Studiengang Informatik

Fakultät Elektrotechnik und Informatik

Bachelor Thesis

to the subject

Analysis of procedural Materials

within Fragment-Shader

Eric Dolch

Malerstraße 21

88250 Weingarten

Adviser:

Prof. Dr. Daniel Scherzer - Hochschule Ravensburg-Weingarten

Dennis Reimer - Hochschule Ravensburg-Weingarten

**Statutory Declaration**

I hereby declare, that the present thesis was written and developed self-reliant and under exclusive use of the stated literature and resources. The thesis was not submitted in the same or similar form or in extracts to any testing authority yet.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Signature) (Ciy, Date)

[**1 Introduction**](#_7rpj8tt8dtge) **3**

[1.1 Motivation](#_qjc0q3di0onm) 4

[1.2 Objectives](#_png0adqqbp1z) 4

[**2 Prerequisites**](#_wwigdn3qc5wd) **4**

[2.1 Procedural](#_kyfjtte2q00z) 4

[2.2 Textures](#_fxbn9cia2i0b) 4

[2.3 Materials](#_ff7gpcgpd2th) 4

[2.4 Implicit and explicit algorithms](#_6w0yiy2j1usw) 5

[2.5 Pro & contra of procedural patterns within fragment shaders](#_v1sf2ph8d0qd) 5

[**3 Analysis of surfaces**](#_7jj8i57kh9os) **5**

[3.1 Extracting surface layers](#_71ytufqjn2f5) 6

[3.2 Visual properties of materials](#_f8klf2mmsde8) 7

[3.3 Environmental influences](#_ry1priibcb07) 7

[**4 Toolbox of algorithms**](#_ogxmx9wb1kgw) **8**

[4.1 Basic math](#_7i5dnylo56kv) 8

[4.2 UV](#_6ttkh0vcat0l) 8

[4.3 Noise](#_kv0ktvzgbpvk) 9

[4.3.1 Hashing as random number generator](#_7bcvj8ccrot8) 9

[4.3.2 Noise base functions](#_khgqryb1nuwt) 10

[4.4 Shapes](#_519zm0rfffy5) 10

[4.5 Easing](#_y4qf63qpy9ma) 11

[**5 References**](#_rln5g48ssf8a) **11**

[**6 Figures**](#_ddw7y5oef95i) **11**

# 1 Introduction

Procedural texturing always has been a subject in computer graphics. Researchers sought for algorithms and improvements to synthesize textures to represent natural looking surfaces. Early algorithms like Perlin-Noise [(P01)] or Worley-Noise [(W01)] are still present today and essential for procedural generations, due to their natural looking appearance which suits replicating natural surface properties.



*> [Figure01] Images from early papers; Left: Perlin noise [(P01)]; Right: Worley noise [(W01)]*

Today, "With the ever increasing levels of performance for programmable shading in GPU architectures, hardware accelerated procedural texturing in GLSL is now becoming quite useful[...]" [(G01)], because "[...] modern shader-capable GPUs are mature enough to render procedural patterns at fully interactive speeds [...]" [(G01)]. Which enables essential algorithms types for procedural pattern generation like noise or distance fields to be used as implicit implementation in shaders, because shaders will query for information about arbitrary points, set by the current pixel distribution.

Today, a variety of modern 3D applications like "Blender" [(BLE01)], "Unity 3D" [(UNI01)], "Unreal Engine" [(UNR01)] or "Cinema 4D" [(CIN01)] offering interfaces to attached renderers to handle the shading of objects in a modular manner, as proposed in the paper "shade trees" [(C01)]. Besides enabling modifications to the shading of the mesh and creative abstract lightning models like toon shading, it enables at the same time generating procedural surface property information. While the interfaces of real time renderer already make use of the GPU with fragment shaders, offline render applications ,which are ray tracing the scene instead of rasterizing, require also implicit algorithms to evaluate a surface color at arbitrary points by ray intersections. Therefore the functionality and behavior of these interfaces are oriented towards fragment shaders. The interfaces are also shipped with abstract implementations of various useful algorithms to hide the underlying complexity.

Artists like Simon Thommes already pushed the limits of these interfaces really far and they were able to create stunning materials, only using procedural methods without any dependency to external resources like textures.



*> [Figure02] Procedural Materials in Blender, created for "Nodevember"; Made by Simon Thommes 2019*

## 1.1 Motivation

Due to the possibility to use and layer multiple algorithms in shaders and interfaces of various render applications, creating procedural materials can be still a tedious and complex task. Manipulating results of algorithms in the right manner often relies on repetitive tasks and practical knowledge to get convincing results. Because the endless possibilities and creative freedom for manipulations and choice of algorithms in shaders and interfaces of render applications will not enforce or guide creators to a specific workflow to create procedural materials. And additionally, only implicit algorithms for pattern generation can be used, because of the shader architecture. This excludes the use of post processing algorithms like blur, normal map generation from height or ambient occlusion. Post processing algorithms rely on neighbor information which cannot be accessed without buffers in fragment shaders. Buffers are not available in every render application interface and if it does, the usage of them can vary for each render application interface.

## 1.2 Objectives

This thesis will serve multiple objectives. First, an understanding for real world surfaces and their composition should be created. Therefore they have to be analyzed how they can be decomposed in distinct information, layers, forms and patterns. Secondly, to know which algorithms are suited for procedural generation and which common use cases are occurring, a categorization based on their task and type needs to be created. Thirdly, guidelines have to be defined, in order to reduce Trial-And-Error phases and guiding creators to a structural process. Finally the analysis, categorization and the capabilities of the workflow are tested by creating a procedural texture and documenting each step.

The order of the named objectives will also represent the structure of the thesis. Details about implementation or specific algorithms are not part of this work. As well as a performance analysis of algorithms or entire procedural materials.

# 2 Prerequisites

By dealing with render applications, procedural texture generation and shaders, some terms can have similar meaning and sound. Therefore their meaning in this work has to be defined to prevent misunderstandings.

## 2.1 Procedural

The paper "A Survey of Procedural Noise Functions" defined “procedural” as:

"The adjective procedural is used in computer science to distinguish entities that are described by program code rather than by data structures. Procedural techniques are code segments or algorithms that specify some characteristic of a

computer-generated model or effect." [(LLC01)]

## 2.2 Textures

Textures are images where a single or more information about surface properties is stored. Combined with the definition for "procedural", a “procedural texture” is defined by Dr Sebastien Deguy as: "[...] a computer-generated image created using an algorithm [...], instead of a digital painting or image processing application[...]" [(D01)]. Render applications are then using these textures to feed the properties of an assigned lightning model.

## 2.3 Materials

Most render applications using the term "material" for the combination of the used lighting model and the collection of information for the lightning model, like albedo or specularity. This thesis uses the term "material" for the collection of information only, excluding the lighting model. Because the lightning model has only a minor influence on the process of replicating a surface in a procedural manner.



*> [Figure03] Different lighting models, same material information; Right: Physical based (PBR); Left: Non-photorealistic (NPR)*

As seen in the figure, with different lighting models the visual appearance of surfaces can change drastically, even if the offered information about the surface properties are the same. The lightning model can influence the process of procedural materials by requiring special information. Nonetheless this will not change the general approach of how to analyze and replicate surfaces, as well as the underlying techniques and algorithms.

## 2.4 Implicit and explicit algorithms

Implicit algorithms are more suited for procedural materials than explicit ones. The difference between those two is that an implicit algorithm will answer a query about an arbitrary point and returns information exclusively for it. While an explicit algorithm returns the whole result, evaluated for a resolution defined by the renderer and not by the shader itself. [(EMP01)] Due to the architecture of shaders, regardless of weather used by ray tracing or rasterization, the task of shaders is to evaluate arbitrary points of a surface. These points are defined either in rasterization by the current resolution and view, in case of ray tracing are the points determined by the hit location of emitted rays. Implicit algorithms which can return results to these points without dependencies to neighbor point information and resolution are therefore preferred or even necessary.

## 2.5 Pro & contra of procedural patterns within fragment shaders

Related work already has explored and analyzed several advantages and disadvantages of pattern creation in fragment shaders, besides the named problems for the motivation.. While they are not part of this thesis they still have to be pointed out. The book “Texturing and Modeling - A Procedural Approach” [(EMP01)] already made a good listening of these pros & cons which can be represented shortened as:

* The size of a procedural representation is way smaller than saving their result as texture.
* The evaluation can be executed at any resolution.
* Procedural materials can be parameterized to change the appearance and features.

Disadvantages are:

* The development process can be difficult, because of the complexity of algorithms and the lack of debug possibilities.
* Results of chaining algorithms are hard to plan without practical knowledge.
* Evaluating can be slower than accessing textures.
* Aliasing can be a problem, especially for far zoomed out textures.

# 3 Analysis of surfaces

The overall objective of the analysis should not confound with building a physical and chemical understanding of real world materials, which nonetheless may be helpful or necessary. The main objective is to extract patterns and geometric information about the visual appearance. To retrieve these information, the analysis of surfaces is carried out in three steps:

- Extracting surface layers

- Visual properties of materials

- Environmental influences

The steps were derived on one hand by looking through guides of the specialized software for procedural texture creation “Substance Designer” [(SD03)], and on the other hand by transferring the knowledge to a pure shader based environment. While creating several textures, these steps have been evolved over the time by analyzing several types of surfaces with the goal in mind to get the information as structured as possible. To support each step in their explanation, their concept is applied to an example reference photo of a wooden floor in a Pub.

The proposed extraction steps rely on pattern, noise and shape recognition. The retrieved information about the surface composition and visual features is reused later by replicating them with matching algorithms, order and techniques that are available to fragment shaders. To utilize the extracted information, they can either be implemented directly while analyzing the surface in parallel or persisted in any preferred way. The implementation does rely on the content of the information, not on its persistence.

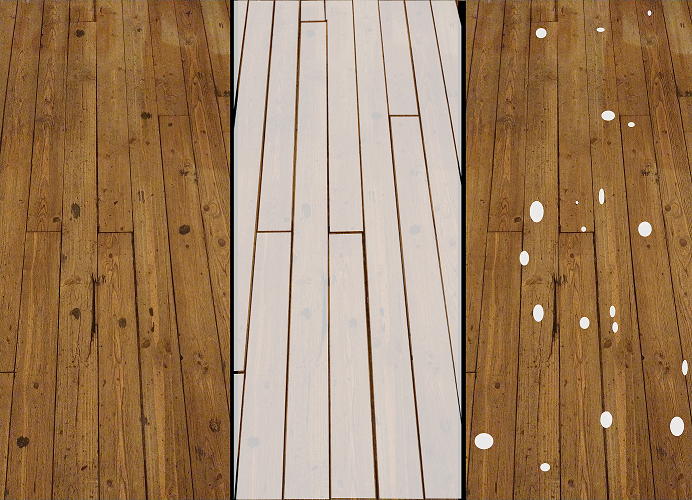
## 3.1 Extracting surface layers

Separating surfaces into different layers of sub-materials helps later to reassemble the surface in the material in the same manner as image editing software does this through blending them into an final image. In addition the separation serves as simplification to the reference surface to overcome possible complex compositions, which then can be understood more easily detached from other influences.

A hierarchical approach for looking out for layers is recommended, because the depth of the analysis will differ by the required level of detail for the material. Further the hierarchical approach can also mimic the creation process, physical and chemical processes over the lifetime. These influences create natural layers of materials on a surface, like leaves on the ground, oxidations or screws in furniture, which almost matches the chronological appearance and history of the surface. Another advantage of a hierarchical approach is the option that replicated materials can be reused in other materials, because the layering should only control their distribution in the final material.

Factors which are decisive for separations are:

* Physical factors; Many surfaces are already separated naturally through manufacturing processes, like stone walls and floors, where the final surface is man made. It also includes occurrences where different materials are placed on top or worn away without initial intent, like posters on a wall, leaves on grass or peeled plaster
* Chemical Factors; surfaces will change their appearance over time. Oxidation is a typical everywhere occurring process for metals, where the metal slowly converts to rust.
* Abstract patterns; Surfaces may also have noticeable features which are expressed in height, color or other properties without dependency to a distinctive material. Patterns on wallpapers or height structures on surfaces to provide more grip are an example of that.



*> [Figure04] Left: Floor of a Pub; Mid: Separation by planks; Right: Separation by trampled gums*

The first separation is made by the wooden planks, because:

* The planks are physically separated and are independent to each other.
* they are the most perceptible feature of the floor.
* they control the base height of the surface.
* the arrangement and shape results in an almost regular pattern.

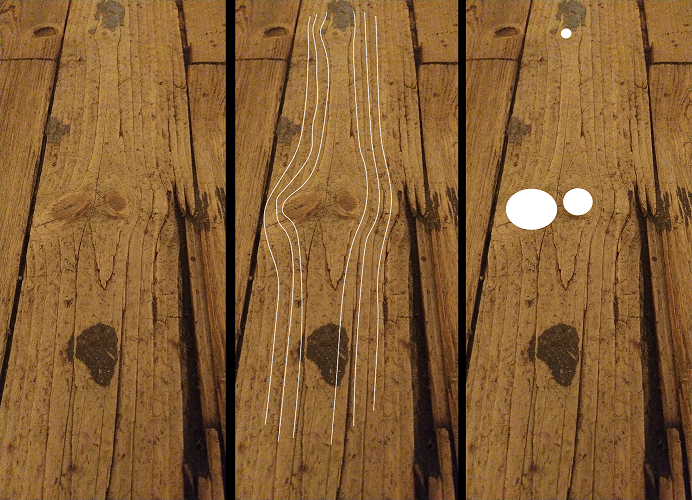
While the first separation is oriented to the wood structure, the black dots on the floor appear to be independent to the plank structure. This is because these dots are old trampled chewing gums. Therefore a second separation is made since:

* they do not show a dependency to the plank or wood structure and overlap some planks.
* they are a different material than wood.
* they are physically placed on top of the floor.

## 3.2 Visual properties of materials

After separating a surface in multiple layers, visual properties about the isolated materials have to be extracted for recreation. Again a hierarchical approach is recommended, this time driven by the obtrusiveness of visual features, which ensures that the material is later immediately recognizable. The hierarchical approach matches the later proposed workflow for implementation by iterating from rough to small details. This takes only features into account which are necessary and required by level of detail for the material.

While the factors for the separation often are quite obvious, the extraction of features from materials is still oriented to the named factors earlier.



*> [Figure05] Left: Single plank of the floor; Mid: Annual rings, Right: Branches*

The floor planks are made of wood. The two most recognizable features of wood are annual rings and branches, which are also represented in the reference photo. There are more features in the reference photo which can be extracted, but the two initial features, annual rings and branches, are enough for a first recognizable implementation of a wood material.

## 3.3 Environmental influences

While a surface can be separated into layers, and the materials which made up the layers are treated separately, a surface is still part as a single instance of the environment. This means the environment where the surface exists has a strong influence on the appearance. Trampled chewing gums on the floor are a result of an environmental influence, because the floor is located in a public place, a wooden floor in a living room may not have the feature of trampled gums.

Visual features which may appear at first glance inexplicable often can be explained by looking though the history of a surface. Environmental influences are the factors which make materials finally believable. Thinking about the environment and gathering background information about the history, conditions or story can leave visual impressions on surfaces. This knowledge can be applied factionary to surfaces to integrate them in the resulting material to fit more convincingly in the final environment.



*> [Figure06] Left: Floor in a Pub; Mid: burned spots from trampled cigarettes; Right: color variation due to spilled liquids*

The wooden floor of the example is no exclusion to that. The floor is located in a smoking area and in the reference photo are all over the place small dark points like "freckles" on the floor. With a close inspection these freckles are identified burned spots from trampled cigarettes. Another information to consider in a pub is the possibility of spilled liquids. These liquids may not be wiped away immediately, so the liquid will be soaked up from the floor, which can cause discolorations to the wood.

# 4 Toolbox of algorithms

Various algorithms are required to replicate the extracted information from the analysis in the form of shapes, noises and patterns. Through exploring different purposes of algorithms that are suited to be implemented in fragment shaders, a library of algorithms dedicated for procedural material generation has been formed. This library serves as the origin for the derivation of the proposed categorization and as a toolbox for the workflow to avoid reimplementation.

Adding algorithms to the library resulted in an evolving categorization by their purpose. This categorization has been settled and shown to cover all tasks of procedural material creation with fragment shaders. The categorization was made to solve different purposes:

* To keep the library organized and make collected algorithms easily findable by the required task, because their name does not always conclude to their purpose. The naming of basic mathematical methods like “absolute” or “sinus” might be clear, complex algorithms, including self-made ones, may have arbitrary names like “valleys” or “voronoi”.
* By implementing algorithms abstract and without specialization for use cases, they enable a modular use of them, which can be compared to Lego bricks. Despite the abstraction, they still lead to distinctive and individual results through chaining and layering. Further the categorization and abstract implementation should support the exploration of algorithm combinations for new results, because of their exchangeability and compatibility.
* Each interface of render applications will ship with different subsets of already implemented algorithms. And in case of pure GLSL, only basic mathematical methods are given. A categorization allows changes for available collections to complete them by adding missing required algorithms. It also enables the classification of altered or specific algorithms of own choice, which may be explored while creating new materials.

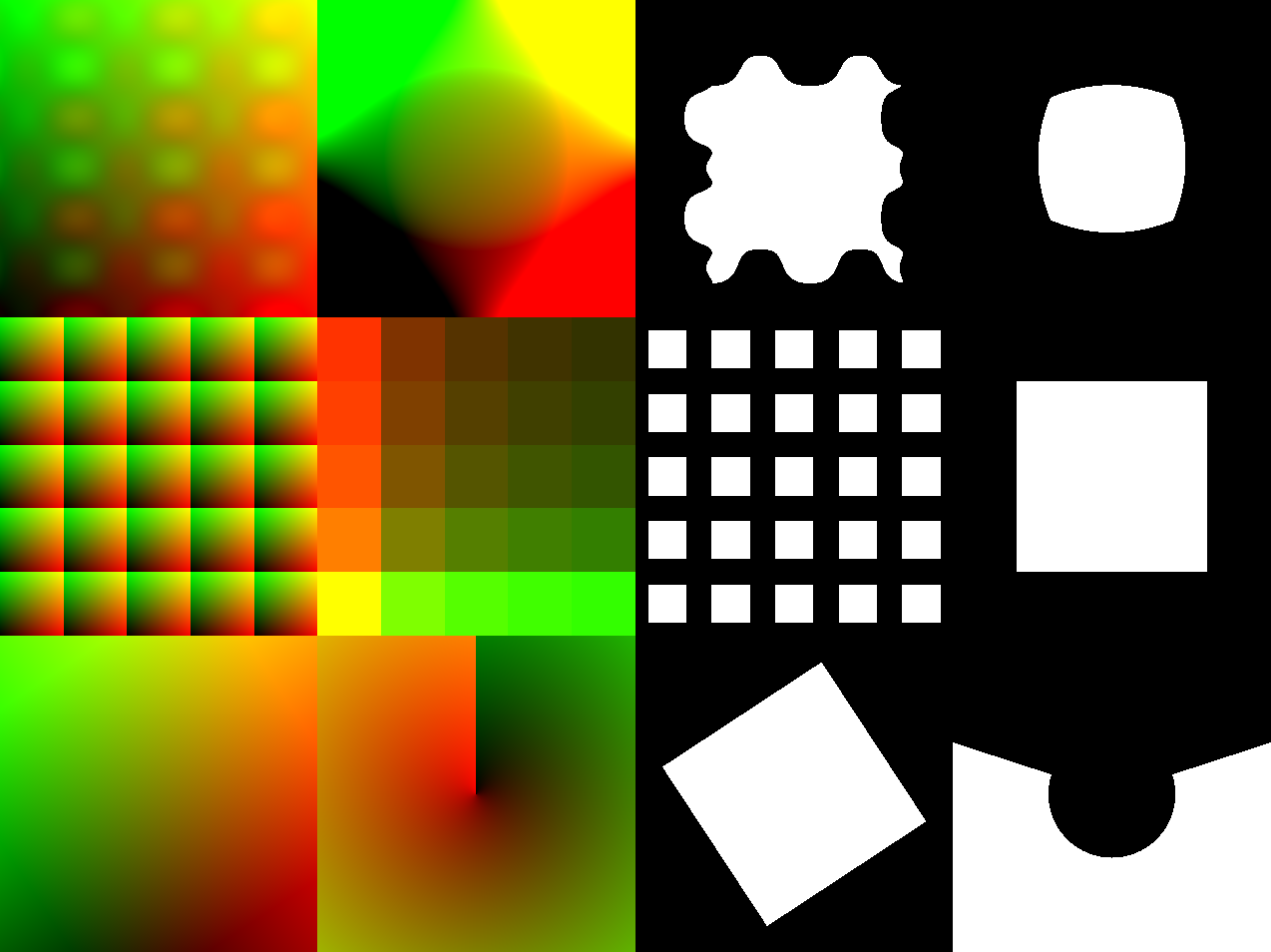
## 4.1 Basic math

Basic mathematical methods like "floor", "absolute" or "sinus" are essential for procedural generation and forming the base of every algorithm. This includes mathematical operations as vector math, trigonometry and matrices. Another use case of basic operations is the processing in results of previous algorithms, where results are either used to be combined or as origin to create new information.

## 4.2 UV

As mentioned earlier in 2.4 (Implicit and explicit algorithms), through ray tracing or rasterization are positions on surfaces evaluated where the underlying material information is required. The evaluated position, which usually is two or three dimensional, results through a conversion in texture coordinates that are commonly named “UV”. The UV represents the required location of the material information as two dimensional coordinates. These coordinates are usually used to access a texture with persisted material information for the lighting model. By generating procedural material information, the UV serves as a pointer of the location for the implicit generative algorithms and therefore the resulting information, which makes them just as important as basic math functions.

Because the UV defines the location of generated information, manipulating them concludes to a change in the generated information distribution, which take effect in the visual appearance of subsequent algorithms. While implementing transformations directly into the generative algorithms like scaling, translation or rotation can be quite complicated but applying changes to the UV results in two major advantages. First, many algorithms require the UV coordinates anyway, because of that manipulations done in forehand and outside of the algorithm can be shared and reused by multiple algorithms. Second, abstracting UV manipulations enables the transfer of them to any algorithm that utilizes UV coordinates. This leads to endless possible combinations which otherwise could not be achieved.



*> [Figure07] Left: UV manipulations; Right: rectangle with same size and position drawn with UV's from left*

Besides basic transformations as scaling, translating and rotating are more complex manipulations possible. These may be often required or enable unique results. Tilling is one common manipulation, where a given UV is fractured in regular sub-UV’s. This is a common approach to mimic repetitive patterns. Figure07 shows the UV and their result in the first column and middle row.

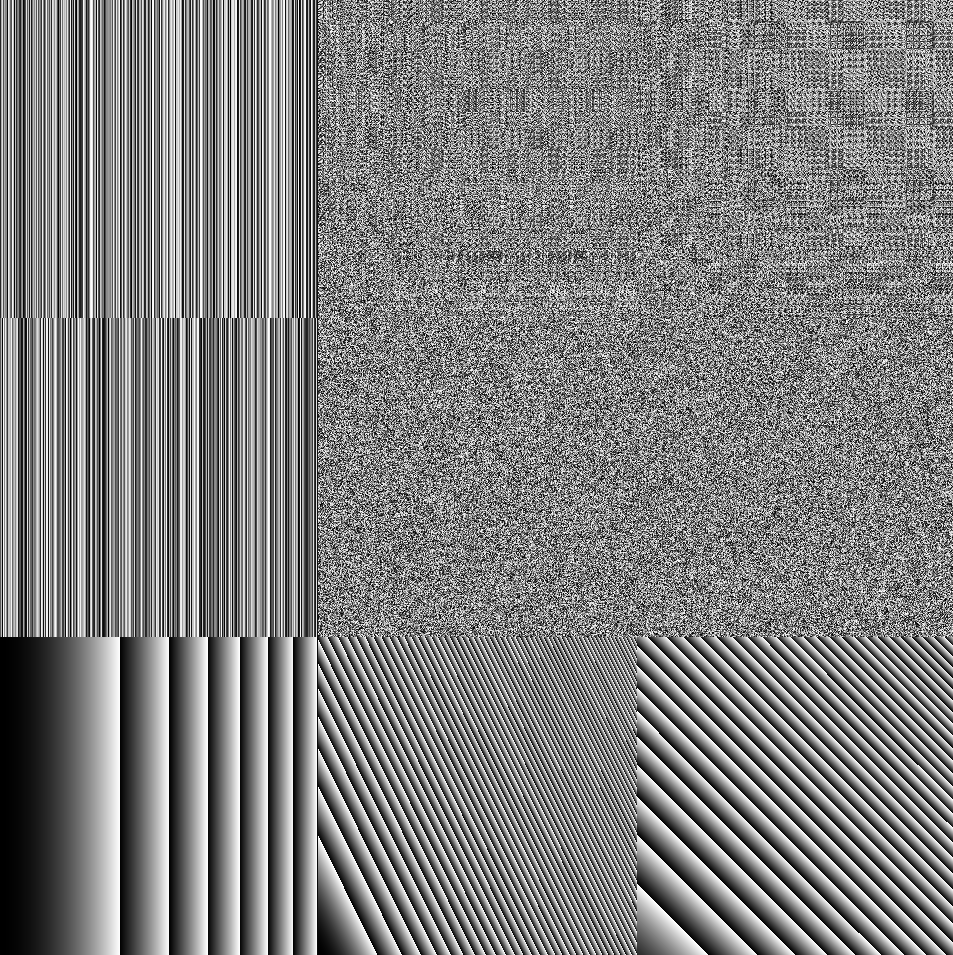
Another manipulation is the conversion into another coordinate system. The UV usually represents the cartesian coordinate system. There are other systems and projections where cartesian coordinates can be transformed to. An useful transformation could be to the polar system, where the position is described as angle and length. Figure07 shows this in the second column and bottom row.

## 4.3 Noise

Noise algorithms are the element which mimic unpredictable patterns or random distributions on surfaces. There are two kinds of algorithms for this category. The first are algorithms that will work as pseudo random number generators (RNG), the second are noise algorithms which rely on the RNG's to create unpredictable still repetitive gradients.

### 4.3.1 Hashing as random number generator

The base of all noise algorithms and random distributions is the access to a random number generator (RNG). While true randomness is hard to achieve with computers, it is even undesirable for creating procedural noise. A RNG for procedural materials has to be unpredictable and reproducible at the same time. Often algorithms need to restore random values e.g. accessing the value of neighbor points in a lattice.

*> [Figure08] White noise; Left to right: 1D, 2D, 3D; Bottom to top: Scaled by x0.0001, x1.0, x1000.0*

Hashing is the perfect solution to be used as RNG, because the results can be unpredictable but still controllable through input. A good result of such a hash function is named "white noise", which contains all frequencies at once. But not any function is suited to be used as a RNG. The hash algorithms should be consistent over the used UV scale in the procedural material. As shown in Figure08, the randomness of hash algorithms can break in extreme scales. There is a good listing in the book "Texture & Modeling A procedural approach" of other properties that a hash algorithms have to fulfill in order to be used as RNG:

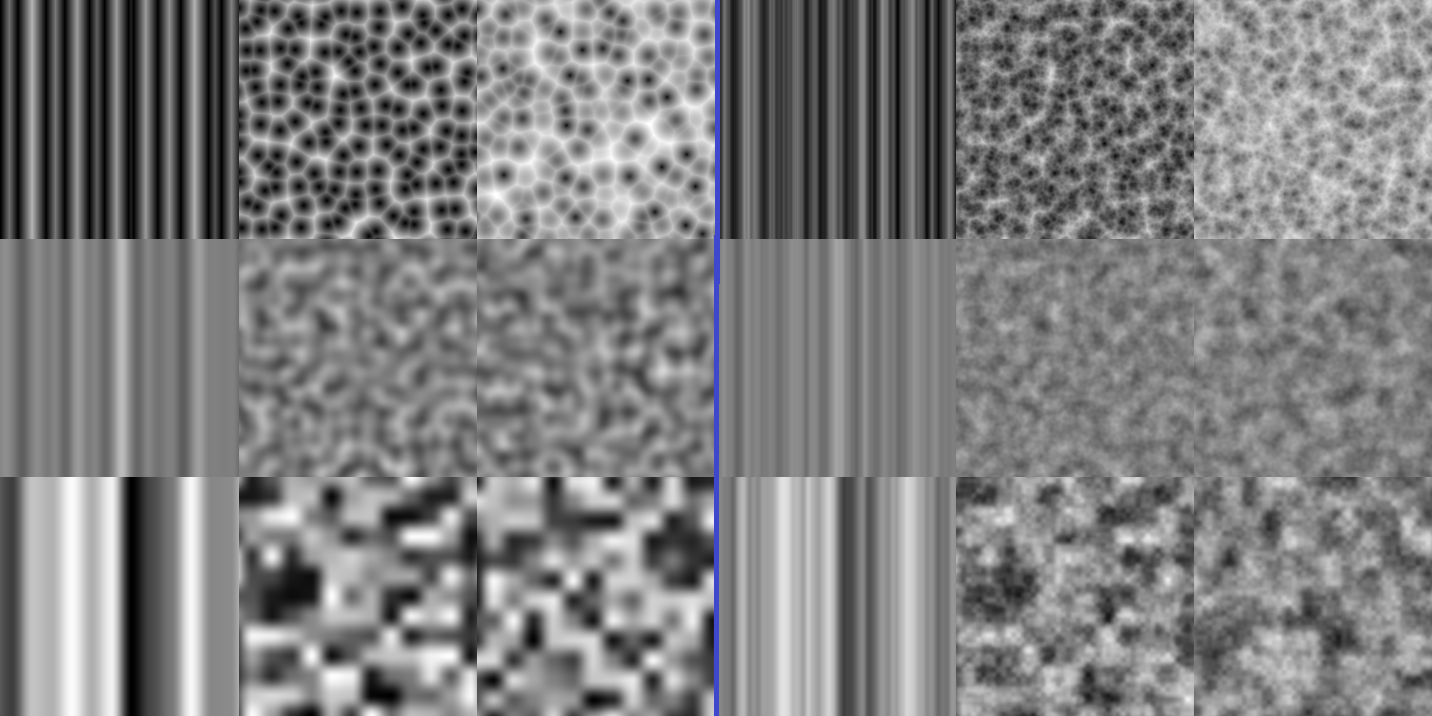
"The properties of an ideal noise function are as follows:

* noise is a repeatable pseudorandom function of its inputs.
* noise has a known range, namely, from −1 to 1.
* noise is band-limited, with a maximum frequency of about 1.
* noise doesn’t exhibit obvious periodicities or regular patterns. [...]
* noise is stationary—that is, its statistical character should be translationally invariant.
* noise is isotropic—that is, its statistical character should be rotationally invariant."

(the term “noise” from the quote is referred as white noise)[(EMP01)]

### 4.3.2 Noise base functions

Base functions of noise are considered as functions which create repetitive unpredictable gradients by interpolating random numbers. The paper "A Survey of Procedural Noise Functions" [(LLC01)] gives a good insight about the noises themselves and a categorization of their types. The most known algorithms are perlin noise[(P01)] and voronoi noise (also known as cellular noise)[(W01)].

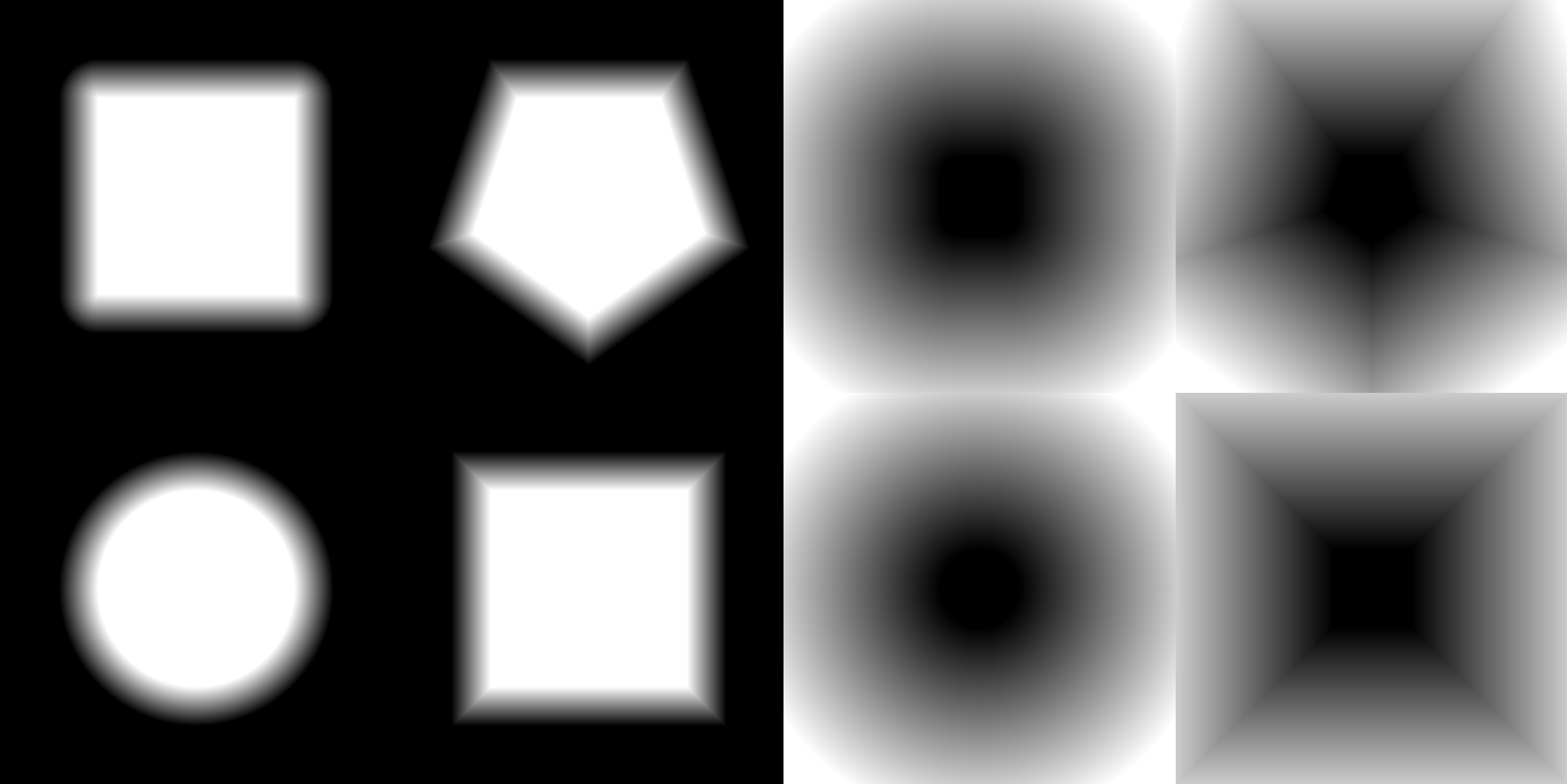


*> [Figure09] Several noises; Left to right: 1D, 2D, 3D; Bottom to top: value, perlin, voronoi; Left: pure noise; Right: fractal Brownian motion based on left sided algorithms*

As shown in [Figure09] the visual appearance of noise can be very different. This is useful in terms of mimicking different natural patterns, where some algorithms are better suited than others. Another consideration of collecting algorithms besides the visual appearance is their application in different dimensions than 2D. One dimensional noises are quite useful when it comes to created color gradients. Their result could either be used to mix defined colors, or their results could control a single color channel. A good example of controlling color channels was made by Inigo Quilez where he used a single offsetted and scaled cosines function for each channel [IQ04]. On the other hand three dimensional noises are useful when it comes to accurately mimicking a cross section, the third dimension can be replaced for that with a consistent number.

## 4.4 Shapes

While noise algorithms are utilized to mimic unpredictable surface structures, shape algorithms are the complementary solution for geometric structures. These structures will appear likely on man-made surfaces like rooftops or fabrics. But also natural surfaces can show derivatives of geometric forms like leaves or pebbles in mud. To mimic geometric features, algorithms for basic shapes are needed. These algorithms should produce, like noise algorithms, a single value within the range of 0 and 1. Not every complex geometric shape has to be implemented as an algorithm. They can be often reassembled by combining basic shapes with boolean operations as subtraction, intersection and union.

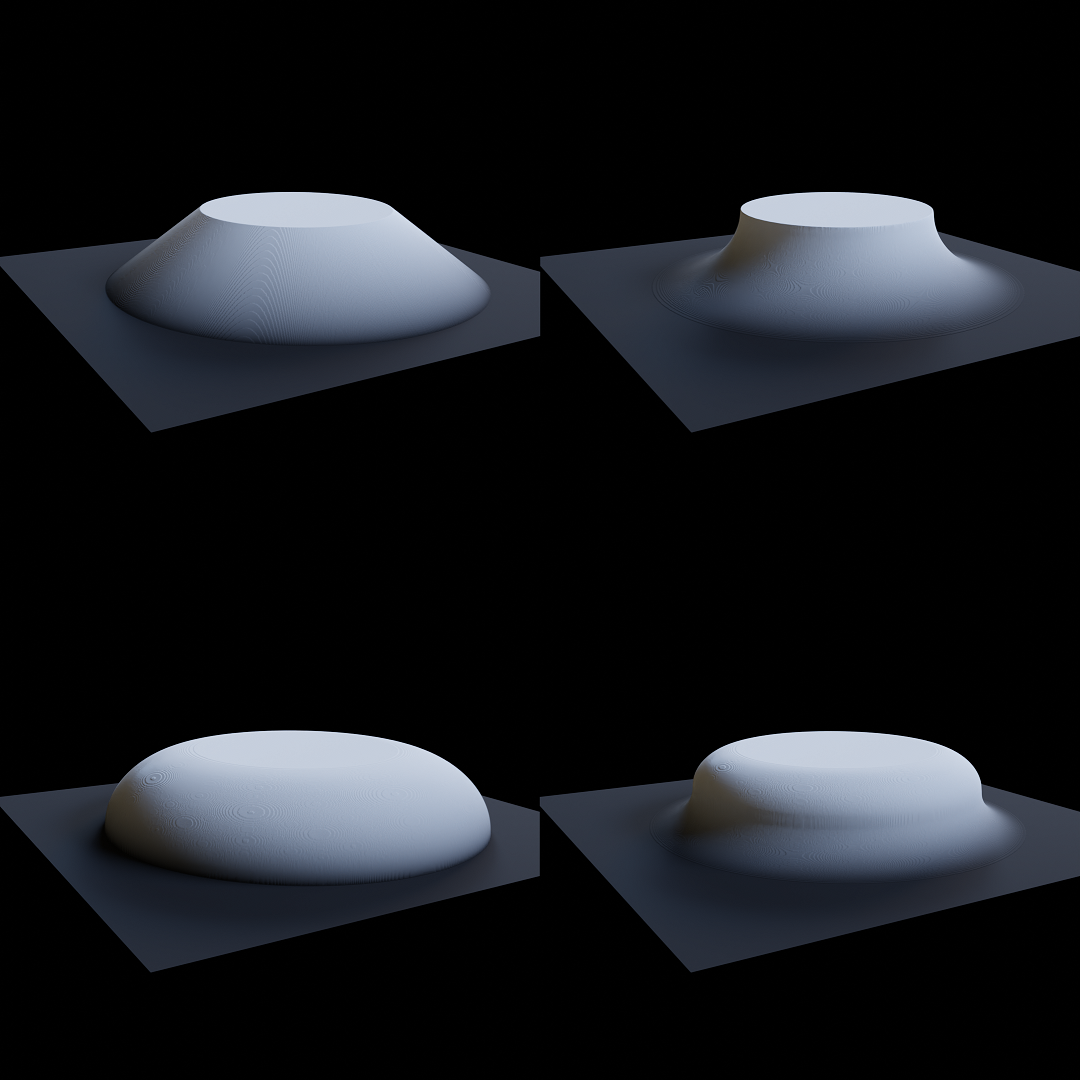


*> [Figure10] Left: various shapes with blur; Right: Distance fields*

An important feature which every shape algorithm has to provide is blur, as shown in [Figure10]. Unfortunately this has to be done within the shape generation, because after the generation there is no way to blur a shape or information without access to neighbor information, which is not given in shader. Blurring shapes are essential for creating procedural materials, because geometric shapes, like noise, have transitions that are represented in various features of a surface, where a transition into another height or color is needed. Without blur there would be an instantaneous change in information which does not reflect most surfaces. To provide blurring basic shapes, using a distance field as base is recommended, because then only the start and end point of the transition has to be given. Inigo Quilez made a nice listening of several distance functions in 2D and 3D space on his website [(IQ02)]. Another detail for blurring shapes which should be noticed is that the blur should be linear. A linear gradient will make modifications to the transition distribution easier and exchangeable.

## 4.5 Easing

Manipulating values is a common element for creating procedural materials. Usually the result of shape and noise algorithms are in the range between 0 and 1. To utilize these results, often their range or distribution has to be adjusted to achieve desired results. This can be either done by scaling and offsetting the result for simple adjustments or the result can be manipulated through easing functions where the results can be very distinguishable from the original input.



*> [Figure11] Left: Left to right, bottom to top: Exponential, Power, Sinus, Circular; Right: blurred circle as displacement and color with applied easing*

Easing is a well known technique for animations to improve linear interpolation between keyframes to emphasize them. Many algorithms, as mentioned before, will also have gradients within a fixed range. These functions are perfectly suited to recreate information like color, height (shown in [Figure11]), basically wherever transitions are occurring.

# 5 References

# 6 Figures