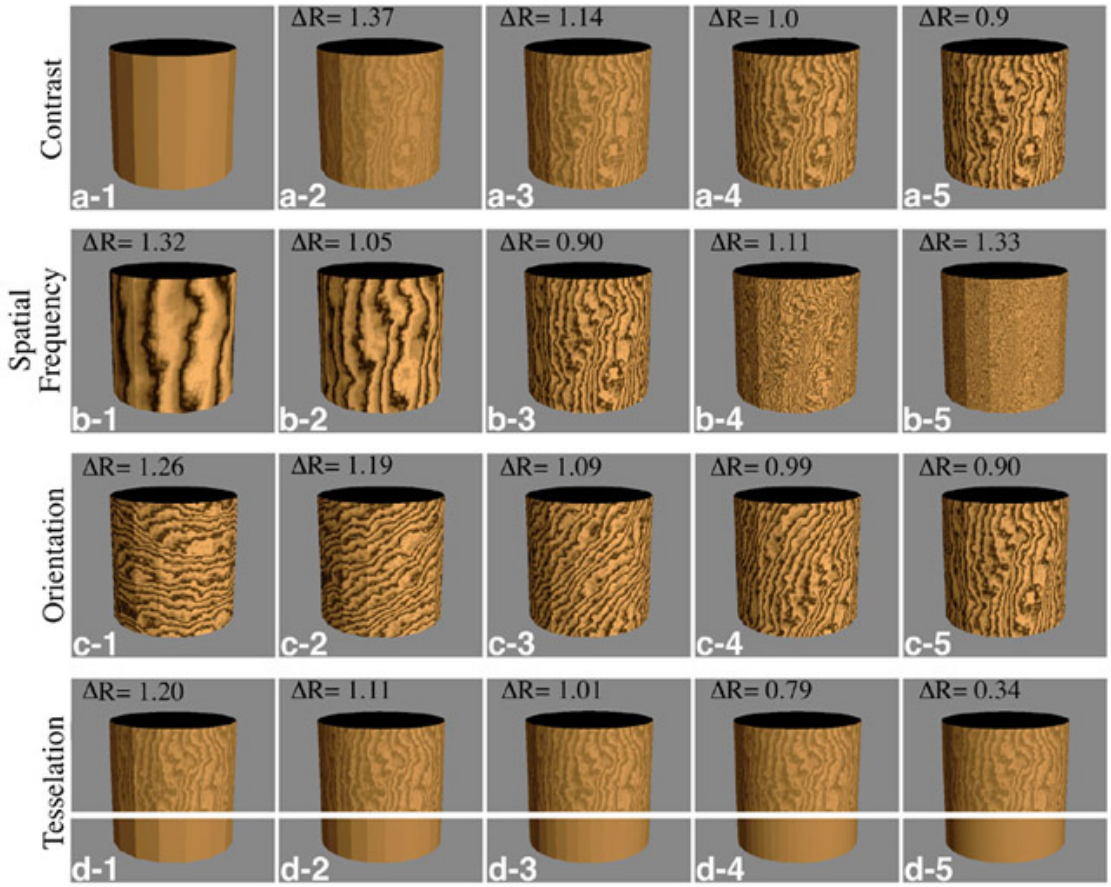


Figure 1.13: Psychophysical model of visual masking of a faceting artifact [Fer+97]. Values of ΔR lower than 1.0 as computed by the model indicate the image is not visibly different from the reference image.

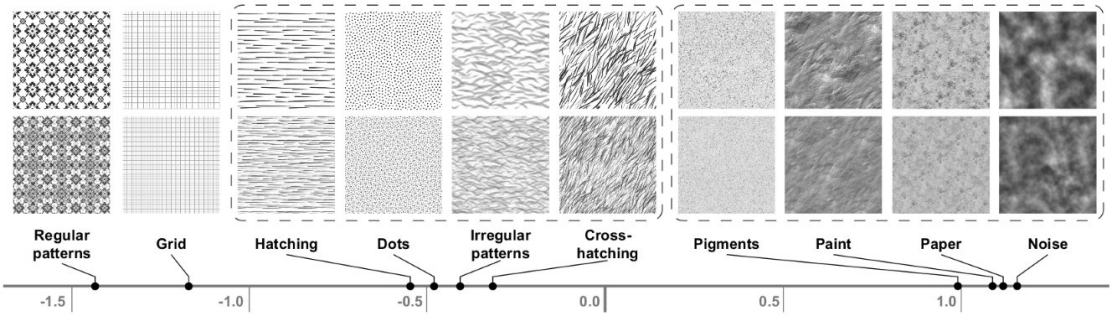


Figure 1.14: Results of the user study from [BTS09] (extract). Texture pairs comprising the original texture and the fractalized version, placed on a scale derived from the results.

2 Studying a stationary zooming texture

In the following parts, we will first introduce the stationary zooming texture used as a basis for our study of texturing artifacts. Then, we will focus on describing and solving visible artifacts. We then conclude by discussing the perception of such artifacts and the possible extensions of this work.

2.1 Building a stationary zooming texture

We are interested in building a procedural texture which gives the user a zooming-in feeling, while keeping the same appearance through the whole evolution process. Evolving this zooming factor over time should result in a sort of "infinite zoom" effect. We will use white noise as the base texture in this case because of its simple statistical description. Note that the resulting texture can be used to produce more complex patterns later in the rendering pipeline, thus not restricting the scope of this kind of texture evolution. Such textures can then be used in non-photorealistic rendering as a base for creating dynamic canvas that adapt to the view transformations.

The base method for scaling a texture is to scale its coordinates. If g is a texture function and $\mathbf{x}$ a point in texture space, varying $a \in \mathbb{R}$ while accessing $g(a\mathbf{x})$ changes the apparent scale of the texture. However, continuously evolving a over time does not maintain the spatial appearance of the base texture. A basic solution is to introduce a modulo operation to reduce the range of the scaling parameter, although the discontinuity of the modulo function directly translates to a jump in scale. Varying the allowed scale range only changes the frequency of those jumps as well as the distortion compared to the original texture, but does not change the intensity of this artifact. See `http://bit.ly/st-MsXyDl` for a visualization.

Noticing that this scaled texture is only "usable" for a short time period indicates that we could use a sequence of temporarily "usable" textures, replacing over-stretched textures with correct ones. Indeed, we can introduce multiple layers, each being continuously scaled over time, using a scaling factor close to 1 in order to match the original texture appearance. This process is analogous to Shepard [She64] tones, which are sounds consisting of multiple base sounds continuously rising in pitch, such as sine waves, but that give the auditory illusion the resulting pitch is rising indefinitely, although it is in fact stationary.

Additive blending is a viable option due to the nature of the contribution of each layer to the final result. This multi-layer zoom texture exhibits time discontinuities corresponding to the replacement of stretched layers by non- stretched ones. Introducing weights at blending time fixes this artifact, by lowering the contribution of appearing (resp. disappearing) texture layers progressively, so there is no noticeable jumps in texture appearance. Any smooth function can be used here, as long as its value and first derivative are 0-valued near the texture layer start (and end) of life. A Gaussian-like function with suitable center and width, function of the scaling factor a is a good fit (see fig. 2.1c). Choosing the weights so their sum over all texture layers equal 1 allows keeping the original texture luminance. This method is used in non-photorealistic rendering, in order to generate fractalized textures which retain their local appearance even though they undergo geometrical transforms to match the scene motions Cunzi et al. [Cun+03].

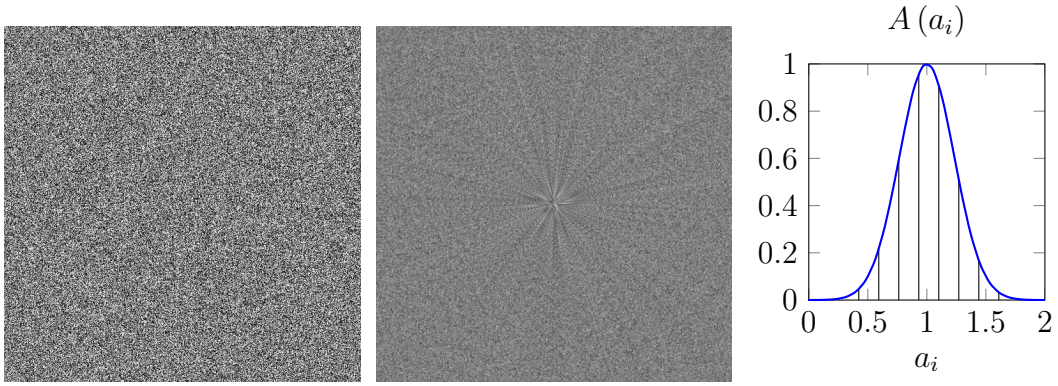
This process can be expressed as in eq. (2.1). All the notations are defined in table 2.1.

$$\begin{aligned} I(\mathbf{x}) &= \sum_{i=0}^{n-1} A(a_i)\, g(a_i \mathbf{x}) \\ \text{with } A(a_i) &= \frac{e^{\frac{(a_i-1)^2}{2\sigma^2}}}{\sum_{j=0}^{n-1} e^{\frac{(a_j-1)^2}{2\sigma^2}}} \\ \text{and } a_i &= \operatorname{mod}\left(i\frac{H}{n} - t, H\right) \end{aligned} \tag{2.1}$$

Symbol	Description
$I(\mathbf{x}) \in [0,1]$	Resulting pixel intensity
$\mathbf{x} \in \mathbb{R}^2$	Point in the texture space, in pixel coordinates
$n \in \mathbb{N}$	Number of noise layers being combined. Defaults to 8.
$H \in \mathbb{R}$	Maximum scale value. Defaults to 2.
$t \in \mathbb{R}$	Current time
$(a_i)_{i \in [0,n[}$	Scaling coefficients for each layer
$A : \mathbb{R} \to \mathbb{R}$	Layer amplitude envelope function Property: $\sum_{i=0}^{n-1} A(a_i) = 1$
$g : \mathbb{R}^2 \to \mathbb{R}$	Texture function. In fig. 2.1a, it is a linearly interpolated white noise grid generated at the beginning of the rendering.

Table 2.1: Notations used in the white noise zoom texture description

While this process effectively removes discontinuities and luminance variations, the resulting texture (fig. 2.1) exhibits various artifacts. These artifacts come



(a) $g(\mathbf{u})$ **(b)** $I(\mathbf{u})$ **(c)** Envelope A

Figure 2.1: (a) White noise texture (b) Zooming in effect on a white noise texture using 16 layers (http://bit.ly/st-lssfW8) (c) Intensity envelope function used in eq. (2.1). a_i is the scaling factor

from different properties of the rendering algorithm, and thus require different mathematical tools to be described.

One of the texturing artifacts visible in this texture is the sliding artifact (fig. 2.2). It is characterized by the perception of several different overlapped motions, giving the impression that texture features *slide* over each other. This artifact is stronger further away from the texture center, and breaks the zooming in illusion.

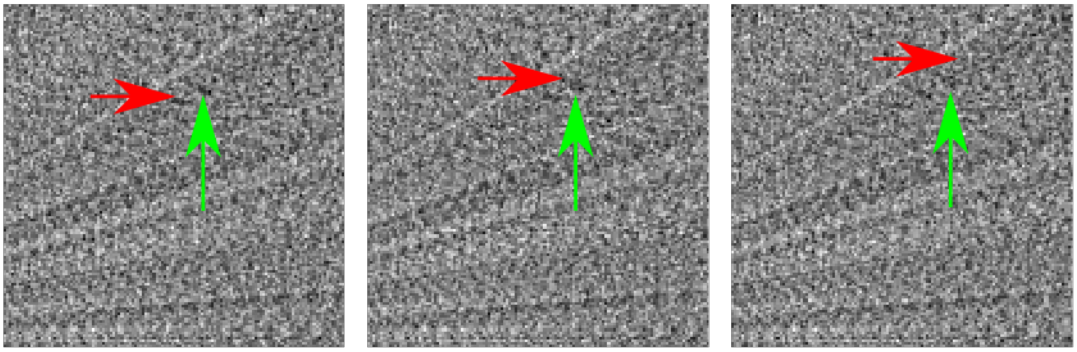
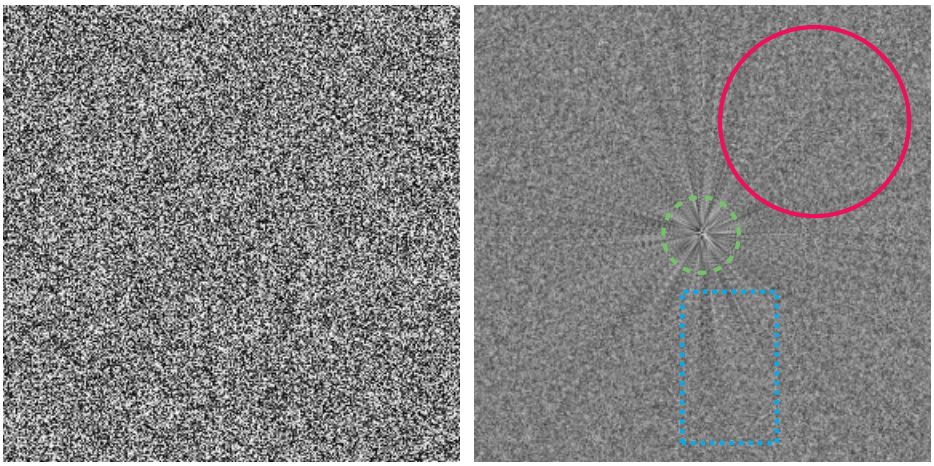


Figure 2.2: Sliding artifact on white noise. The dark spot (green arrow) appears to move slower than the light spot (red arrow), giving an impression of sliding. Contrast has been reinforced for better visibility. Effect is easier to see on animated version: http://bit.ly/st-lssfW8

The second class of artifacts visible in this texture is related to the statistical properties of the image. Compared to the white noise texture, the zooming texture exhibits a noticeable contrast loss, as well as new features that have been

introduced. These features can easily be distinguished from the surrounding noise: echoes, fixed fetchers, darker and lighter areas. See fig. 2.3 for details.



(a) White noise texture **(b)** Resulting texture

Figure 2.3: Statistical artifacts on white noise. The contrast loss results in more gray tones than in the original white noise (left). Other statistical artifacts include echoes or repeating features (red, solid), still features (green, dashed), darker and lighter areas (blue, dotted).

As these artifacts come from different properties of the rendering algorithm, they will be studied in the following parts, using suitable methods. The statistical model will be used for both contrast and feature-based artifacts, and thus will be introduced only once.

Note: non-normalized pixel coordinates will be used in the following parts.

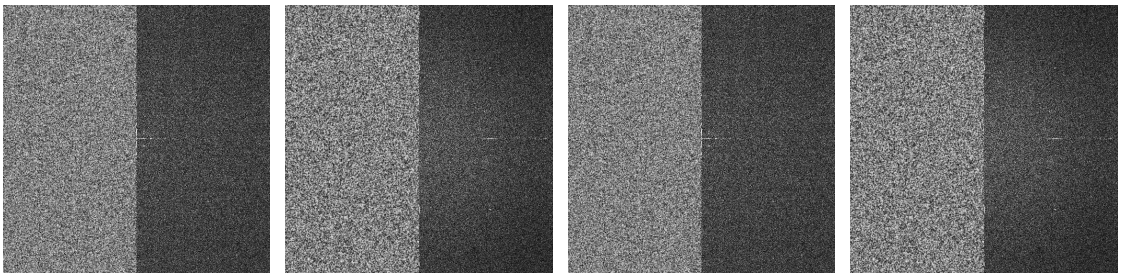
2.1.1 Choosing a suitable number of layers

One of the main parameters introduced when building this stationary zooming texture is the number of scaled layers being combined n. We introduced multiple layers to ensure temporal consistency, by maintaining the textural properties of the white noise. One of those textural properties is the noise scale, which should remain as close as possible to the raw white noise, as we want to keep the appearance of white noise.

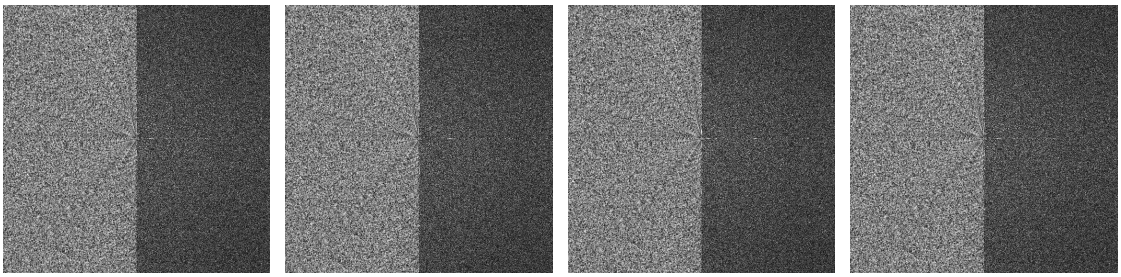
One of the tools that can be used for this is the power spectrum of the resulting texture. It can be computed using the discrete Fourier transform, and provides valuable information on the frequency distribution within the texture. In fig. 2.4, we computed the power spectrum of a few frames from an animation of the stationary zooming texture for two values of n, 3 and 8.

As we can see, some of the frames in the $n = 3$ case exhibit less power in the high frequencies. This means the resulting texture does not feature a similar power spectrum to white noise, which is constant over the whole frequency domain. Visually, this means the noise scale is not constant through time, therefore temporal consistency is not achieved. With more layers, this problem does not appear, as we can see in the $n = 8$ case.

Note that the power spectrum is only *expected* to be flat for white noise, meaning that any realization of white noise has a noisy spectrum. This is why power spectrums in fig. 2.4 are noisy. In order to measure a flat spectrum from white noise, one needs to average many realizations of white noise instances. Note that temporal averaging is not an option when trying to characterize temporal consistency artifacts, which would be smoothed out.



(a) $n = 3$



(b) $n = 8$

Figure 2.4: Loss of temporal coherence with a low number of layers. Each image shows the resulting texture and its associated power spectrum. Each frame has been captured one second apart. Note how high frequencies (corners of the spectrum) are sometimes lost when $n = 3$, especially on the 2^{nd} and 4^{th} frames, which is not the case when $n = 8$.

At the time of writing, the power spectrum has only been empirically observed. We do believe objective measures of artifacts could be derived from the power spectrum, and we will study this possibility during the remaining internship time.

2.2 Sliding artifact: studying and fixing

A sliding artifact is described by the viewer being able to see the features of different layers move over each other at different speeds. This artifact stems from the same screen point not having the same speed across all texture layers. To investigate this issue, let's express the speed of a point P of a texture layer i, expressed in screen coordinates. The a_i scaling coefficients are defined according to the introduction about building this texture, and formalized as eq. (2.2). $[0, H], H \in \mathbb{R}$ is the allowed scaling range for the texture layers.

$$a_i = \mathrm{mod}\left(i\frac{H}{n} - t, H\right), 0 \leq i < n \tag{2.2}$$

As the texture layer i is being scaled by a factor a_i, the screen coordinates $\mathbf{u}_P$ of a point $\mathbf{x}_P$ in the base texture is given by:

$$\mathbf{u}_P = a_i \mathbf{x}_P$$

Reversing this equation gives the relation between any texture layer point and the corresponding screen point. Deriving this equation over time results in $\mathbf{v}_P$, the speed of the point P in a texture layer i.

$$\mathbf{v}_P = \frac{d}{dt}\left(\frac{\mathbf{u}_P}{a_i}\right) = \frac{\mathbf{u}_P}{a_i}\left(-\frac{d\left(a_i\right)}{dt}\frac{1}{a_i}\right) = \frac{\mathbf{x}_P}{a_i} \tag{2.3}$$

The speed $\mathbf{v}_P$ is indeed dependent on i, thus introducing the sliding artifact. Rewriting eq. (2.3) for a generic function of time $\beta_i\left(t\right)$ and introducing the condition that $\frac{\mathbf{v}_P}{\mathbf{x}_P}$ should match a given function $\delta(t)$, independent of i, gives the differential eq. (2.4). In other words, we add the constraint that the speed of a given screen point should be constant with respect to the texture layer number i, resulting in an equal speed among all texture layers.

$$\begin{aligned} & a_i = \beta_i\left(t\right) \\ \Rightarrow & \mathbf{v}_P = -\frac{\beta_i'\left(t\right)}{\beta_i\left(t\right)}\mathbf{x}_P = \delta(t)\mathbf{x}_P \\ \Rightarrow & \beta_i'(t) + \delta\left(t\right)\beta_i(t) = 0 \end{aligned} \tag{2.4}$$

The family of solutions for β_i in eq. (2.4) is $\left\{t \mapsto Ke^{-\int_0^t \delta(u)du}, K \in \mathbb{R}\right\}$. Choosing how the apparent speed of texture features evolves is done by defining δ. To achieve the same linear zoom effect as in eq. (2.2), but without sliding, we choose $\delta(t) = 1$

(constant speed) and $K = He^{\frac{H}{n}i}$ (uniform distribution of layers across $[0, H]$). We then reintroduce the modulo operation in order to keep the same appearance over a given period, giving the new expression for a_i in eq. (2.5).

$$a_i = He^{-\bmod(t-\frac{H}{n}i,H)} \tag{2.5}$$

As can be seen in the animated[1] version of the resulting texture, the sliding artifact is no longer present, thus validating this model.

[1] http://bit.ly/st-lssfW8. Press the S key to toggle the fix on and off.

2.3 Texture properties: a statistical model

The remaining contrast loss and higher-order artifacts can be described in terms of image statistics. Indeed, an image's mean and variance describe its luminance and contrast. Higher-order statistics give information about higher-level features, such as shape, self-similarity, etc. These properties may either be viewed globally, as a single scalar describing the whole image, or locally, for each point in the image or over small analysis windows (fig. 2.5).

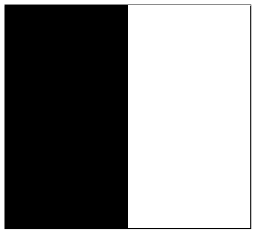


Figure 2.5: Image statistics. This image has a global mean of 1/2, however its local mean is either 0 or 1 depending on the position within the image. Its global contrast (standard deviation) is 1/2, while it is locally null everywhere except over the black and white transition.

Among higher-order statistics, autocorrelation is particularly interesting because of its perceptual significance. Autocorrelation is a metric describing how a given texture is spatially self-similar. This is analogous to the definition of autocorrelation in signal theory: a signal is said to be autocorrelated if it is self-similar over a certain time period. Perception experiments have shown that it is very likely the human perceptual system computes local autocorrelation values in early stages of vision, to prepare further shape and contour analysis. See section 2.6 for a discussion about the relevance of this descriptor.

In order to characterize the texturing artifacts, we will introduce a statistical model of the resulting texture. In eq. (2.6), we define $I : \mathbb{R}^2 \to \mathbb{R}$ to be the statistical process of the resulting texture. The intensity at each point $\mathbf{x}$ of the resulting texture is seen as a random variable. The associated probability space is the entire space of possible base texture realizations (noted g in eq. (2.6)) as we are interested in the properties of the texture creation process, not the artifacts of a given texture instance.

$$I(\mathbf{x}) = \sum_{i=0}^{n-1} A(a_i)\, g(a_i \mathbf{x}) \tag{2.6}$$

We want to study the statistical properties of $I(\mathbf{x})$, such as its mean, variance, and correlation between two points. We recall the following formulas for the variance, covariance and correlation coefficient for $\mathbf{x}, \mathbf{x_1}$ and $\mathbf{x_2}$ points in the texture.

$$\begin{aligned}
\operatorname{Var}(I(\mathbf{x})) &= E\left[(I(\mathbf{x}))^2\right] - E[I(\mathbf{x})]^2 \\
\operatorname{Cov}(I(\mathbf{x_1}), I(\mathbf{x_2})) &= E[I(\mathbf{x_1})\, I(\mathbf{x_2})] - E[I(\mathbf{x_1})]\, E[I(\mathbf{x_2})] \\
\operatorname{Corr}(I(\mathbf{x_1}), I(\mathbf{x_2})) &= \frac{\operatorname{Cov}(I(\mathbf{x_1}), I(\mathbf{x_2}))}{\sqrt{\operatorname{Var}(I(\mathbf{x_1}))}\sqrt{\operatorname{Var}(I(\mathbf{x_2}))}}
\end{aligned}$$

To ease notations, we introduce Φ, defined by relation eq. (2.7).

$$\begin{aligned}\Phi(\mathbf{x_1},\mathbf{x_2}) &\triangleq E[I(\mathbf{x_1})\,I(\mathbf{x_2})]\\ &= E\left[\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A(a_i)\,A(a_j)\,g(a_i\mathbf{x_1})\,g(a_j\mathbf{x_2})\right]\\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A(a_i)\,A(a_j)\,E[g(a_i\mathbf{x_1})\,g(a_j\mathbf{x_2})]\end{aligned} \tag{2.7}$$

This results in the expressions summarized in table 2.2 for the statistical descriptors we are interested in.

Property of I	Value
Mean $E[I(\mathbf{x})]$	$E[g(\mathbf{x})]$
Variance $\mathrm{Var}(I(\mathbf{x}))$	$\Phi(\mathbf{x},\mathbf{x}) - E[g(\mathbf{x})]^2$
Covariance $\mathrm{Cov}(I(\mathbf{x_1}), I(\mathbf{x_2}))$	$\Phi(\mathbf{x_1},\mathbf{x_2}) - E[g(\mathbf{x_1})]\,E[g(\mathbf{x_2})]$
Correlation $\mathrm{Corr}(I(\mathbf{x_1}), I(\mathbf{x_2}))$	$\frac{\Phi(\mathbf{x_1},\mathbf{x_2})-E[g(\mathbf{x_1})]E[g(\mathbf{x_2})]}{\sqrt{\Phi(\mathbf{x_1},\mathbf{x_1})-E[g(\mathbf{x_1})]^2}\sqrt{\Phi(\mathbf{x_2},\mathbf{x_2})-E[g(\mathbf{x_2})]^2}}$

Table 2.2: Statistical properties of a blended texture

Let's further define the base texture g to explain those properties. Let g be the result of the linear interpolation of a discrete 2D white noise instance f to produce the scaled layers. f is of mean μ_f and variance σ_f^2, generated at the beginning of the rendering. The linear interpolation process can be written as a discrete convolution of f with Λ, the 2D triangle function of unit amplitude, zero-valued outside $[-1,1]^2$, as in eq. (2.8), $\mathbf{x}$ being a point in the resulting texture.

$$g(\mathbf{x}) = \sum_{\mathbf{k}} f(\mathbf{k})\,\Lambda(\mathbf{x}-\mathbf{k}) \tag{2.8}$$

Note that linear interpolation preserves intensity over the texture, implying eq. (2.9).

$$\sum_{\mathbf{k}} \Lambda(\mathbf{x}-\mathbf{k}) = 1,\ \mathbf{k} \text{ over the whole texture} \tag{2.9}$$

Using the linearity property of the mean, we obtain $E[g(\mathbf{x})] = E[f(\mathbf{x})]$.

We then use the property of independence between two points of the white noise texture f, as well as the definition of A to simplify Φ (eq. (2.10)).

$$\begin{aligned}
\Phi(\mathbf{x_1},\mathbf{x_2}) &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A(a_i)\,A(a_j)\sum_{\mathbf{k}}\sum_{\mathbf{l}} E\left[f(\mathbf{k})\,f(\mathbf{l})\right]\Lambda(a_i\mathbf{x_1}-\mathbf{k})\,\Lambda(a_j\mathbf{x_2}-\mathbf{l}) \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A(a_i)\,A(a_j)\left(\mu_f^2+\sigma_f^2\sum_{\mathbf{k}}\Lambda(a_i\mathbf{x_1}-\mathbf{k})\,\Lambda(a_j\mathbf{x_2}-\mathbf{k})\right) \\
&= \mu_f^2+\sigma_f^2\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A(a_i)\,A(a_j)\sum_{\mathbf{k}}\Lambda(a_i\mathbf{x_1}-\mathbf{k})\,\Lambda(a_j\mathbf{x_2}-\mathbf{k}) \\
&= \mu_f^2+\sigma_f^2\alpha(\mathbf{x_1},\mathbf{x_2})
\end{aligned} \tag{2.10}$$

To ease notations we introduced $\alpha(\mathbf{x_1},\mathbf{x_2})$ as follows:

$$\alpha(\mathbf{x_1},\mathbf{x_2}) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A(a_i)\,A(a_j)\sum_{\mathbf{k}}\Lambda(a_i\mathbf{x_1}-\mathbf{k})\,\Lambda(a_j\mathbf{x_2}-\mathbf{k}) \tag{2.11}$$

Note that the statistical properties of discrete white noise (independence between two samples) gives the equalities 2.12, used in eq. (2.10).

$$\begin{aligned}
&\sum_{\mathbf{k}}\sum_{\mathbf{l}} E\left[f(\mathbf{k})\,f(\mathbf{l})\right]\Lambda(a_i\mathbf{x_1}-\mathbf{k})\,\Lambda(a_j\mathbf{x_2}-\mathbf{k}) \\
=&\sum_{\mathbf{k}}\left(E\left[f(\mathbf{k})^2\right]\Lambda(a_j\mathbf{x_2}-\mathbf{k})+\sum_{\mathbf{l}\neq\mathbf{k}} E\left[f(\mathbf{k})\,f(\mathbf{l})\right]\Lambda(a_j\mathbf{x_2}-\mathbf{l})\right)\Lambda(a_i\mathbf{x_1}-\mathbf{k}) \\
=&\sum_{\mathbf{k}}\left(\left(\sigma_f^2+\mu_f^2\right)\Lambda(a_j\mathbf{x_2}-\mathbf{k})+\sum_{\mathbf{l}\neq\mathbf{k}} E\left[f(\mathbf{k})\right]E\left[f(\mathbf{l})\right]\Lambda(a_j\mathbf{x_2}-\mathbf{l})\right)\Lambda(a_i\mathbf{x_1}-\mathbf{k}) \\
=&\sum_{\mathbf{k}}\left(\left(\sigma_f^2+\mu_f^2\right)\Lambda(a_j\mathbf{x_2}-\mathbf{k})+\mu_f^2\sum_{\mathbf{l}}\Lambda(a_j\mathbf{x_2}-\mathbf{l})-\mu_f^2\Lambda(a_j\mathbf{x_2}-\mathbf{k})\right)\Lambda(a_i\mathbf{x_1}-\mathbf{k}) \\
=&\mu_f^2+\sigma_f^2\sum_{\mathbf{k}}\Lambda(a_i\mathbf{x_1}-\mathbf{k})\,\Lambda(a_j\mathbf{x_2}-\mathbf{k})
\end{aligned} \tag{2.12}$$

Injecting the new expression of Φ into the expression of the statistical descriptors from table 2.2 results in the following properties for this texture, summarized in table 2.3.

The following properties of the resulting texture can be deduced:

- The resulting texture mean is the same as the base texture, and independent of the position. The resulting texture has the same luminance as the base texture.
- The resulting texture variance is proportional to the original texture variance, with a coefficient dependent on the position within the texture. Knowing that $\alpha(\mathbf{x_1}, \mathbf{x_2}) \leq 1$, we can deduce the resulting texture features a loss of contrast compared to the base texture.
- The correlation coefficient is only dependent on α, thus the artifacts that are described by this descriptor are only a consequence of the choice of the scaling coefficient and how the different base texture layers are combined.

Property of I	Value
Mean $E[I(\mathbf{x})]$	μ_f
Variance $\mathrm{Var}(I(\mathbf{x}))$	$\sigma_f^2 \alpha(\mathbf{x}, \mathbf{x})$
Covariance $\mathrm{Cov}(I(\mathbf{x_1}), I(\mathbf{x_2}))$	$\sigma_f^2 \alpha(\mathbf{x_1}, \mathbf{x_2})$
Correlation $\mathrm{Corr}(I(\mathbf{x_1}), I(\mathbf{x_2}))$	$\frac{\alpha(\mathbf{x_1},\mathbf{x_2})}{\sqrt{\alpha(\mathbf{x_1},\mathbf{x_1})}\sqrt{\alpha(\mathbf{x_2},\mathbf{x_2})}}$

Table 2.3: Statistical properties of the white noise zoom texture

2.4 Contrast loss artifact: studying and fixing

The loss of contrast is a direct consequence of the blending of multiple layers. In this texture, the blending weights have been chosen in such a way that the mean of the resulting texture is the same as the original white noise process mean. Indeed, the texture luminance hasn't changed compared to the white noise texture, but it shows significantly more half-gray values than the base texture.

This is expected, according to the previously introduced statistical model. As the variance of the resulting texture is $\sigma_f^2 \alpha(\mathbf{x}, \mathbf{x})$, and $\alpha(\mathbf{x}, \mathbf{x})$ is lower than 1 by definition, the resulting contrast is lowered by an $\alpha(\mathbf{x}, \mathbf{x})$ factor compared to the raw white noise. We can, to a certain extent, recover from this loss of contrast through various methods, as presented in the following sections.

2.4.1 Linear contrast correction

The variance of an image is a descriptor of its local contrast, therefore one can change its contrast through operations on the intensity values. We know that for $a \in \mathbb{R}$ and X a random variable of finite variance, we have property eq. (2.13).

$$\operatorname{Var}(aX) = a^2 \operatorname{Var}(X) \tag{2.13}$$

Yu et al. [Yu+11] used this property to restore the contrast of the result of a texture advection process. If $I(\mathbf{x})$ describes the pixel intensity values of a texture, using property eq. (2.13) we can restore the contrast of the texture using a suitable coefficient.

$$\operatorname{Var}\left(\sqrt{a}\frac{I(\mathbf{x})}{\sqrt{\operatorname{Var}(I(\mathbf{x}))}}\right) = a \tag{2.14}$$

In the case of the zooming texture, $a = \sigma_f^2$, the variance of the base texture, and $\operatorname{Var}(I(\mathbf{x})) = \sigma_f^2 \alpha(\mathbf{x}, \mathbf{x})$. However, because pixel intensity values are bounded to the $[0, 1]$ interval, this method introduces clamping: new pixel values that lie outside of this interval are replaced by 0 or 1 in order to be displayed on screen. In fig. 2.6 page 38, we can see that the linear contrast corrected image is only made of black and white because of clamping[2]. As a consequence, the texture does not have the same appearance as the source white noise which does have a whole range of gray values.

Even if this texture is not displayed on screen, and thus does not "require" clamping, the dynamic range of pixel intensities is left unknown, which renders the

[2] Please note additional gray values may be introduced by the document viewer interpolating between black and white pixels.

texture unusable as a source of randomness for further processing. In the next part, we will try to find a smoother function which does not introduce as much clamping as this method.

2.4.2 Polynomial contrast correction

In order to find a suitable method for contrast correcting, we need to introduce more parameters to allow for greater control of the contrast correction process. Let h be a continuous function of $I(\mathbf{x})$ with the following properties:

1. It should map intensities from the $[0,1]$ range to the same $[0,1]$ range, to prevent clamping.
2. It should be monotonously increasing, in order to prevent exchanging brighter and darker tones.
3. It has to restore the variance of the original texture to the best possible extent, but the mean should also be preserved.

The first two properties can be represented by the set of equations eq. (2.15):

$$\begin{cases} h(0) = 0 \\ h(1) = 1 \\ h'(x) \geq 0 \ \forall x \in [0,1] \end{cases} \tag{2.15}$$

The last property can be approximated by using the Taylor expansions of the moments of functions of random variables. h is the function being expanded, and $I(\mathbf{x})$ the random variable. Discarding higher-order terms results in the approximations in equation system eq. (2.16). σ_r^2 is the variance of $I(\mathbf{x})$.

$$\begin{cases} E\left[h\left(I\left(\mathbf{x}\right)\right)\right] \approx h\left(\mu_f\right) + \dfrac{h''\left(\mu_f\right)}{2}\sigma_r^2 \\ \mathrm{Var}\left(h\left(I\left(\mathbf{x}\right)\right)\right) \approx \left(h'\left(\mu_f\right)\right)^2 \sigma_r^2 \end{cases} \tag{2.16}$$

Let h be a 3rd degree polynomial, i.e. $h(x) = a + bx + cx^2 + dx^3$. We then introduce the constraints $E\left[h\left(I\left(\mathbf{x}\right)\right)\right] = \mu_f$ and $\mathrm{Var}\left(h\left(I\left(\mathbf{x}\right)\right)\right) = \sigma_f^2$, resulting in equation system eq. (2.17).

$$\begin{cases} h(0) = 0 \\ h(1) = 1 \\ h'(x) \geq 0 \ \forall x \in [0,1] \\ \left(h'(\mu_f)\right)^2 = \dfrac{\sigma_f^2}{\sigma_r^2} \\ h(\mu_f) + \dfrac{h''(\mu_f)}{2}\sigma_r^2 = \mu_f \end{cases} \tag{2.17}$$

From eq. (2.17) we can extract the linear subsystem eq. (2.18) of the parameters of h, which can be solved using Cramer's rule. If a solution to this system exists, it will be unique and will be a solution of eq. (2.17) if it verifies the remaining equations. Note that in order to keep eq. (2.18) of dimension 3, we trivially resolve $h(0) = 0$ to $a = 0$.

$$\begin{cases} b + & c + & d = 1 \\ b + & 2\mu_f c + & 3\mu_f^2 d = \sqrt{\dfrac{\sigma_f^2}{\sigma_r^2}} \\ \mu_f b + & \left(\mu_f^2 + \sigma_r^2\right) c + & \left(\mu_f^3 + 3\mu_f\sigma_r^2\right) d = \mu_f \end{cases} \tag{2.18}$$

Solving this system results in a potential solution for h:

$$\begin{aligned} h(x) = {} & \left(\mu_f^2 - 2\mu_f^3 + \mu_f^4 - \sigma_r^2\left(1 - 3\mu_f + 3\mu_f^2\right)\right)^{-1} \\ & \left(\left(2\mu_f^2 - 3\mu_f^3 + \mu_f^4 - 3\mu_f^2\sigma_r^2 - \sqrt{\frac{\sigma_f^2}{\sigma_r^2}}\left(\mu_f^2 - \mu_f^3 + \sigma_r^2 - 3\mu_f\sigma_r^2\right) \right) x \right. \\ & \left. + \left(1 - \sqrt{\frac{\sigma_f^2}{\sigma_r^2}}\right)\left(\left(-\mu_f + \mu_f^3 + 3\mu_f\sigma^2\right)x^2 + \left(\mu_f - \mu_f^2 - \sigma_r^2\right)x^3\right) \right) \end{aligned} \tag{2.19}$$

This solution is only acceptable for eq. (2.17) as long as $h' \geq 0$. Let Δ be the discriminant of $h'(x)$. We can deduce from its expression that $\Delta \leq 0$ if $\sigma_r > \sigma_f$, but this is impossible since $\alpha(\mathbf{x}, \mathbf{x}) \leq 1$ and $\sigma_r^2 = \alpha(\mathbf{x}, \mathbf{x})\,\sigma_f^2$. Though, we may introduce a condition on the positive solution of $h'(x)$ when $\Delta \geq 0$, such as this solution is outside of the variance range $\left[0, \sigma_f^2\right]$.

Indeed, if $\sigma_r^2 \geq \frac{4}{9}\sigma_f^2$ (for $\mu_f = \frac{1}{2}$), we can ensure $h'(x) \geq 0$. This means h is a valid solution only for those areas, although we can verify on the resulting texture that not that many pixels exhibit lower variance thus breaking this condition. The direct effect is that some pixel values will be clamped, but they will be less

numerous than when restoring contrast using a linear relation. In fig. 2.6 page 38, we can see the contrast has been restored, and less pixels are being clamped. The texture appearance is closer yet still different from the raw white noise texture, as indicated by the histogram: a raw white noise should have a flat histogram, and this is not yet the case with this method.

This method exploited the property that statistical descriptors may be approximated using power series. In other words, the effect of a contrast correction may be approximated using the derivatives of the contrast correction function at the mean value. Therefore, a function that keeps all previous properties while being controllable (in value and derivative) near the mean value can serve as a contrast correction method. Such a function is presented in the next section.

2.4.3 Sigmoid contrast correction

Another method used by image editors, including the open-source ImageMagick[3] suite is a kind of functions referred to as "normalized sigmoids" due to their resemblance with the sigmoid function, but over a compact support. A definition of a normalized centered sigmoid over $[-1, 1]$ is given in eq. (2.20). Note that $x \mapsto \frac{1+h(2x-1)}{2}$ can be used to change the support to $[0, 1]$.

$$h(x) = \frac{kx}{1 + k - |x|}, k \in]-\infty, -1] \cup [0, +\infty[\tag{2.20}$$

This function is not derivable at 0, however the left and right limits of the derivative h' converge to the same value, so the definition of h' can be extended to be defined at 0 (see eq. (2.21)).

$$\begin{aligned} h'(x) &= \frac{k}{1 + k - |x|} + \frac{kx\,(|x|)'}{(1 + k - |x|)^2} \\ \Rightarrow \lim_{x \to 0^{\pm}} h'(x) &= \frac{k}{1 + k} + 0^{\pm} \end{aligned} \tag{2.21}$$

The second derivative h'' is however undefined near 0 and cannot be extended. This means the development of $h\left(I\left(\mathbf{x}\right)\right)$ is only possible up to the first order. In order to recover the lost contrast, this leads to the definition of k as follows:

$$k = \frac{\sqrt{\frac{\sigma_f^2}{\sigma_r^2}}}{1 - \sqrt{\frac{\sigma_f^2}{\sigma_r^2}}} \tag{2.22}$$

[3] https://www.imagemagick.org/script/index.php

This function never introduces clamping, however it has uncontrollable effects on the histogram due to the absence of constraints on higher-order derivatives, as can bee seen in fig. 2.6. These effects may be undesirable depending on the use for the zooming texture, and even worse if used with other procedural noise primitives with more complex histograms.

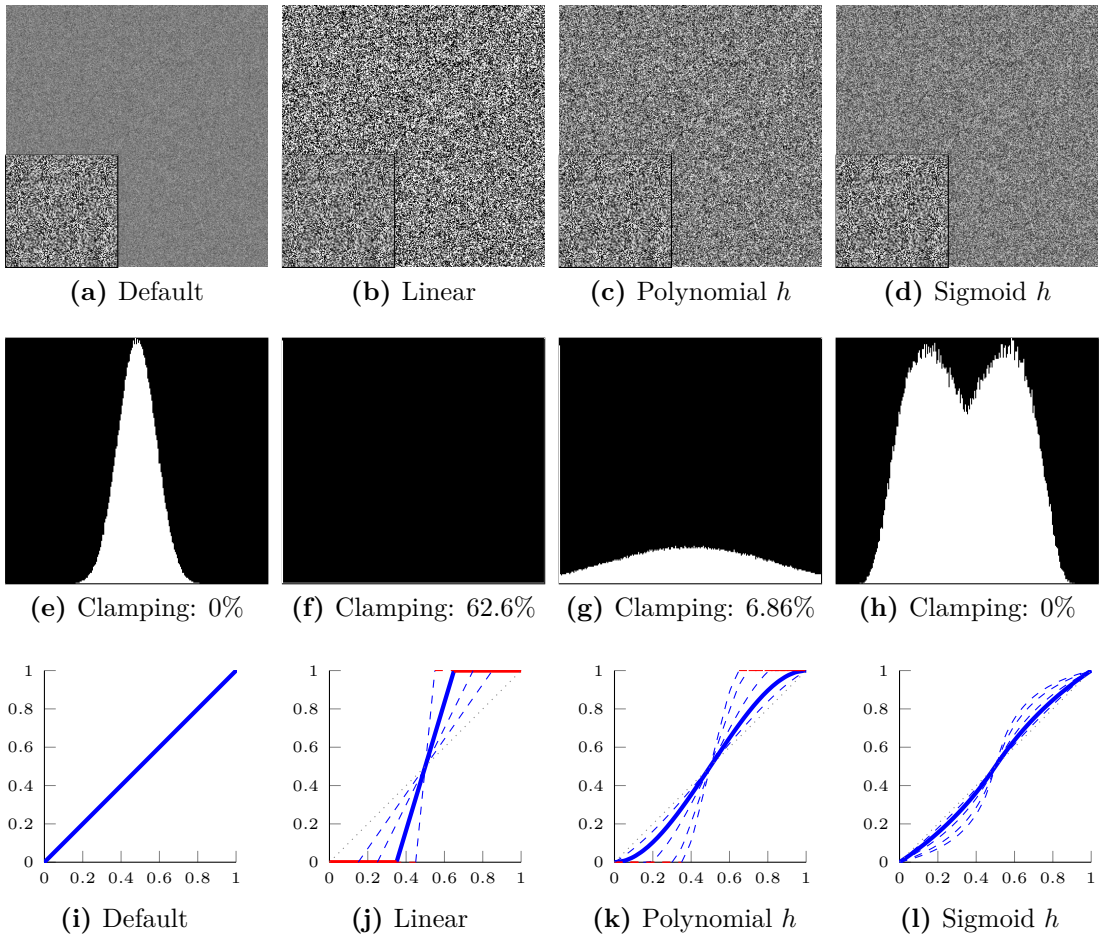
2.4.4 Discussion

The results of all three contrast-correction methods established in this part are represented in fig. 2.6. Linear normalization cannot be used when the variance to be restored is too low, as this will introduce clamping which destroys the texture appearance and histogram.

The polynomial normalization statistically restores the contrast of the original picture, although some differences can still be perceived compared to the raw white noise texture. Residual clamping may be the cause, but other effects may interfere with this perception[4].

More complex functions such as sigmoids may seem a good option to restore contrast, however their effect on the image histogram is unpredictable, and highly dependent on the original texture. The problems arising from the use of such functions are similar to the ones that arise from using mapping curves in image and texture authoring software. Such non-linear effects require a lot of tuning in order to choose the best compromise between all texture properties (luminance, contrast, histogram, etc.) to obtain the desired rendering.

[4]Interfering effects may include low spatial frequencies naturally present in white noise instances, which are easily tracked by the visual system as blurry shapes.



(a) Default **(b)** Linear **(c)** Polynomial h **(d)** Sigmoid h

(e) Clamping: 0% **(f)** Clamping: 62.6% **(g)** Clamping: 6.86% **(h)** Clamping: 0%

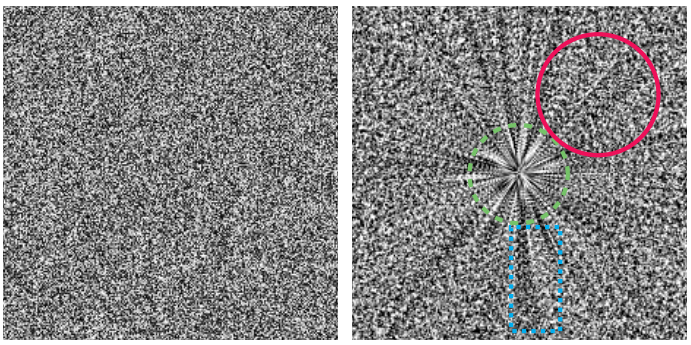
(i) Default **(j)** Linear **(k)** Polynomial h **(l)** Sigmoid h

Figure 2.6: Different methods of contrast correction (with $n = 16$). From top to bottom: zooming texture, histogram, and plot of the contrast correction function. Inset shows a raw white noise texture for reference. The linear correction (b) introduces a lot of clamping thus making the histogram bimodal, pure black and white. The polynomial correction (c) still introduces clamping, but the rest of the histogram is closer to a flat (white noise-like) histogram. Clamping effects decrease with the number of layers. Histograms (f) and (g) have two bins with high values completely to the left and to the right. Red area on plots indicate clamping intervals. Clamping percentages indicate the amount of pixels which have been clamped on these 512x512 textures.

2.5 Correlation artifacts: studying and fixing

For better readability, the figures in this part have been contrast-corrected using the polynomial method introduced in subsection 2.4.2.

The star-like pattern and other radial artifacts (see fig. 2.7) are a visual consequence of the non-zero correlation between different points in the image. The human perceptual system is very sensitive to autocorrelation in images (see section 2.6 for more details), which is why the zoom illusion looks very disrupted by this artifact. As white noise is autocorrelation-free, this also renders the resulting texture visually different from the base texture.



(a) White noise texture **(b)** Resulting texture

Figure 2.7: Correlation artifacts on white noise. Correlation artifacts include echoes or repeating features (red, solid), still features (green, dashed), darker and lighter areas (blue, dotted). To the left is a white noise texture for comparison.

2.5.1 Properties of correlation artifacts

Note that the darker and lighter areas (blue, dotted in fig. 2.7) are artifacts caused by a given white noise instance. They arise from low-frequency patterns existing in white noise instances, and extend over the whole texture when the fractalization process is applied to it. Using a different white noise instance (for example, by changing the seed for the pseudo-random number generator) radically changes the appearance of these artifacts. However, their effect is not visible on the value of the correlation coefficient when the probability space is the entire space of base texture realizations instead of the space of a single rendered texture. We chose to focus firstly on the correlations introduced by the process itself, but as we will see, fixing the process artifacts also fixes these "instance- specific" artifacts. In other words, we don't have a mathematical description of the lighter and darker areas, but they will not be present once the correlation artifacts introduced by the process are fixed.

As we mentioned when analyzing the expression of Corr $(I(\mathbf{x_1}), I(\mathbf{x_2}))$ (section 2.3), correlation artifacts are only dependent on the choice of the a_i and how they are used to scale the different layers. We can therefore change the different texture parameters such as the number of layers n and definition of a_i coefficients (removing or introducing sliding) in order to study their influence on the appearance of the texture.

As we defined the correlation coefficient on the probabilistic space comprising all the possible realizations of base textures (the white noise instances), we will therefore visualize the correlation coefficient of the texturing process itself. However, the correlation coefficient of a 2D texture requires a 4D space for visualization, which is not practical for drawing conclusions. The radial symmetry of the mentioned artifacts can be exploited to instead visualize Corr $[I(\langle x;0\rangle), I(\langle y;0\rangle)]$, ie. the correlation coefficient between two points x and y along the horizontal axis. We can also visualize Corr $[I(\langle x;0\rangle), I(\langle x;y\rangle)]$ in order to prove the absence of non-radial correlation artifacts.

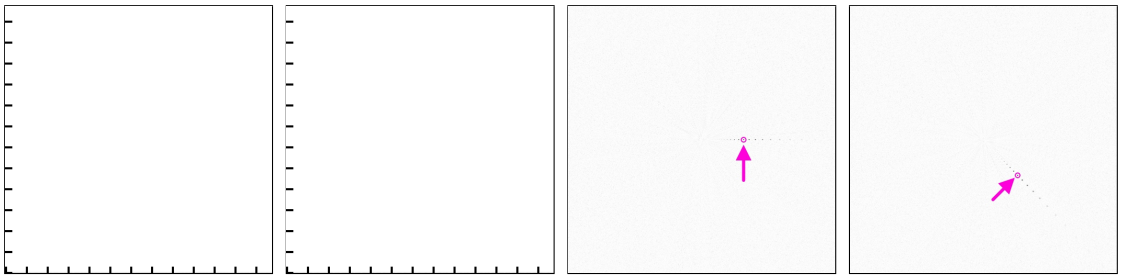
This results in 2D gray level plots, as in fig. 2.10 and 2.9b. Plot 2.9b is computed using the analytical expression correlation coefficient (see Table 2.3), however plot 2.9c has been computed using a statistical estimator of the correlation coefficient over 512 white noise samples, thus resulting in a slight noise less than 5% in value. Converging analytical and statistical plots indicate that the analytical expression is coherent with the actual values of the correlation coefficient in the texture.

The resulting plots allow us to describe the correlation artifacts in the texture (see fig. 2.9):

- The center area (red) which looks like a distinct shape, corresponds to an area where non-zero autocorrelation spans over several pixels. A large autocorrelation area indeed describes a feature of matching size.
- Correlation lines in the 2D plot describe the echoes that can be seen in the resulting texture (orange). The distance along the y axis of the plot maps to the distance between one feature and the corresponding echoes. Figure 2.9d is a plot of the actual correlation values in this area.
- Although the correlation coefficient is non-zero, no artifacts can be perceived far away from the texture origin (blue). This shows a discrepancy between the descriptor (the correlation coefficient) and the appearance (how we perceive the texture). This will be further discussed in section 2.6.
- The pattern along the diagonal (green and inset) characterizes both the interpolation kernel used for scaling the base textures and the base texture themselves. Indeed, it is the result of reconstruction of the base texture using the chosen interpolation kernel. Here, as the discrete texture is a white noise, it is described by a Dirac comb distribution, which results in

the interpolation kernel being displayed. The inset (2.9b) details the green area using either a nearest or linear interpolation mode.

The plots of Corr $[I(\langle x;0\rangle), I(\langle x;y\rangle)]$ (see fig. 2.8) allow us to confirm our hypothesis that no correlation occurs outside of radial axes, thus making the chosen visualization of the correlation between points $\langle x;0\rangle$ and $\langle y;0\rangle$ coherent.



(a) $n = 16$ **(b)** $n = 8$ **(c)** $n = 16$ **(d)** $n = 16$

Figure 2.8: Non-radial correlation. (a) and (b) represent Corr $[I(\langle x;0\rangle), I(\langle x;y\rangle)]$ for two different layer counts, computed from the analytical expression. No correlation appears aside from the expected correlation of value 1.0 between a point and itself (bottom of the graph). (c) and (d) are maps of the statistically measured correlation between a chosen point (circled black dot) and the rest of the texture over 4096 samples. Aside from the slight statistical noise ($< 5\%$), no correlation exists outside the radial axis for the current point.

fig. 2.10 features the same plots as 2.9, for varying values of n (the number of layers) and a_i definition. Correlation artifacts are less noticeable as the number of layers decrease, however a layer count too low does not ensure temporal coherence, as luminance, contrast and scale fluctuations become noticeable (see section 2.1.1).

The correlation coefficient plots look blurrier when the sliding artifact is present (2.10, (a) to (d)). This is explained by the fact that layers exhibiting sliding are less coherent, thus autocorrelation patterns are spread out. When summed over multiple layers, this results in a blurred autocorrelated area, corresponding to the sliding area near the edges of the texture. When the sliding artifact is not present, each layer contributes an exact (scaled) copy of a particular point to the final rendering, resulting in clear echoes in other plots (2.10, (e) to (p)).

2.5.2 Fixing correlation artifacts

The correlation coefficient expression (table 2.3 and eq. (2.11)) is a product of triangle Λ functions which results from the summation of linearly interpolated layers of the same texture. If the a_i coefficients are close enough, the triangle

function product is non-zero, resulting in correlation. If the support domains of the triangle functions are disjoint, no correlation results.

In other words, since correlation is entirely due (but for the small interpolation kernel) to layer summation, using independent textures for each layer would cancel it. But storage as well as hardware texture access are expensive, which makes this solution impractical. A simple solution to emulate independent texture layers is to use an offset when accessing the base white noise instance. If this array is large enough, accessing it using an offset and wrapping around the edges (ie. the array is repeating infinitely).

Note that simple offsets, even non-linear just displaces the location of the artifacts, which does not solve the problem (see fig. 2.10, (i) to (l)). The expression of the texture in this case is as follows (w is the texture width in pixels):

$$I(\mathbf{x}) = \sum_{i=0}^{n-1} A(a_i)\, g\left(a_i\mathbf{x} + \left(w\frac{i}{n}\right)^2 \langle 1;0\rangle\right)$$

Simple "hacks" like this are usually done by artists when authoring procedural textures. In that context, such a solution may be sufficient if, for instance, the part of the texture being rendered was sufficiently far away from the area where the artifact is perceptible. However, correlation plots show the artifact is still present. In order to really avoid correlation artifacts, we can instead introduce a pseudo-random offset function of i. A typical way of generating pseudo-random numbers is "hashing", a process that scrambles the bits of the binary representation of floating point numbers[5] In this work, we use the following function ($fract$ returns the fractional part of a number, operating piecewise on vectors, and sin is expanded to also operate piecewise):

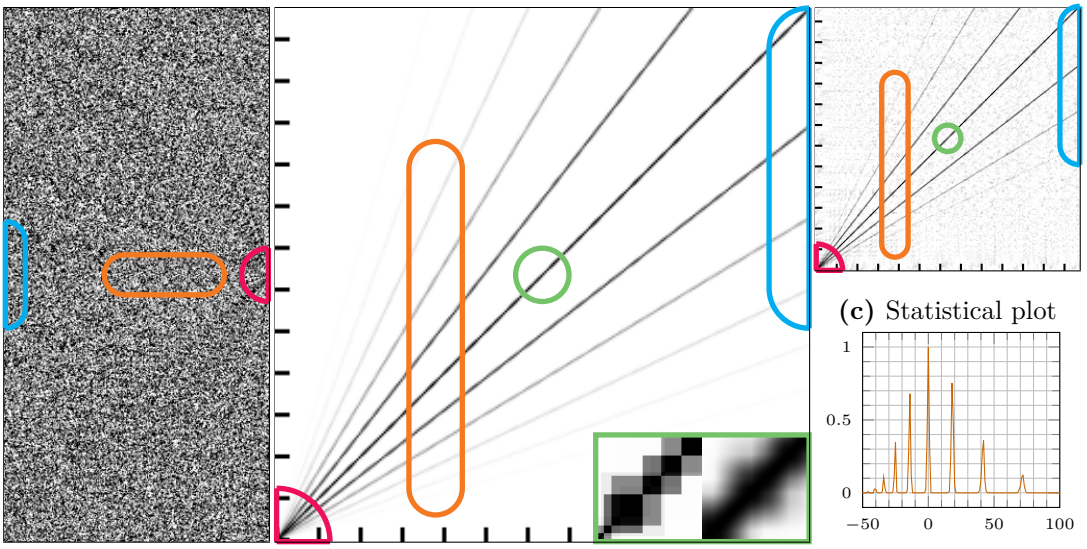
$$rand(i) = fract(4567.89 \sin(765.43 \times i \times \langle 1;-13.17\rangle + \langle 1;1\rangle)) \tag{2.23}$$

We can see on fig. 2.10 ((m) to (p)) that the correlation artifacts are no longer present. The expression of the texture in this case is as follows ($\times$ denotes piecewise multiplication, and h the texture height in pixels):

$$I(\mathbf{x}) = \sum_{i=0}^{n-1} A(a_i)\, g\left(a_i\mathbf{x} + rand(i) \times \langle w;h\rangle\right)$$

Note how fixing the artifacts resulting from the texture process also removed the lighter and darker areas resulting from low-frequency patterns in white noise instances.

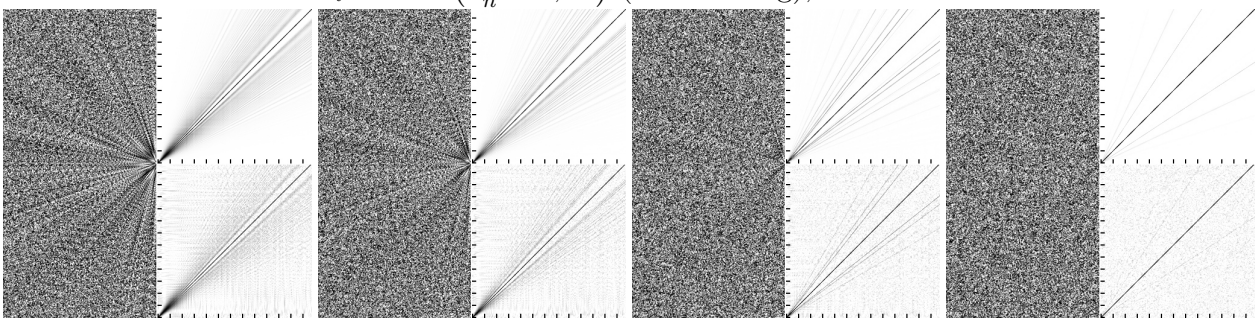
[5] Proving the quality of numbers generated by this method is outside the scope of this work. We only checked the mean, range and histogram empirically.



(a) Resulting texture **(b)** Correlation plot **(d)** 1D slice

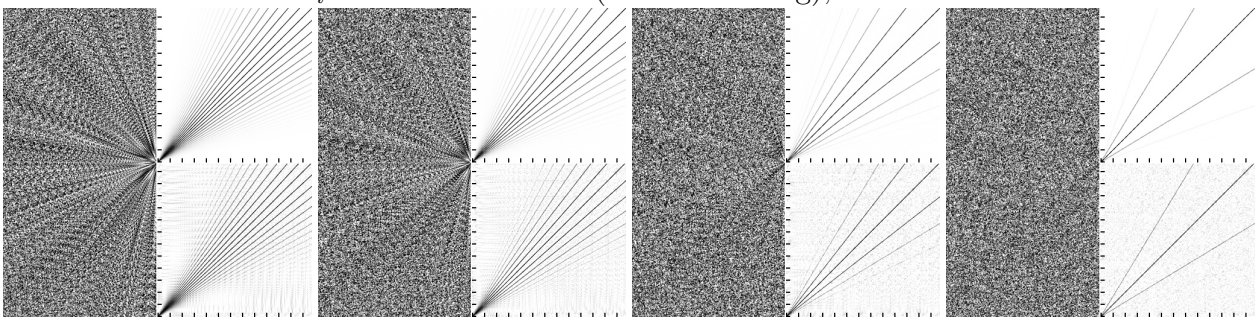
Figure 2.9: Correlation plot (b) and one resulting texture instance (a) with $n = 8$, without sliding artifacts. (c) is a statistical version of the correlation plot (b). (d) is a slice from the orange area in plot (b) to show the actual correlation values. The center of the texture is (a): in the middle to the right, (b) and (c): in the bottom left corner. The origin in (d) corresponds to the correlation between a point and itself, as is the main diagonal in graphs (b) and (c).

$a_i = \mathrm{mod}\left(i\frac{H}{n} - t, H\right)$ (with sliding), no offset



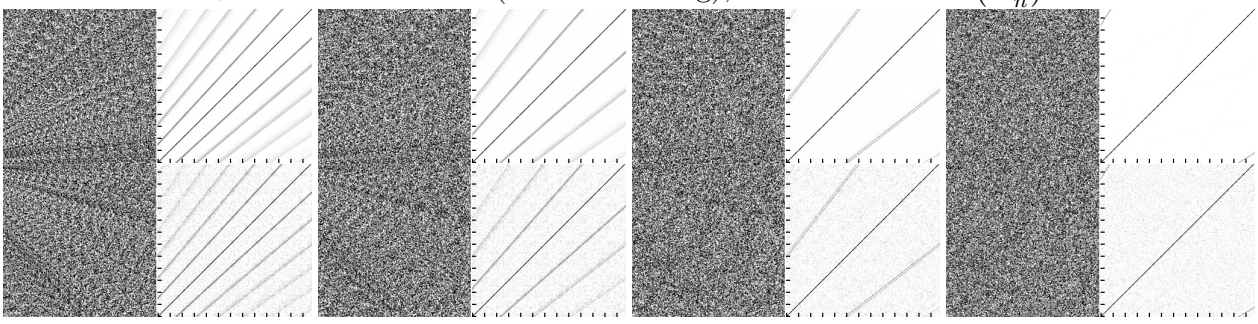
(a) $n = 24$ **(b)** $n = 16$ **(c)** $n = 8$ **(d)** $n = 4$

$a_i = He^{-\mathrm{mod}\left(i\frac{H}{n} - t, H\right)}$ (without sliding), no offset



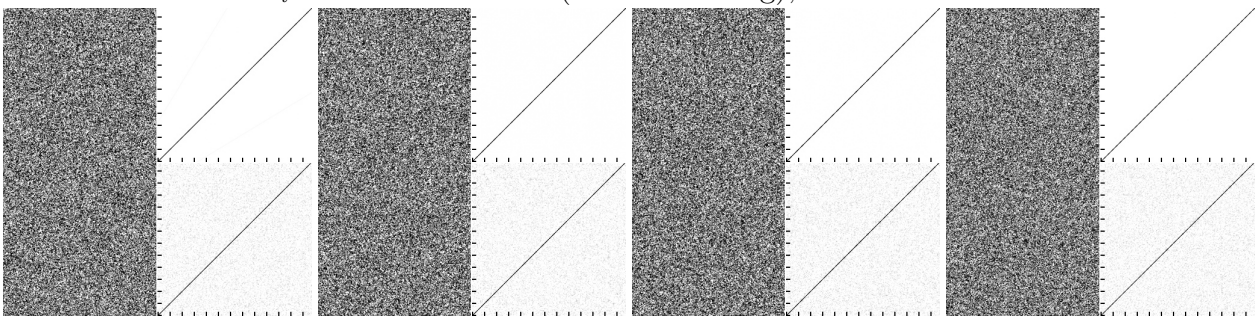
(e) $n = 24$ **(f)** $n = 16$ **(g)** $n = 8$ **(h)** $n = 4$

$a_i = He^{-\mathrm{mod}\left(i\frac{H}{n} - t, H\right)}$ (without sliding), non-linear offset $\left(w\frac{i}{n}\right)^2$



(i) $n = 24$ **(j)** $n = 16$ **(k)** $n = 8$ **(l)** $n = 4$

$a_i = He^{-\mathrm{mod}\left(i\frac{H}{n} - t, H\right)}$ (without sliding), random offset



(m) $n = 24$ **(n)** $n = 16$ **(o)** $n = 8$ **(p)** $n = 4$

Figure 2.10: Correlation plot for the white noise zoom texture. The left side of each picture is an instance of the resulting texture, while the right side shows the probabilistically computed (top) vs. statistically measured over 512 samples (bottom) correlation. A (x, y) point in a correlation plot corresponds to the correlation value between two points $\langle x; 0\rangle$ and $\langle y; 0\rangle$, ie. $\mathrm{Corr}\left[I\left(\langle x; 0\rangle\right), I\left(\langle y; 0\rangle\right)\right]$. Top to bottom: results for different variants. Left to right: decreasing number of layers.

2.6 Perception of correlation

2.6.1 Introduction on perception experiments

Perception experiments as introduced by Glass [Gla69] showed that the perception of superimposed dot patterns (see fig. 2.11) provides evidence for the fact that the human visual process may include the computation of local autocorrelations of perceived images. These experiments consisted in showing correlated (or not) dot patterns to users, and asking them if they saw a specific pattern or not. In this case, the pattern being perceived is called a Moiré effect, as concentric circles appear when a dot pattern is shown on top of a slightly rotated copy of itself. Later experiments showed a threshold exists on the perception of correlated dot pairs Lánsky, Mates, and Radil [LMR89]: users were not able to figure out the orientation of rotated dot patterns if the rotation angle was too large (see fig. 2.12).

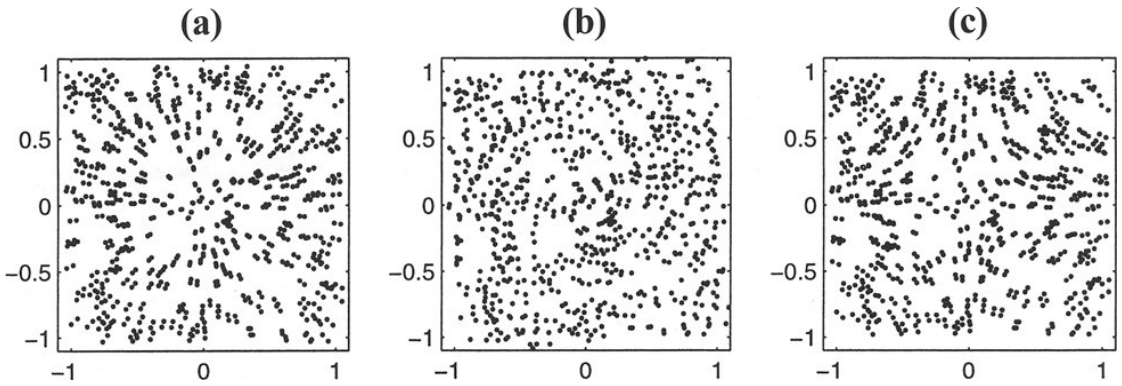


Figure 2.11: Correlated dot patterns. These patterns are created by superimposing a random dot pattern with another copy of itself after (a) scaling along x and y axes (b) rotating (c) scaling differently along x and y. Source: [Gla02]

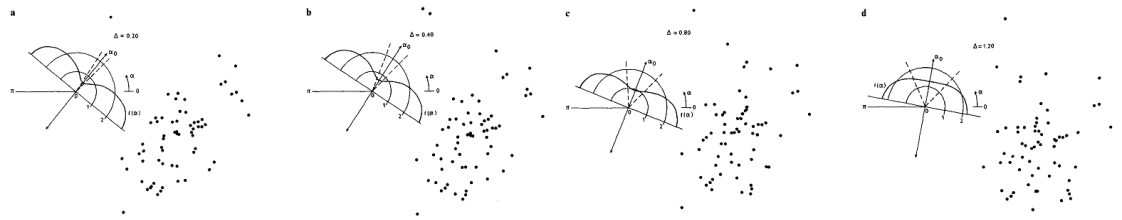


Figure 2.12: Correlated rotated dot patterns. Plots show the distribution of correlated dot pairs on the dot patterns, built by superimposing a rotated copy of a dot distribution on itself. Finding out the orientation of the rotated pattern on the last case is harder than on the first three. Source: [LMR89]

The efficiency of the visual system has also been studied in the context of correlated dot patterns perception by Ward [War85]. A random dot distribution only contains a specific amount of local orientation clues, which may or may not contribute to the perception of the global pattern. Experiments showed that the efficiency of the visual system at the perception of translated dot patterns decreases exponentially as the dot density increases.

2.6.2 Perception of correlated white noise patterns

As we analyzed the correlation plots in the previous part, several observations were worth noting. Indeed, It seems there is no direct mapping from the correlation coefficient to the exact texture appearance: correlated occurrences (echoes in the texture) are only perceived easily if they are close enough and correlation occurrences with values lower than approximately 0.6 are hard to perceive. Time also plays a role in perception, as correlated occurrences are easier to perceive after a focusing period.

These effects are visible in the following figures. In fig. 2.13, the same square region of size w pixels of white noise is tiled every w pixels. Increasing the value of w makes the perception of the repeating tiles considerably harder. We introduce $\mathbf{w} = \langle w; w \rangle$, the 2D vector representing the size of these tiles. This situation is similar to what happens in the blue region of the correlation plots in section 2.5: far away from the texture center, texture echoes are spread out and harder to perceive.

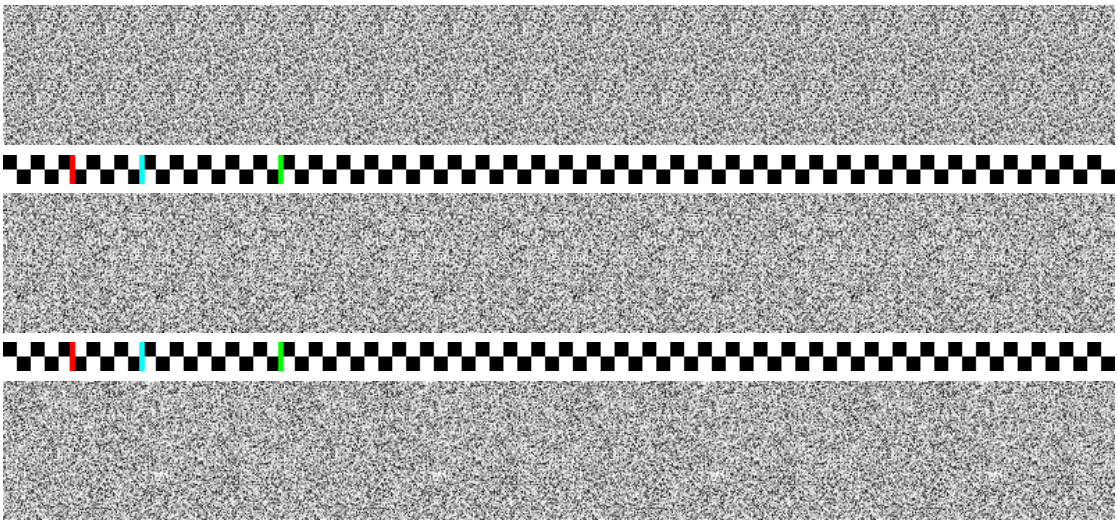


Figure 2.13: Correlated white noise tiles. Checkerboard squares are 10 pixels wide. The three white noise bands are respectively repeated every 50 pixels (red mark), 100 pixels (blue mark) and 200 pixels (green mark).

In fig. 2.14 however, w is fixed, but the repeated noise tiles are only partly correlated. This means that all tiles are not exactly the same, and include a certain amount of random uncorrelated noise. They are indeed blended with a random noise by a certain amount α. By changing the value of α, we can change the correlation amount in the resulting texture. Let's determine the correlation coefficient as a function of α. Let $X(\mathbf{x})$ be the random variable describing a tile point, and $Y(\mathbf{x})$ the random variable for the corresponding point in the random noise. $Z(\mathbf{x})$ is the result of blending $X(\mathbf{x})$ and $Y(\mathbf{x})$ in α proportion, with a subsequent linear contrast correction (see eq. (2.24)).

$$Z(\mathbf{x}) = \frac{\alpha X(\mathbf{x}) + (1-\alpha)Y(\mathbf{x})}{\sqrt{\alpha^2 + (1-\alpha)^2}} \tag{2.24}$$

$X(\mathbf{x})$ and $Y(\mathbf{x})$ are independent uniform variables over $[-1, 1]$ with variance σ. We can thus derive the following statistical quantities:

$$\begin{aligned} E\left[Z(\mathbf{x})\right] &= \frac{\alpha E\left[X(\mathbf{x})\right] + (1-\alpha)E\left[Y(\mathbf{x})\right]}{\sqrt{\alpha^2 + (1-\alpha)^2}} \\ &= 0 \end{aligned} \tag{2.25}$$

$$\begin{aligned} \operatorname{Var}\left(Z(\mathbf{x})\right) &= \frac{\operatorname{Var}\left(\alpha X(\mathbf{x}) + (1-\alpha)Y(\mathbf{x})\right)}{\alpha^2 + (1-\alpha)^2} \\ &= \frac{\alpha^2 \operatorname{Var}\left(X(\mathbf{x})\right) + (1-\alpha)^2 \operatorname{Var}\left(Y(\mathbf{x})\right)}{\alpha^2 + (1-\alpha)^2} \\ &= \sigma^2 \end{aligned} \tag{2.26}$$

Note that in order to express the correlation coefficient between two points $\mathbf{x}$ and $\mathbf{y}$, one must consider the repetition of the layer X, thus resulting in non-zero correlation if $\mathbf{x}$ and $\mathbf{y}$ are equal modulo the tile size $\mathbf{w}$. We then have:

$$\begin{aligned} \operatorname{Cov}\left(Z(\mathbf{x}), Z(\mathbf{y})\right) &= E\left[Z(\mathbf{x})Z(\mathbf{y})\right] - E\left[Z(\mathbf{x})\right] E\left[Z(\mathbf{y})\right] \\ &= \frac{E\left[\left(\alpha X(\mathbf{x}) + (1-\alpha)Y(\mathbf{x})\right)\left(\alpha X(\mathbf{y}) + (1-\alpha)Y(\mathbf{y})\right)\right]}{\alpha^2 + (1-\alpha)^2} \\ &= \frac{1}{\alpha^2 + (1-\alpha)^2}\big(\alpha^2 E\left[X(\mathbf{x})X(\mathbf{y})\right] + (1-\alpha)^2 E\left[Y(\mathbf{x})Y(\mathbf{y})\right] \\ &\qquad + \alpha(1-\alpha)\left(E\left[X(\mathbf{y})Y(\mathbf{x})\right] + E\left[X(\mathbf{x})Y(\mathbf{y})\right]\right)\big) \\ &= \frac{\alpha^2\left(\sigma^2 + \mu^2\right) + (1-\alpha)^2\mu^2 + 2\mu^2\alpha(1-\alpha)}{\alpha^2 + (1-\alpha)^2} \text{ for } \mathbf{x} \equiv \mathbf{y} \pmod{\mathbf{w}} \wedge \mathbf{x}, \mathbf{y} > \mathbf{w} \\ &= \sigma^2 \frac{\alpha^2}{\alpha^2 + (1-\alpha)^2}, \text{ as } \mu = 0 \end{aligned} \tag{2.27}$$

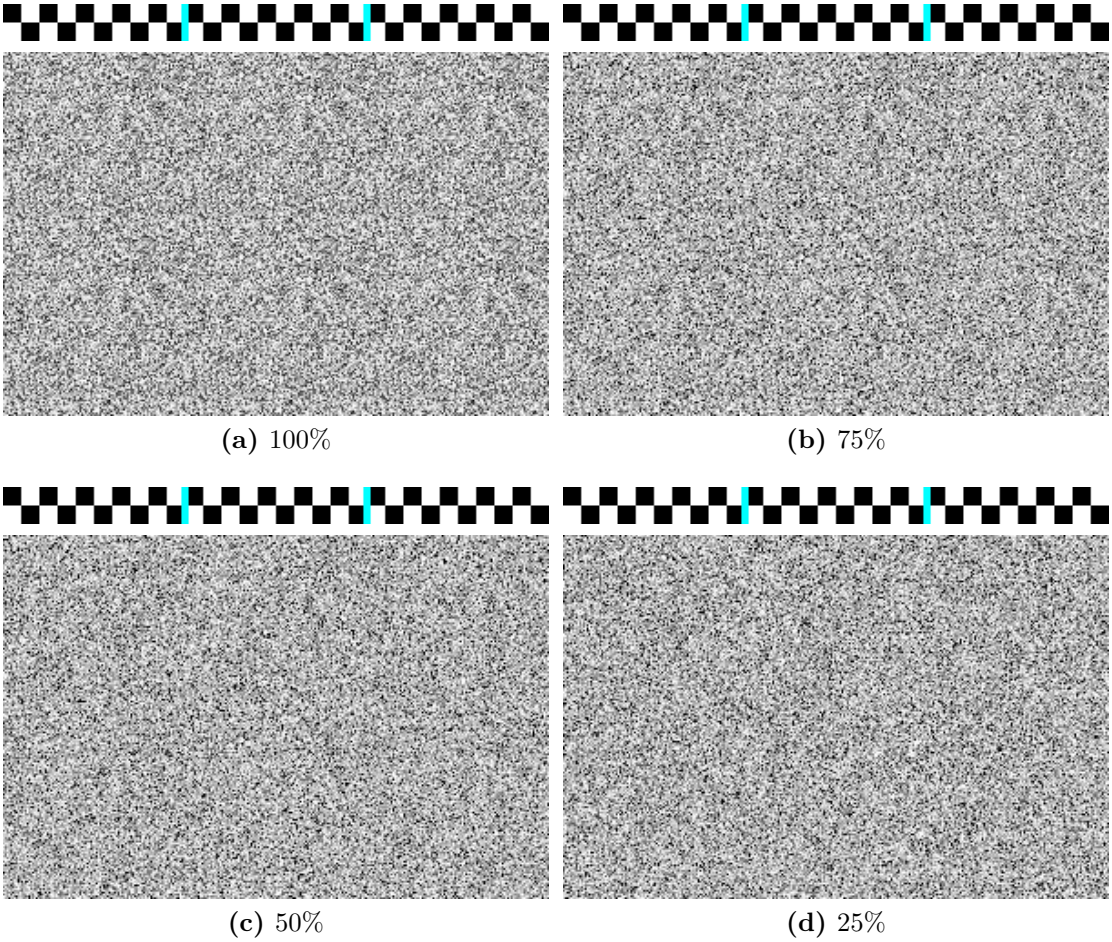
$$\begin{aligned}\operatorname{Corr}\left(Z(\mathbf{x}), Z(\mathbf{y})\right) &= \frac{\operatorname{Cov}\left(Z(\mathbf{x}), Z(\mathbf{y})\right))}{\sqrt{\operatorname{Var}\left(Z(\mathbf{x})\right)}\sqrt{\operatorname{Var}\left(Z(\mathbf{y})\right)}} \\ &= \frac{\alpha^2}{\alpha^2 + (1-\alpha)^2}\end{aligned} \tag{2.28}$$

The α blending coefficient thus directly controls the amount of correlation visible in the tiled textures. We use this in fig. 2.14 to generate more or less correlated tiled textures. As we can see, when the correlation coefficient is low (less than 0.5, or 50%), the repeated noise tiles become very hard to see.

These stimuli allow us to control two variables: the correlation distance and the correlation amount. They could be used as the basis for a perception experiment, in order to prove the existence of a correlation perception threshold in the visual system. Based on the perception experiments described in [War85] and [LMR89], we could setup an experience as follows:

- In a controlled environment (calibrated monitors, fixed viewing distance, controlled lighting), users would be shown white noise stimuli.
- These stimuli may either be raw uncorrelated white noise or tiled correlated white noise with varying correlation amounts.
- The stimuli would only last for a short amount of time, to prevent the user from focusing on the details and extracting higher-order information than what is available through correlation. Note that care must be taken when generating the tiled noise and displaying it, as the viewing window may have an influence on the perception of stimuli [DB02]. Between stimuli, the screen would be reset to average luminance to prevent retinal persistence.
- The user would then have to choose between two options:
  - There are no repeating patterns.
  - I have seen repeating patterns.
- By varying the parameters (tile size w and correlation amount) and collecting enough samples, we should be able to separate the influence of the correlation amount from the correlation distance.
- We could also analyze the white noise texture instances for distinct patterns through Fourier analysis, in order to qualify the efficiency of users in detecting correlated noise patterns (as Ward [War85] measured for translated dot patterns).

Depending on the results, we may or may not be able to validate the hypothesis that there is a distinct perceptive threshold on the distance between two correlated occurrences of a pattern, and in the amount of correlation between these two patterns. Note that white noise is a lot more perceptually rich than dot patterns, thus we would not be able formulate hypothesis about the psychophysical implications of these results.



(a) 100% **(b)** 75%

(c) 50% **(d)** 25%

Figure 2.14: Correlated white noise tiles. A tiled noise is repeated every 100 pixels (blue mark), but blended with an uncorrelated white noise. The 100% correlated instance (a) is the same as in figure 2.13, while a 0% correlated instance is just white noise (not shown here). Noise tiling is not perceptible in the 25% case. It is however perceptible in the 50% case, after a focusing period. Note that the visual system may use clues present in surrounding textures to infer the tiling in this case.

2.7 Extending the correlation tool

2.7.1 Extensions to non-white noise textures

We conducted all our studies above with white noise for simplification, but as we mentioned in the introduction, artists often use more complex noise primitives. More complex noise primitives have more structure than white noise, and as such they exhibit autocorrelation over larger areas, whereas in the white noise case the autocorrelation function is only a Dirac. Since the base-noise statistics will interfere, we have to study it as well. In this section, we proceed a fast overview to illustrate this.

In section 2.5 we used statistically computed 2D maps of the correlation coefficient in order to show the absence of non-radial correlation (fig. 2.8). These maps were generated interactively, using the mouse cursor to choose a point in the texture, the result showing the correlation coefficient between the chosen point and the rest of the texture. This is interesting because this does not require an analytical expression of the correlation coefficient to be computed. In particular, we can use this tool to view the correlation using base textures which are not white noise.

An extreme case of non-white noise texture is real world textures[6]. In fig. 2.17, we use several image textures (bricks, iron, pebbles, stone and wood) and study their look after using them as the base texture for the non-sliding stationary zooming process presented in this work. Figure fig. 2.15 show the autocorrelation of the base image textures used in fig. 2.17. We can see that textures with regular structures such as the bricks and pebbles textures suffer from this process: ghosting artifacts resulting from blending multiple layers are clearly visible.

However, the iron, stone and wood textures offer good-looking results. As we built this stationary zooming process using the properties of white noise, textures highly stochastic (ie. with a rich spectrum as white noise has) keep their appearance after the fractalization process. This is coherent with the fractalization survey done by Bénard, Thollot, and Sillion [BTS09] in the context of non-photorealistic rendering, and known limitations of the texture advection process introduced by Yu et al. [Yu+11].

The results are not perfect though, as the frequency distribution of these real textures is denser towards low frequencies. In the case of the iron texture, an example is the rusted area. Such low frequencies induce temporal inconsistencies because the visual system is capable of tracking large shapes in image, which then disappear (or appear) as the scaled layers are regenerated. The resulting texture frequency spectrum may help characterizing these artifacts, as in fig. 2.16. Note that textures which do not wrap will also introduce visible spatial discontinuities.

[6] As this on-going study will be completed during the remaining internship time with the study of the common noise primitives (Perlin, Worley, Gabor, etc.)

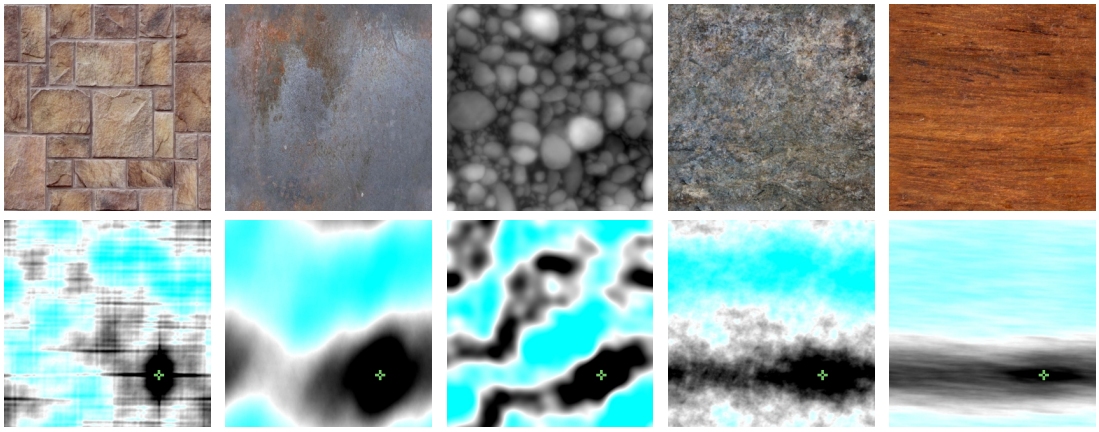


Figure 2.15: Autocorrelation in image textures. Top: base image texture. Bottom: correlation between the texture image and the reference point in the bottom right quadrant. In correlation plots, the color scale ranges from cyan (-1) to white (0) to black (1).

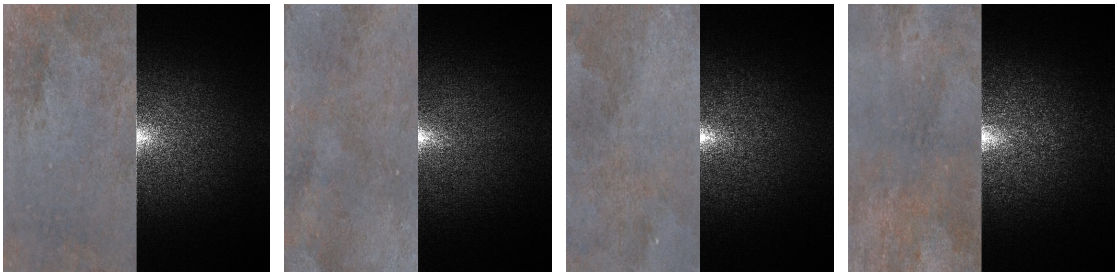


Figure 2.16: Temporal coherence in fractalized iron texture. Note how the low frequency lobe shape (center of the images) gets smaller over time, particularly in the 3rd frame. Here 8 layers have been used with a random offset.

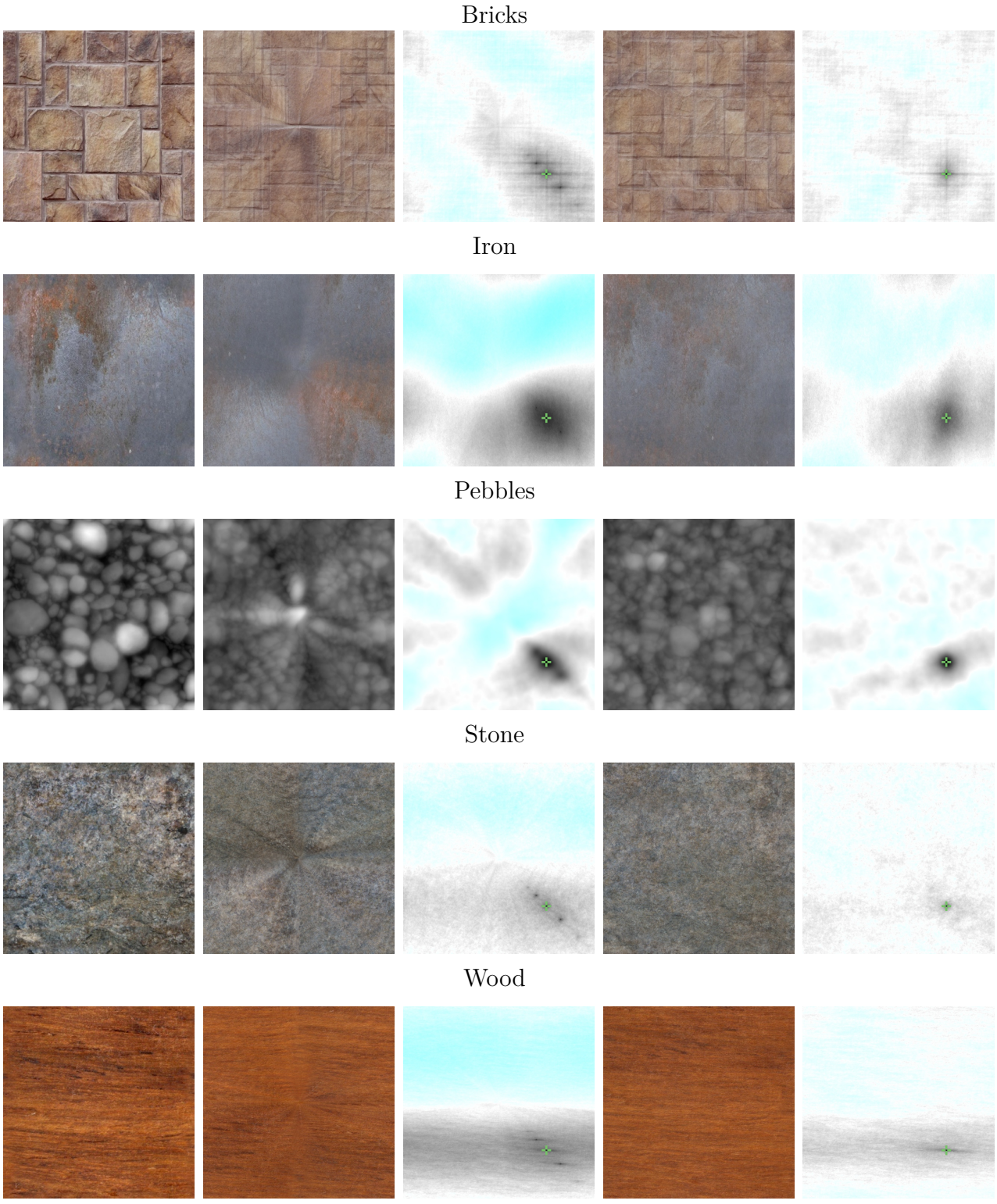


Figure 2.17: Correlation using real-world textures. From left to righ: original texture, fractalized texture with no offset and its correlation plot, fractalized texture with random offset and its correlation plot. Color textures are converted to gray for computing the correlation coefficient using the luminosity formula $I(r, g, b) = 0.21r + 0.72g + 0.07b$. In correlation plots, the color scale ranges from cyan (-1) to white (0) to black (1). The reference point is marked by a green cross. All fractalized textures use 8 layers. Source textures from https://shadertoy.com.

With these textures, the correlation coefficient plots are harder to exploit. Indeed, the local autocorrelation of the source texture interferes with the correlation introduced by the fractalization process. It can be seen in anti- correlated areas (cyan areas in fig. 2.17), which can only be the consequence of anti-correlation in the source texture. In the case of white noise, the local autocorrelation was only a Dirac distribution, which did not exhibit this problem (see fig. 2.18).

(a) Fractalized noise **(b)** Correlation plot **(c)** Fractalized iron **(d)** Correlation plot

Figure 2.18: Correlation and anti-correlation in fractalized textures. Note how the correlation plot for the fractalized iron texture is way different from the fractalized white noise correlation plot, although the process is the same (8 layers, no offset). The circled area is centered on the reference point, thus high correlation is mostly located inside it.

2.7.2 Extensions to texture patterns

Up to know we have studied many kinds of spatial and temporal artifacts concentrated on one single test-bed example: the fractalized zoom texture. But other kind of artifacts can occur in different use-case of procedural textures: for instance, marble-like pattern is generally obtain by disturbing a basic vein pattern with noise displacing it stochastically. We will study the artifacts occurring when varying the scaling parameter of noise in space or time (see fig. 2.19).

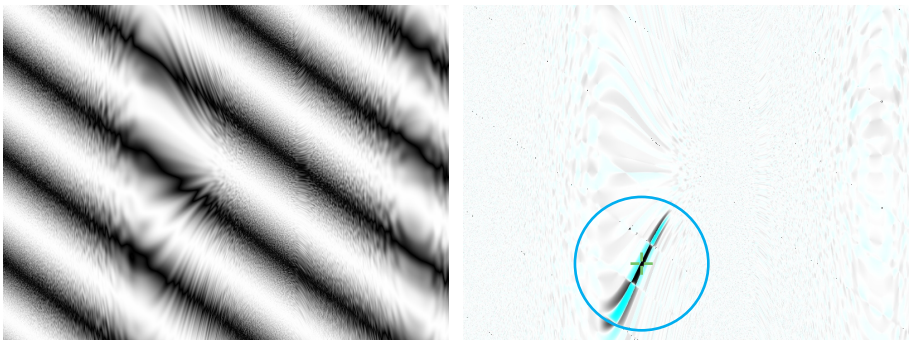
Texturing artifacts can arise from the continuous evolution of some procedural texture parameters in space or time. At design time, an artist can also experience distracting artifacts while interactively tuning procedural texture parameters, resulting in harder texture authoring processes.

We thus want to apply our various study tools to this new case[7]. First, we wanted to extend correlation tools to this kind of artifacts.

The texture in fig. 2.19 has been generated by using the value of a gradient noise[8] to distort a band pattern, in order to produce a marble-like texture. Formally,

[7] Note that this is still on-going work that will be completed during the remaining internship time, so here we quickly overview some aspects

[8] A gradient noise is created by choosing random gradient at points of a lattice defined over the texture space, and interpolating between them. Perlin noise is an example of gradient noise.



(a) Marble texture **(b)** Correlation plot

Figure 2.19: Perturbation scaling problems. Here, a spatial evolution of the scale of a noise. Source: http://bit.ly/st-MlGGW1. The reference point in the correlation plot is marked by a green cross.

let $noise : \mathbb{R}^2 \to [-1, 1]$ be the noise function, and $pattern : \mathbb{R} \to \mathbb{R}$ a function that produces the marble aspect, such as black and white bands. This process is described by the following equation:

$$I(\mathbf{x}) = pattern\left(\mathbf{a}.\mathbf{x} + bnoise\left(\frac{\mathbf{x}}{2^l}\right)\right) \tag{2.29}$$

$\mathbf{a}$ is a vector that can be used to change the direction of the pattern, and b a real number which controls the amount of pattern distortion caused by the noise. 2^l controls the scale of the noise "granularity". The naive method for evolving the noise scale in 2.29, as used in fig. 2.19, is to make l a space-dependent parameter, e.g. for $m, n, q \in \mathbb{R}$, $l = m + n\cos(q\mathbf{x}_x)$. Doing so result in very unpleasant visual artifacts (fig. 2.19a): instead of the expected granularity scaling (see fig. 2.20a) some veins of the marble pattern are abnormally stretched along specific paths.

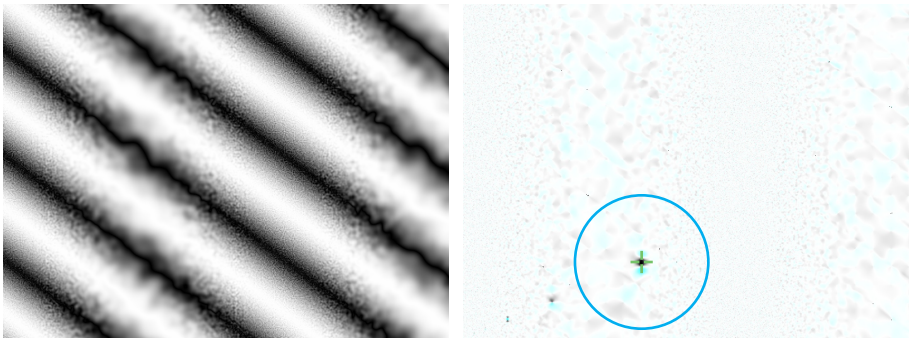
In fig. 2.19b, we chose the reference point for the correlation computation to be placed on a stretched marble vein. Since a visually stretched area correspond to correlation, it is represented by a black area in the correlation plot. However, a very distinct pattern shows up in the plot: strongly anti-correlated areas are present around the correlated area of the reference point. Even after fixing the stretching artifact (by introducing discontinuities in l and blending the resulting noise), this "checker" pattern is still visible, although without stretching (fig. 2.20). In fact, this pattern comes from the underlying noise we used to perturb the marble pattern. In fig. 2.21, we plot the correlation coefficient of a gradient noise texture for two different reference points. Since this gradient noise[9] relies on a non-convex interpolation of grid values, it is equivalent to a filtering kernel with negative lobes.

[9] See http://bit.ly/st-lsffzj for a code to generate this kind of noise.

This shows the readability limits of the correlation tool: if we simply compute the correlation coefficient of the process, we get a description of the whole process. In the case of a marble texture, veins are expected to be correlated to some extent, and noise patterns are usually chosen for their appearance by artists. These noise patterns are not white noise, and thus introduce correlation. This correlation, which we are not interested in for determining the presence of artifacts, pollutes the plot and makes it hard to draw conclusions.

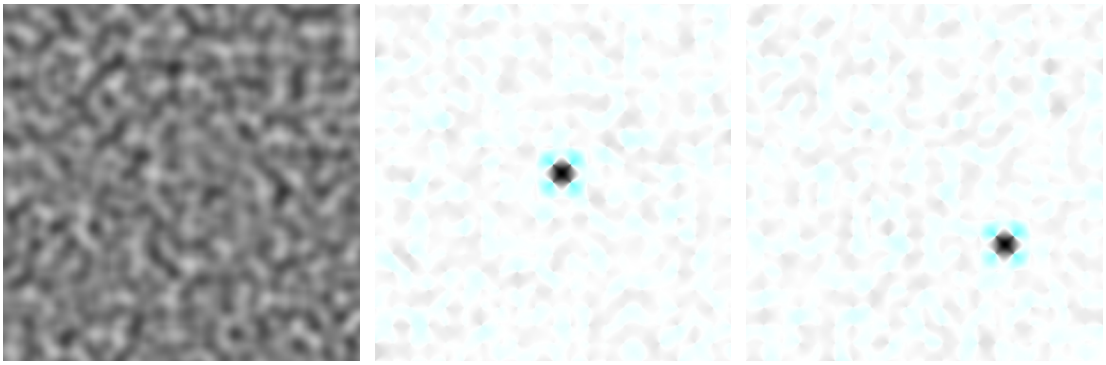
Future work on the exploitation of the correlation coefficient would probably involve computing it for different steps in the texturing algorithm. In the case of our present study, namely, the stretching artifact in the marble texture, we clarify the measure by computing the correlation coefficient for the noise scaling operation, independently of the vein pattern and of the underlying noise.

Note that in the case of directional artifacts, the power spectrum can be used to detect anisotropy and changes in spatial frequency distribution, as can be seen in fig. 2.22. This lead has not been explored at the time of writing.



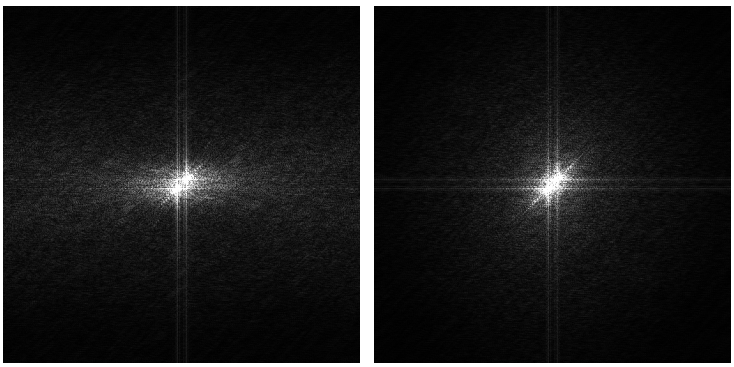
(a) Marble texture **(b)** Correlation plot

Figure 2.20: Correlation in fixed marble texture. Even though no stretching occurs, a distinct anti-correlated pattern is visible around the reference point (circled in blue).



(a) Gradient noise **(b)** Correlation plots

Figure 2.21: Correlation in gradient noise. The anti-correlation pattern around the reference point is very similar to the one visible in fig. 2.20b. Gradient noise source: http://bit.ly/st-lsffzj.



(a) Stretched marble **(b)** Non-stretched marble

Figure 2.22: Power spectrum of marble textures. As stretching artifacts are directional, we can see that the power spectrum of the stretched variant is anisotropic, while the power spectrum of the non-stretched variant is isotropic.

2.8 Temporal coherence in texture patterns

In section 2.7.2, we introduced a marble texture whose veins were scaled depending on their location within the texture. Formally, the noise used to perturb the band pattern was scaled as a function of the texture coordinates. This spatial evolution of a noise scale introduced stretching which altered the texture appearance.

Another possibility is to have the scale of this noise evolve over time. Figure 2.23 shows a few frames from such an animation, however we advise the reader to follow the link to view the live animation. As with the spatial evolution of the noise scale, the naive method for scaling over time introduces artifacts. In this case, a sliding artifact is introduced.

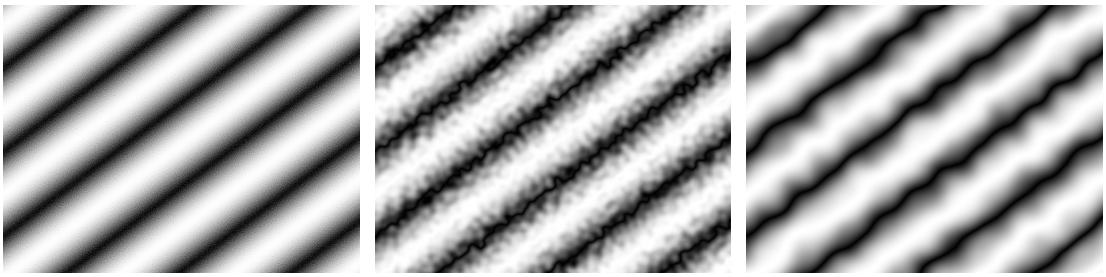
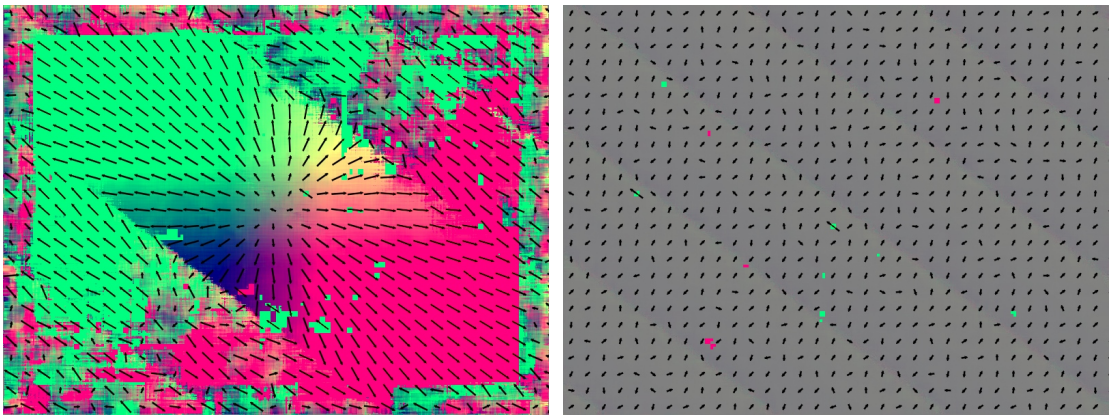


Figure 2.23: Temporal scaling of a marble texture. See http://bit.ly/st-llGGW1 for a live animation exhibiting the sliding artifact, and a solution which does not exhibit this artifact.

As this procedural texture is not built using layers, the sliding artifact can not be described by evaluating the speed of a given point in different layers. Instead, what we see is the patterns introduced by the noise function which move continuously over the band pattern over time.

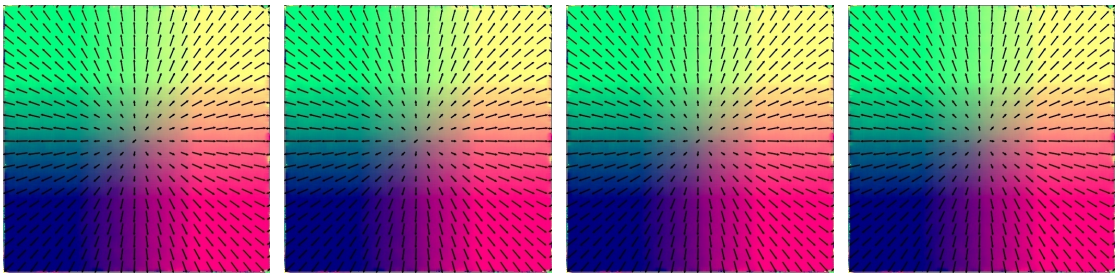
During the remaining time for this project (after the time of writing), we could introduce a measure of the unwanted motion in the resulting texture, we could use the optical flow. Usually used in computer vision, optical flow algorithms determine the motion field of an image sequence by matching gray levels under a brightness constancy constraint. However, we can run an optical flow algorithm on the animated texture to describe its motion (see fig. 2.24).

Note that the type of artifacts that can be described using an optical flow characterization is yet to be defined, but would have to take into account the constraints of such algorithms. Indeed, the optical flow of the stationary white zoom texture is always globally the same, independently of the sliding and correlation artifacts (see fig. 2.25). Finer statistical tests would have to be introduced to deduce properties from the results of the optical flow in that case.



(a) Sliding marble **(b)** Non-sliding marble

Figure 2.24: Optical flow in a marble texture. The use of optical flow in artifact analysis is still a work in progress and requires more tuning to remove the noise in the results. Color and arrow directions represent the local orientation of the motion field, while saturation and arrow length indicate its magnitude. We can see that the optical flow of the sliding marble texture shows an outward motion, while the non-sliding marble texture does not exhibit any motion.



(a) Sliding, no offset **(b)** Non-sliding, no offset **(c)** Non-sliding, non-linear offset **(d)** Non-sliding, random offset

Figure 2.25: Optical flow in a stationary zooming texture. The result of the optical flow is globally the same on all variants of the stationary zooming texture, although we perceive the resulting textures differently, especially in the case of the variant exhibiting the sliding artifact.

3 Implementation

This section discusses the tooling requirements for rendering and analyzing textures, which lead me to implement a desktop application to produce the results for this report.

3.1 Rendering procedural textures

Efficient rendering of procedural textures is best achieved using hardware acceleration, through the OpenGL API to use the available graphic hardware. Textures can be implemented as *fragment shaders*, and rendered to the screen without any transformations. Complex textures may use multiple render buffers and multiple fragment shaders, but the process stays the same. See https://www.opengl.org/ for the full OpenGL documentation.

However, a fragment shader alone is not an executable program in itself, and must be compiled and run in a suitable OpenGL context created by an application, referred to as "the host application". A typical OpenGL context is created using the GLUT library on Linux, or more modern alternatives such as GLFW3.

Another alternative for creating an OpenGL context is the new WebGL API, designed to provide support for JavaScript-driven OpenGL inside modern web browsers. This technology is in fact in use at https://www.shadertoy.com/, a website for sharing shader programs. It can be used to quickly test and share texturing algorithms, as a suitable entry point is already set up with the right inputs (texture coordinates), outputs (fragment color), and the appropriate uniforms (time, screen resolution, etc.). Some of the figures from this document have been rendered using the ShaderToy website, and we provide the links so the reader can interact with them. A recent browser is recommended (latest versions of Mozilla Firefox or Google Chrome).

3.2 Analyzing rendered textures

Although GLSL is a very expressive language, not all algorithms efficiently map to shader code. For example, if we want to compute the optical flow of the result of a shader program, a preferred alternative is to use dedicated code for this computation, such as what is provided by the OpenCV libraries. These libraries

are the de facto standard for computer vision and image manipulation, providing both CPU and GPU[1] algorithm implementations.

During this internship, I developed the *GLcv* tool, a desktop application written in C++14 (in approximately 4000 lines) that:

- Can host GLSL programs while being compatible with the ShaderToy host specifications.
- Integrates GPU-accelerated analysis and computer-vision algorithms through the use of OpenCV.
- Provides various facilities for rendering animations to files, to include images in this report or if further analysis is needed.

The main advantage of running analysis algorithms "on-line" (as opposed to off-line, ie. on rendered frames stored on disk) is that this allows modifying texture parameters and directly observing their effect on the property we are visualizing through the analysis algorithm. This also allows interactive experiments, and exploring multidimensional spaces such as the correlation in textures (which is 4D, since it is defined between two 2D points). This also is globally faster, since rendered textures on the GPU using OpenGL don't have to be moved back to the host memory[2].

These requirements lead to the design of the rendering and analysis pipeline in fig. 3.1. This pipeline has been implemented using the libraries described in table 3.1. Multiple view modes of the different outputs are supported to enable the user of this tool to visually match features in the texture to features in the algorithm view. See fig. 3.2 for an overview of these modes. CMake has been used to automate the configuration process, as well as Doxygen to provide a documentation framework for possible future developments of the tool. All the source code and associated configuration files are under the Git version control system.

OpenCV algorithms are implemented as functions of the current texture frame called the "input frame" and the previous input frame called the "last frame", producing an "output frame" which is displayed on the render target. These frames are GPU multi-dimensional arrays, manipulated using the OpenCV library which itself uses CUDA to perform computations efficiently and asynchronously. This simple structure allows for good extensibility, as adding new algorithms is done by implementing the corresponding wrapper functions. The user just specifies which algorithms should be run when starting the program.

If application-specific algorithms (ie. not implementable in terms of OpenCV provided algorithms) were to be needed, access to the raw data on the device memory

[1] GPU support based on the Nvidia CUDA toolkit.

[2] Early versions of *GLcv* were affected by a bug in OpenCV which caused textures to be moved to the host memory and back to the device memory instead of performing a GPU-GPU copy, leading to a 3x slowdown.

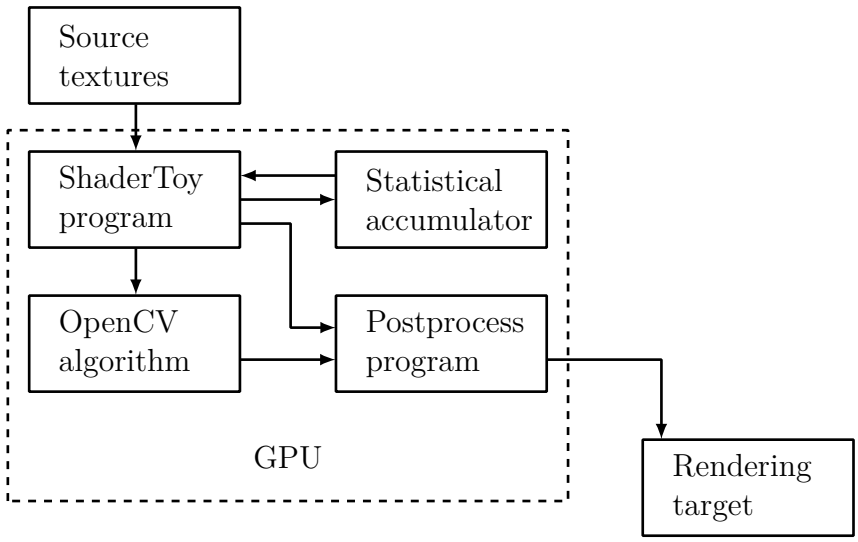


Figure 3.1: Our *GLcv* rendering pipeline. The main rendering steps are all executed on the GPU, and resulting textures can be fetched to the CPU to be exported as files. See section 3.3 for a description of the statistical accumulator.

is still possible, through the use of the CUDA runtime. At the time of writing, this has not yet been the case.

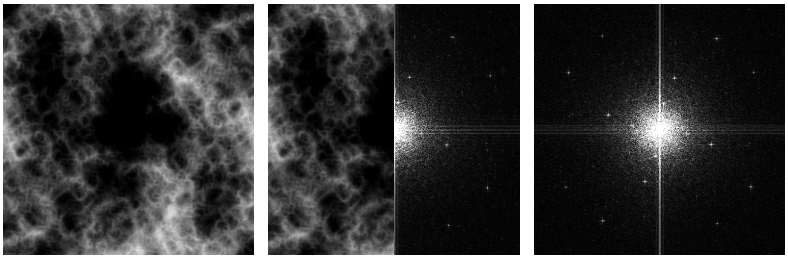
The list of algorithms I have currently integrated in *GLcv* is reproduced in table 3.2, along with their measured speed in terms of frames per second, and compared to the processing speed without running an analysis algorithm. Special care has been taken to ensure no unnecessary performance hits are present in the rendering pipelines:

- Memory allocations for OpenCV algorithms are done only once, and reused for each frame.
- Only two copies are performed between different memory types for each frame: one between the texture memory (OpenGL) and the device memory, and one back to the texture memory. This is required because texture memory is distinct from general purpose memory as required by CUDA. However this copy is fast when performed using "device-to-device" (ie. on GPU) copy.

Other visualizations have been implemented using the statistical accumulator (presented in section 3.3), which allow visualizing statistical properties of a texture such as its mean, variance, covariance and correlation. A shader program is responsible for computing these values from the contents of the statistical accumulator, thus giving this tool great versatility for building visualizations.

Library	Usage	License
OpenCV 2	Vision and image processing algorithms	BSD License
GLEW	OpenGL extension loader	MIT
oglplus	C++ OpenGL wrapper	Boost Software License
Boost	C++ utilities	Boost Software License
CUDA	Nvidia GPU computing	CUDA Toolkit EULA
GLFW3	OpenGL context creation library	zlib/libpng License
nanogui	OpenGL widget library	BSD License

Table 3.1: Libraries used in *GLcv*



(a) Texture **(b)** Split **(c)** Algorithm

Figure 3.2: *GLcv* view modes. The algorithm used here is a representation of the texture power spectrum.

Name	Implementation	FPS	Speed
None	No processing nor copy	124.6	1.00x
Copy	Copy input frame to output	73.50	0.59x
Spectrum	FFT spectrum magnitude	67.53	0.54x
Optical flow	Lukas-Kanade optical flow	30.62	0.25x

Table 3.2: *GLcv* implemented algorithms. The texture used in these tests is an animated combination of Perlin noise layers. FPS values are averaged over 1000 frames, on an Intel Xeon W3520/Nvidia GTX 670 computer with 8GB RAM.

3.3 Computing the correlation coefficient over texture instances

In section 2.5 we studied the statistical correlation coefficient between the values of two points in a texture to produce correlation plots. Here, we detail how this can be achieved efficiently on the GPU.

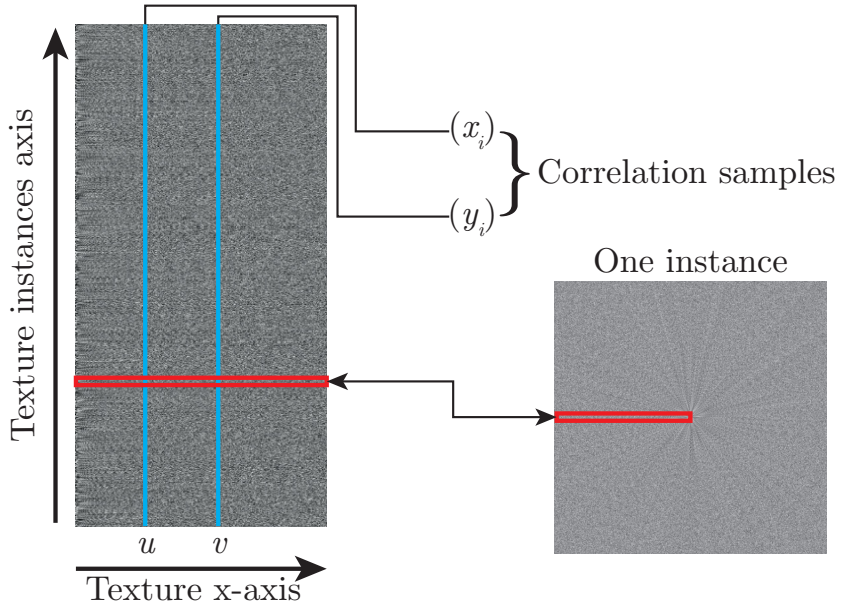
An estimator for the correlation coefficient between samples $x = (x_i)_{i\in[0,n[}$ and $y = (y_i)_{i\in[0,n[}$ is given by the following equation. $\bar{x}$ (resp. $\bar{y}$) designates the empirical mean of (x_i) (resp. (y_i)).

$$r_{xy} = \frac{\frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2}\sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \bar{y})^2}} \tag{3.1}$$

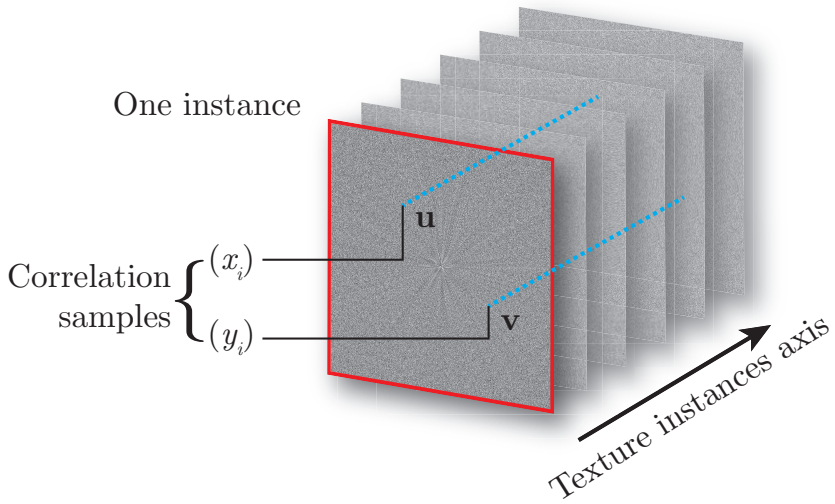
In our case (x_i) and (y_i) are the intensity values over n instances of the base texture. If the corresponding points are described by only one coordinate, such as when computing $\mathrm{Corr}\left[I\left(\langle x;0\rangle\right), I\left(\langle y;0\rangle\right)\right]$, we use the following strategy (see fig. 3.3a). A texture image is generated in which each row represents the intensity values for a given instance, and each column u is a sample of the texture process at a given coordinate u, ie. a sample of $I\left(\langle u;0\rangle\right)$. By computing eq. (3.1) for u and v being two columns in this texture, and (x_i) and (y_i) the respective samples, we obtain the correlation coefficient between $I\left(\langle u;0\rangle\right)$ and $I\left(\langle v;0\rangle\right)$ using n samples.

However, an issue arises if we want to compute the correlation coefficient between two 2D points $\mathbf{x}$ and $\mathbf{y}$, since each point has now an extra degree of freedom. With 1D coordinates, we relied on 2D textures to store samples. With 2D coordinates, we thus add one dimension, thereby introducing 3D textures for storing samples. Each of the n layers in the generated 3D texture is now a complete instance of the resulting image. In order to collect the samples (x_i) and (y_i), we simply accumulate the samples through all texture instances, and then compute the correlation coefficient value.

In our *GLcv* rendering pipeline, the statistical accumulator is responsible for storing the instances as layers of a 3D texture. Note that the 2D correlation variant (ie. for 1D coordinates) only requires traditional textures.



(a) 2D correlation



(b) 4D correlation

Figure 3.3: Implementations of correlation computation. (a) shows the generated 2D sampling storage with the stationary zoom texture samples. u and v are the coordinates of the two points for which we want to compute the correlation coefficient. The corresponding columns are highlighted in blue. (b) is a representation of how the concept introduced in (a) extends to 4D. The two columns u and v are replaced by two "rays" at locations **u** and **v** which traverse all layers of the 3D texture. A particular instance of the texture is highlighted in red.

4 Conclusion

The study of artifacts is a vast domain, which holds much more to be discovered. In this work, we chose a simple example of a procedural texture with notable uses in non-photorealistic rendering for adding temporally coherent detail.

This texture has a simple statistical description: white noise, which can be seen as a set of uniform random variables, can easily be described mathematically. This allowed us to use the power spectrum to describe the temporal continuity of the resulting texture. We also modeled the sliding artifacts to remove them, before describing the statistical properties of the resulting texture.

The study of statistical properties of a procedural texture posed the question of the probability space to choose. We found out that the most practical space to work with is the entire space of texture realizations. Statistical descriptors built on that space thus describe properties of the texturing process, instead of a particular instance.

We compared different methods for restoring the variance of the resulting texture, using either a linear function, a polynomial function or a sigmoid. Each method required compromises between the contrast restoration, the introduction of clamping, and the distortion of the histogram.

We also studied the link between the correlation coefficient between two points in the resulting texture and how it describes unwanted self-similarities in the rendered result. As extensions of this work, we gave a fast overview of the use of the correlation coefficient tool on other types of textures, though more work on this subject will be carried out after the time of writing.

The correlation coefficient revealed to be a perceptually sound artifact, as early perception experiments provided strong clues that the human visual system may indeed compute local autocorrelation in early stages of vision. We propose a draft for a perception experiment to better understand how we perceive correlation on white noise, on the same principles as previous perception experiments.

The study of texturing artifacts in terms of descriptors, either statistical or of higher order such as the power spectrum, required a way to visualize those descriptors alongside a texture instance. We implemented these visualizations as interactive programs so we could visualize the influence of changing parameters on both the texture and the artifact. We used accelerated graphics hardware to efficiently compute statistical descriptors on textures, following our defined probability space of texture instances.

We propose here leads for future work on the domain of the study of rendering artifacts:

- White noise has a simple statistical description, however more complex noises are usually used by computer graphics artists. It would be interesting to see how the method introduced here extends to noises such as Perlin noise, Worley noise or Gabor noise. As we quickly noted using real-world textures, the correlation coefficient is harder to exploit, which may require more complex descriptors. They would however probably be built on the correlation coefficient, as it has a perceptual significance.
- The stationary zooming texture is an example with a simple mathematical description. However it exhibits blending artifacts which are present in a lot of texturing scenarios, which artists often have to work around. It may be interesting to apply the fixes we propose here to blending artifacts as part of other texturing processes.
- We used the power spectrum of resulting textures to quickly describe the frequency distribution of an instance. Describing the power spectrum of a texture process instead of an instance may reveal artifacts independently of an instance, a process which has proved to be successful for the study of statistical descriptors of the stationary zooming texture.
- In section 2.8 we quickly introduced the use of an optical flow algorithm to detect unwanted motion. The optical flow as well as other more advanced computer vision tools could be used to describe artifacts in textures, which is a field we believe should be explored.

Bibliography

[BBT11] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. "State-of-the-Art Report on Temporal Coherence for Stylized Animations". In: *Computer Graphics Forum* 30.8 (Dec. 2011), pp. 2367–2386. DOI: 10.1111/j.1467-8659.2011.02075.x. URL: https://hal.inria.fr/inria-00636210.

[Bou+06] Adrien Bousseau, Matthew Kaplan, Joëlle Thollot, and François X. Sillion. "Interactive watercolor rendering with temporal coherence and abstraction". In: *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. Annecy, France: ACM, 2006. URL: https://hal.inria.fr/inria-00510223.

[BTS09] Pierre Bénard, Joëlle Thollot, and François X. Sillion. "Quality Assessment of Fractalized NPR Textures: a Perceptual Objective Metric". In: *6th Symposium on Applied Perception in Graphics and Visualization (APGV)*. Chania, Crete, Greece: ACM Press, Sept. 2009, pp. 117–120. DOI: 10.1145/1620993.1621016. URL: https://hal.inria.fr/inria-00405966.

[CD05] Robert L. Cook and Tony DeRose. "Wavelet Noise". In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. Los Angeles, California: ACM, 2005, pp. 803–811. DOI: 10.1145/1186822.1073264. URL: http://doi.acm.org/10.1145/1186822.1073264.

[Cun+03] Matthieu Cunzi, Joëlle Thollot, Sylvain Paris, Gilles Debunne, Jean-Dominique Gascuel, and Frédo Durand. "Dynamic Canvas for Immersive Non-Photorealistic Walkthroughs". In: *Proc. Graphics Interface*. Halifax, Canada: A K Peters, LTD., 2003. URL: https://hal.inria.fr/inria-00510176.

[DB02] S.C Dakin and P.J Bex. "Summation of concentric orientation structure: seeing the Glass or the window?" In: *Vision Research* 42.16 (2002), pp. 2013–2020. ISSN: 0042-6989. DOI: https://doi.org/10.1016/S0042-6989(02)00057-3. URL: http://www.sciencedirect.com/science/article/pii/S0042698902000573.

[Fer+97] James A. Ferwerda, Peter Shirley, Sumanta N. Pattanaik, and Donald P. Greenberg. "A Model of Visual Masking for Computer Graphics". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 143–152. ISBN: 0-89791-896-7. DOI: 10.1145/258734.258818. URL: http://dx.doi.org/10.1145/258734.258818.

[Gal+12] Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. "Gabor Noise by Example". In: *ACM Transactions on Graphics* 31.4 (July 2012), Article No. 73. DOI: 10.1145/2185520.2185569. URL: https://hal.archives-ouvertes.fr/hal-00695670.

[Gla02] Leon Glass. "Looking at Dots". In: *Mathematics Intelligencer* 24.4 (2002), pp. 37–43.

[Gla69] Leon Glass. "Moire Effect from Random Dots". In: *Nature* 223.5206 (Aug. 1969), pp. 578–580. DOI: 10.1038/223578a0. URL: http://dx.doi.org/10.1038/223578a0.

[GZD08] Alexander Goldberg, Matthias Zwicker, and Frédo Durand. "Anisotropic Noise". In: *ACM SIGGRAPH 2008 Papers*. SIGGRAPH '08. Los Angeles, California: ACM, 2008, 54:1–54:8. ISBN: 978-1-4503-0112-1. DOI: 10.1145/1399504.1360653. URL: http://doi.acm.org/10.1145/1399504.1360653.

[Lag+09] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. "Procedural Noise using Sparse Gabor Convolution". In: *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2009)* 28.3 (July 2009), pp. 54–64. DOI: 10.1145/1531326.1531360.

[LMR89] Petr Lánsky, J. Mates, and T. Radil. "On the visual orientation of random dot Moiré patterns". In: *Biological Cybernetics* 60.3 (1989), pp. 213–219. ISSN: 1432-0770. DOI: 10.1007/BF00207289. URL: http://dx.doi.org/10.1007/BF00207289.

[MN88] Don P. Mitchell and Arun N. Netravali. "Reconstruction Filters in Computer-graphics". In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '88. New York, NY, USA: ACM, 1988, pp. 221–228. ISBN: 0-89791-275-6. DOI: 10.1145/54852.378514. URL: http://doi.acm.org/10.1145/54852.378514.

[Ney03] Fabrice Neyret. "Advected Textures". In: *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. Ed. by D. Breen and M. Lin. San diego, United States: Eurographics Association, July 2003, pp. 147–153. URL: https://hal.inria.fr/inria-00537472.

[Per85] Ken Perlin. "An Image Synthesizer". In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '85. New York, NY, USA: ACM, 1985, pp. 287–296. ISBN: 0-89791-166-0. DOI: 10.1145/325334.325247. URL: http://doi.acm.org/10.1145/325334.325247.

[Sal+15] Rohini Salunke, Dipali Badhe, Vanita Doke, Yogeshwari Raykar, and Bhushan S. Thakare. "The State of the Art in Image Compression". In: *International Journal of Advanced Research in Computer and Communication Engineering* 4.2 (Feb. 2015). DOI: 10.17148/IJARCCE.2015.

42109. URL: http://www.ijarcce.com/upload/2015/february-15/IJARCCE109.pdf.

[She64] Roger N. Shepard. "Circularity in Judgments of Relative Pitch". In: *The Journal of the Acoustical Society of America* 36.12 (1964), pp. 2346–2353. DOI: 10.1121/1.1919362. eprint: http://dx.doi.org/10.1121/1.1919362. URL: http://dx.doi.org/10.1121/1.1919362.

[War85] Roger Ward. "The Location of Noisy Visual Stimuli". In: *Canadian Journal of Psychology* 39.3 (1985), pp. 387–399.

[Wor96] Steven Worley. "A Cellular Texture Basis Function". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 291–294. ISBN: 0-89791-746-4. DOI: 10.1145/237170.237267. URL: http://doi.acm.org/10.1145/237170.237267.

[Yu+11] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. "Lagrangian Texture Advection: Preserving both Spectrum and Velocity Field". In: *IEEE Transactions on Visualization and Computer Graphics* 17.11 (Nov. 2011), pp. 1612–1623. DOI: 10.1109/TVCG.2010.263. URL: https://hal.inria.fr/inria-00536064.

Glossary

Device memory refers to the memory embedded on the GPU board, in a usual computer architecture. The GPU is seen as the "device" as opposed to the "host", the CPU. 60, 61, *see also* host memory

FPS Frames per second. 62

Host memory refers to the RAM in a usual computer architecture. This term comes from the abstraction where the CPU (and the RAM) are the "host" for GPU-accelerated computations, sending data and instructions to the "device", the GPU. 60

LUT Lookup Table. 11, *Glossary:* lookup table

Non-photorealistic rendering is a class of rendering techniques whose aim is not to reproduce life-like images, but instead to depict a scene using a specific art style, such as painting, pencil, etc. which may include a supporting texture (paper, canvas, etc.). 11, 20, 22, 23, 50, 65

NPR Non-photorealistic rendering. *Glossary:* non-photorealistic