Procedural Solid Texturing

Project Report

02562 Rendering - Introduction
December 2025

Anton Thestrup Jorgensen
s194268

DANMARKS TEKNISKE UNIVERSITET
DTU COMPUTE

Contents

1 Introduction

Procedural textures generate surface appearance analytically, e.g. as a function of position, rather than relying on stored image data. When applicable, this approach offers several advantages, including the absence of texture memory usage, resolution independence, and consistent appearance under magnification. Because procedural textures are defined in continuous space, they integrate naturally with physically based rendering and path tracing pipelines.

Procedural textures are particularly well suited for materials with repetitive or stochastic structure, such as wood, marble, stone, clouds, smoke, or terrain, where the visual pattern follows geometric rules rather than specific imagery. They are also advantageous when materials must scale to large scenes, support extreme zoom levels, or remain stable under deformation and animation.

However, procedural textures are not universally appropriate. Materials that require precise, artist-authored detail (such as signage, text, logos, decals, or unique surface markings) are typically better represented using image-based textures. In these cases, direct control over exact pixel content outweighs the benefits of analytic generation. As a result, procedural and image-based textures are best viewed as complementary tools, each suited to different classes of materials and visual requirements.

2 Method

This project implements two procedural materials: a wood material and a marble material. Both are defined analytically in 3D space and evaluated at shading time, avoiding texture lookups. The materials are based on geometric projections, trigonometric modulation, and multi-octave noise.

Procedural Wood

The wood material models a cylindrical tree trunk with growth rings and longitudinal grain.

Log* Coordinate System (*the tree part, not the function) A central axis is defined by a point $\mathbf{p}_0$ and a unit direction $\mathbf{d}$. For a surface position $\mathbf{p}$, the coordinate along the trunk is

$$t = (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{d}.$$

The closest point on the axis is $\mathbf{q} = \mathbf{p}_0 + t\mathbf{d}$, and the radial vector in the plane orthogonal to the trunk is

$$\mathbf{r} = \mathbf{p} - \mathbf{q}, \qquad r = \|\mathbf{r}\|.$$

An orthonormal basis $(\mathbf{u}, \mathbf{v})$ spanning the plane orthogonal to $\mathbf{d}$ is constructed, allowing an angular coordinate around the trunk:

$$\theta = \operatorname{atan2}(\mathbf{r} \cdot \mathbf{v},\ \mathbf{r} \cdot \mathbf{u}).$$

Growth Rings Growth rings are modeled as a periodic function of the radial distance. To reproduce wider rings near the center and tighter rings near the bark, a nonlinear phase function is used:

$$\phi(r) = f\left(r + \alpha r^2\right),$$

where f is a base ring frequency and α controls the tapering of ring width. The raw ring signal is then

$$R(r) = \tfrac{1}{2}\left[1 + \sin(\phi(r))\right].$$

A smooth thresholding operation is applied to sharpen the contrast between earlywood and latewood regions.

Domain Warping To avoid perfectly concentric rings, the radial coordinate is perturbed using low-frequency noise:

$$\tilde{r} = r + \beta_1 N_1(\mathbf{p}) + \beta_2 N_2(\mathbf{p}),$$

where N_1 and N_2 are noise functions at different scales and β_i control the warp amplitude. This technique, known as domain warping, produces natural irregularities in the ring structure.

Color Composition Two base colors are defined for earlywood and latewood. The final diffuse color is computed by mixing these colors according to the ring signal and modulating the result by the grain term and an additional low-frequency tint variation.

Procedural Marble

The marble material is based on sinusoidal bands distorted by turbulent noise.

Vein Direction A unit vector $\mathbf{d}$ defines the dominant vein direction. For a point $\mathbf{p}$, a coordinate along this direction is

$$x = (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{d}.$$

Turbulence and Bands A multi-octave noise function $T(\mathbf{p})$ is evaluated and remapped to $[-1, 1]$ to produce turbulence. The marble pattern is generated by modulating a sinusoid with this turbulence:

$$M(\mathbf{p}) = \tfrac{1}{2}\left[1 + \sin\left(\omega x + \gamma T(\mathbf{p})\right)\right],$$

where ω controls band spacing and γ controls vein distortion. A smooth threshold is applied to emphasize veins.

Color Mapping Two colors are defined for the base stone and the veins. The final diffuse color is obtained by interpolating between them using the processed band signal.

Noise Functions

Both materials rely on procedural noise for natural variation.

Value Noise A scalar value noise function is defined on a 3D grid. For a point $\mathbf{p}$, the surrounding lattice points are assigned pseudo-random values via a hash function. Trilinear interpolation with a smoothstep kernel produces a continuous noise field:

$$N(\mathbf{p}) = \mathrm{lerp}_{xyz}\left(h(\lfloor \mathbf{p} \rfloor + \mathbf{i})\right),$$

where h is a hash function and $\mathbf{i} \in \{0, 1\}^3$.

Fractal Brownian Motion To obtain richer structure, multiple octaves of noise are summed:

$$\mathrm{fBm}(\mathbf{p}) = \sum_{k=0}^{K-1} a_k \, N(2^k \mathbf{p}),$$

with amplitudes a_k decreasing geometrically. This produces scale-invariant, natural-looking variation used for both domain warping and fine detail.

3 Implementation

This section explains the concrete WGSL implementation of the procedural noise functions and the two materials. Code excerpts are included using `listings` and referenced explicitly in the text.

Noise Functions

All procedural variation is built on a compact value-noise implementation and its multi-octave extension.

Hash Function

Listing 1 shows a simple hash that maps a 3D point to a pseudo-random scalar in $[0, 1)$. It uses a dot product with large constants followed by a sine and fractional extraction.

Listing 1: Hash function

```
fn hash3(p: vec3f) -> f32 {
    let h = dot(p, vec3f(127.1, 311.7, 74.7));
    return fract(sin(h) * 43758.5453);
}
```

This function is inexpensive and sufficient for decorrelating lattice points in procedural textures.

Value Noise

Value noise is implemented by hashing the eight corners of the unit cube surrounding the query point and trilinearly interpolating between them. The implementation is shown in Listing 2.

Listing 2: 3D value noise

```
fn value_noise(p: vec3f) -> f32 {
    let i = floor(p);
    let f = fract(p);
    let u = f * f * (3.0 - 2.0 * f);
    ...
    return mix(nxy0, nxy1, u.z);
}
```

The cubic polynomial used to compute `u` is a smoothstep function, ensuring continuous first derivatives across cell boundaries.

Fractal Brownian Motion

Fractal Brownian motion (fBm) is implemented as a fixed sum of five octaves of value noise (Listing 3). Frequency doubles and amplitude halves each octave.

Listing 3: Fractal Brownian motion

```
fn fbm(p: vec3f) -> f32 {
    var sum = 0.0;
    var amp = 0.5;
    var freq = 1.0;
    for (var i = 0; i < 5; i++) {
        sum += amp * value_noise(p * freq);
        freq *= 2.0;
        amp *= 0.5;
    }
    return sum;
}
```

This produces smooth, scale-invariant noise used for both domain warping and fine detail.

Procedural Wood Material

The wood material implementation is shown in Listing 4. It models a tree trunk using an axial coordinate system, growth rings, and longitudinal grain.

Listing 4: Procedural wood material

```
fn progressive_material_wood(pos: vec3f) -> Material {
    let center_pos = vec3f(25.0, 0.0, 25.0);
    let d = normalize(vec3f(0.2, 1, -0.1));
    ...
    return Material(vec3f(0.0), color);
}
```

Coordinate System The trunk axis is defined by `center_pos` and direction `d`. An orthonormal basis $(\mathbf{u}, \mathbf{v}, \mathbf{d})$ is constructed using cross products. A conditional choice of the reference "up" vector avoids numerical instability when `d` is nearly vertical.

Each shading point is decomposed into:

$$t = (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{d}, \qquad r = \|\mathbf{p} - (\mathbf{p}_0 + t\mathbf{d})\|.$$

Domain Warping The radial distance r is perturbed using two fBm evaluations at different frequencies. This step (lines computing `turb`, `turb2`, and `warp_r`) introduces irregular, natural-looking ring deformation.

Growth Rings Ring spacing is controlled by a nonlinear phase:

$$\phi(r) = f\,(r + \alpha r^2),$$

implemented via `ring_freq` and `taper`. This produces wide inner rings and progressively tighter outer rings. The sinusoidal signal is sharpened using `smoothstep` to emphasize contrast between earlywood and latewood.

Longitudinal Grain The angular coordinate

$$\theta = \text{atan2}(\mathbf{r} \cdot \mathbf{v}, \mathbf{r} \cdot \mathbf{u})$$

is combined with the axial coordinate t and evaluated through fBm. This creates streaks aligned with the trunk. The resulting grain signal modulates the base color multiplicatively.

Color Composition Two fixed colors represent dark and light wood. Ring structure selects between them, while grain and a low-frequency tint variation further modulate the result. The material has no emission and is purely diffuse.

Procedural Marble Material

The marble material is shown in Listing 5.

Listing 5: Procedural marble material

```
fn progressive_material_marble(pos: vec3f) -> Material {
    let center_pos = vec3f(0.0, 0.0, 0.0);
    let d = normalize(vec3f(0.2, 0.6, 0.77));
    ...
    return Material(vec3f(0.0), color);
}
```

Directional Parameterization A dominant vein direction `d` defines a scalar coordinate

$$x = (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{d}.$$

Turbulence and Veins An fBm-based turbulence term perturbs the phase of a sinusoidal function of x:

$$M(\mathbf{p}) = \tfrac{1}{2}\left[1 + \sin(\omega x + \gamma T(\mathbf{p}))\right].$$

This produces characteristic marble veins. A smoothstep operation sharpens the result.

Color Mapping Two colors define the base stone and the veins. The final diffuse color is obtained by interpolating between them using the processed band signal.

Overall, the implementation closely mirrors the mathematical formulation while remaining compact and efficient in WGSL, requiring no texture memory and minimal branching.

4 Results

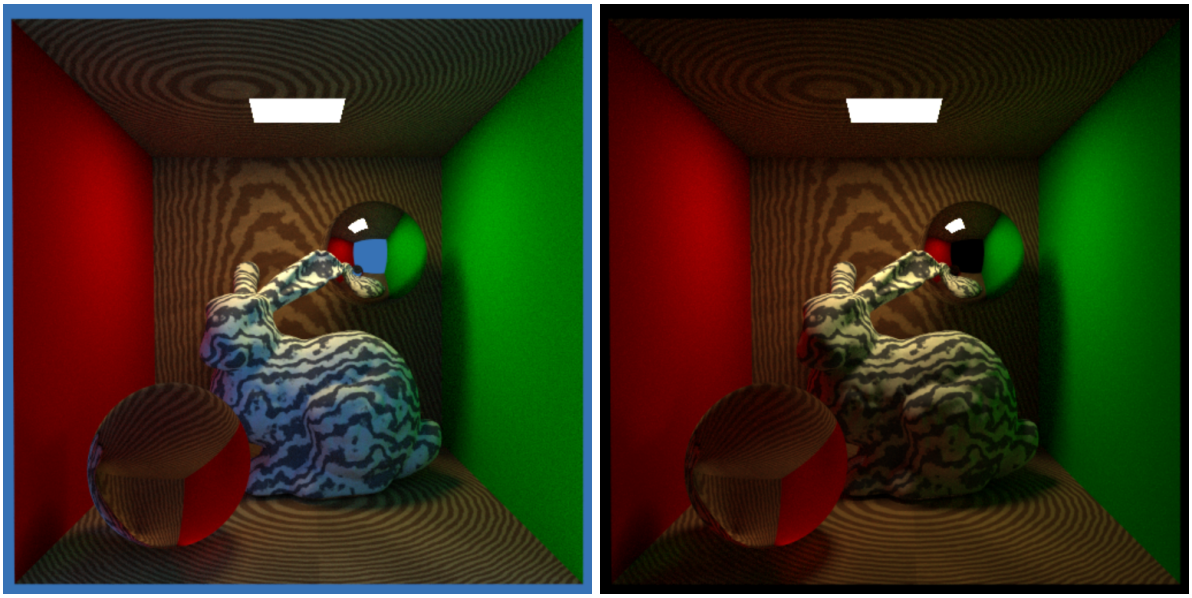

Figure 1: Final render of the scene with procedural wood and marble materials using different lighting conditions.