Kimmo Riihiaho

Procedural weathering

Master's Thesis in Information Technology

December 27, 2019

University of Jyväskylä

Faculty of Information Technology

Author: Kimmo Riihiaho

Contact information: kimmo.a.riihiaho@jyu.fi

Supervisors: Tuomo Rossi, and Jarno Kansanaho

Title: Procedural weathering

Työn nimi: Proseduraalinen ikääntyminen

Project: Master's Thesis

Study line: Applied Mathematics and Computational Sciences

Page count: 61+4

Abstract: Representing weathering of surfaces is an important aspect when creating believable virtual worlds. Existing weathering techniques are often either simulations concentrating to a single weathering phenomenon at a time, or pure visual models that do not have a connection to the birth mechanisms of the phenomenon. Our model fits in between these two techniques providing a visual weathering model that approximates real world birth mechanisms with a small set of parameters, which can be used for several weathering phenomena. Evaluation of our model is fast enough to be used as an interactive tool. Our model can be implemented on commonly available 3D-modeling software without programming skills required, which makes it suitable for audiences larger than the science community.

Keywords: weathering, aging, rusting, biological growth, moss, surface roughness, color changes, realistic rendering, texturing, procedural, noise

Suomenkielinen tiivistelmä: Pintojen ikääntymisen kuvaaminen on tärkeä osa virtuaalisten maailmojen uskottavuutta. Olemassa olevat ikääntymistekniikat ovat usein joko yksittäiseen ikääntymisilmiöön keskittyviä simulaatioita tai puhtaasti visuaalisia malleja, jotka eivät pysty mallintamaan ilmiöiden syntymekanismeja. Meidän mallimme asettuu näiden tekniikoiden väliin. Se on visuaalinen malli, jolla voidaan approksimoida useiden ikääntymisilmiöiden syntymekanismeja muutamilla parametreilla. Mallimme on riittävän nopea käytettäväksi interaktiivisena työkaluna. Malli voidaan toteuttaa yleisesti saatavilla olevilla

3D-mallinnusohjelmilla ilman ohjelmointitaitoja, joten se soveltuu tiedeyhteisöä laajemman yleisön käytettäväksi.

Avainsanat: kuluminen, ikääntyminen, ruostuminen, biologinen kasvusto, sammal, pintakarkeus, värimuutokset, realistinen renderöinti, teksturointi, proseduraalinen, kohina

List of Figures

List of Tables

Contents

1 Introduction

In physics, the direction of time can be stated in terms of increasing entropy, that is, the decrease in structure. An object made of iron will rust in time, but a rusted object will never spontaneously transform back into pure iron. In our everyday life, we can usually tell if an object is old or new just by looking at it. An old object would have defects caused by events in history, such as scratches and dents, color changes, and biological growth. In order to produce realistic looking synthetic images in computer graphics, it is important to be able to replicate these defects.

There are two commonly used terms to describe the phenomena that cause the defects: weathering and aging. The terms are often used interchangeably, though weathering is sometimes used to specifically cover phenomena occurring outdoors. In this study, we will not differentiate between the two and use the term weathering to describe weathering and aging phenomena in general. Weathering *phenomenon* is something that happens in the real world. We use the term weathering *effect* when referring to its synthetic counterpart in a virtual world, i.e., the visual effect which we create using computer graphics.

Weathering phenomena are commonly divided into mechanical, chemical, and biological weathering. *Mechanical*, also called physical, weathering refers to changes in physical structure of an object, or changes induced by physical objects, such as dents and scratches caused by impacts. *Chemical* weathering means changes in object's molecular structure, which are often caused by exposure to light, moisture, and chemical reactants, such as air pollution. *Biological* weathering refers to growth of living organisms, such as moss or mold, on an object. As these organisms are often very slow to grow, their presence gives a strong visual cue of the object's age.

In the field of computer graphics, techniques used to depict various weathering phenomena can be roughly divided into two categories: physically accurate models and visual models. Physically accurate models are usually simulations, which require detailed knowledge of the phenomenon under inspection. The computational costs of simulations are often high. On the other hand, they provide results accurately reflecting the real world and they may

even be used as predictions for engineering purposes. Visual models aim only to capture the visual appearance of the phenomena–not to reproduce them accurately. One benefit of visual models is that even without deep understanding of the phenomenon, plausible results may be produced. Computational costs of visual models are generally lower than simulations.

Weathering techniques presented in the literature often focus on a single phenomenon. Hirota, Tanoue, and Kaneko (1998) and Hirota, Tanoue, and Kaneko (2000) have simulated cracking of solid objects while Gobron and Chiba (2001b) have simulated peeling of a thin surface layer, such as paint. Bosch et al. (2004) and Mérillou, Dischler, and Ghazanfarpour (2001b) have simulated scratches on a surface. Paquette, Poulin, and Drettakis (2001) have studied dents caused by physical impacts. Great amount of interest has been shown to corrosion of metallic surfaces: Mérillou, Dischler, and Ghazanfarpour (2001a), Codaro et al. (2002), Frankel (1998), and Jain, Kalra, and Kumar (2014), to name a few. Kimmel et al. (2013) and Kimmel and Baranoski (2016) have simulated yellowing of paper. Biological weathering is not well represented in computer graphics, but Desbenoit, Galin, and Akkouche (2004) have simulated growth of lichen.

In addition to models dedicated to a single weathering phenomenon, there exists some models that aim to depict several weathering phenomena with a single model. One of these is the γ-ton model developed by Chen et al. (2005). Their model utilizes particles called γ-tons to transport aging agents such as humidity, heat, and dirt to produce weathering effects such as corrosion and moss growth. A similar but more versatile approach, called μ-ton framework, was later developed by Kider (2012). Both models are implemented as simulations.

Visual weathering models are often implemented by means of texture synthesis, where an existing image texture is modified by some criteria and projected on a virtual object. Lu et al. (2007) used texture synthesis techniques to transfer weathering effects observed in the real-world objects onto virtual ones. Experimental data acquired by laser scanning and photographing was used to determine into which areas and how strongly each weathering phenomena affect. Other approaches to weathering using texture synthesis are presented in Clément, Benoit, and Paquette (2007) and Bellini, Kleiman, and Cohen-Or (2016).

In this study, we propose a weathering model which is not physically accurate but its roots

lie in occurring patterns of physical phenomena. Its purpose is to provide visually believable weathering effects. Compared to existing visual models, our model is able to approximate in which parts of the object the weathering effect should appear and how strong it should be without relying on costly simulations. The approximation is based on object geometry and very limited knowledge of its surroundings. Our model utilizes procedural texturing techniques rather than image-based ones, which allows us to modify the textures on the fly, and renders the cumbersome UV-mapping process unnecessary. Our implementation also supports mixing image-based and procedural approaches.

Intended end user is a 3D-artist who does not necessarily have a background in natural science or computer sciences. We provide tools that can be implemented with commonly available software by hobbyists and professionals alike. Therefore, example implementation presented in this study is implemented with an open source 3D-modeling software Blender, which offers a node programming interface with myriad of ready to use graphical algorithms.

Presented model can depict multiple weathering phenomena. By decoupling the causes of the phenomena from their appearance, we are able to represent several phenomena with a small set of initial parameters. The decision of how to use the parameters on each weathering effect is left to the user, though general guidelines and examples are presented. This approach allows, for example, to create non-realistic weathering effects for imaginary worlds, say, a magical sword made of black metal of which oxidation is accelerated by moonlight and provides pink oxidation products.

The rest of the study is organized as follows. Section 2 presents weathering phenomena and existing implementations to depict them in the field of computer graphics. Section 3 introduces the mathematical formulation of our model. Section 4 briefly describes the nodes used in the actual Blender implementation, which is described in section 5. Results are shown in section 6. Conclusions and suggestions for further research are presented in section 7.

2 Weathering Phenomena

In this section, we cover a few examples of mechanical, chemical, and biological weathering phenomena, their implementations, and parametrization used in the literature. Parametrization is of special interest as there are few parameters that contribute to a wide variety of weathering phenomena.

2.1 Mechanical

Mechanical weathering is a common phenomenon that can be seen on virtually every surface around us. No surface is safe from impacts and scratches caused by other objects. Cracking of surface layers can be seen in painted and plastered surfaces as well as on solid objects, such as concrete structures. Glossy surfaces lose their shininess due to microscopical scratches caused by windblown sand, and frequently used stone path gets polished where people walk.

2.1.1 Cracking and Peeling

Gobron and Chiba (2001a) define a crack as a systematic breaking of material connections. Cracks appear when stress in material exceeds a certain threshold (Gobron and Chiba 2001a; Hirota, Tanoue, and Kaneko 2000). Stress originates from nonuniform expansion and contraction of material due to thermal expansion and external forces (Hirota, Tanoue, and Kaneko 1998). Shape of the cracks depends on material and geometry of an object (Gobron and Chiba 2001a).

It is important to distinguish between surface cracks and volume cracks, since surface cracks are considerably simpler to model. Volume cracks can cause significant deformations in an object and they can even destroy it (Hirota, Tanoue, and Kaneko 1998). Gobron and Chiba (2001a) have modeled the relationship between cracks and stress as a physical simulation. Their simulation is a cycle where stress is first created by material deformation and then relieved by formation of cracks. Their research is focused on ceramics as a multilayer system. Hirota, Tanoue, and Kaneko (1998) have simulated surface cracks as a spring system and later presented volumetric cracking model of drying clay in Hirota, Tanoue, and Kaneko

(a) (b)

Figure 1: Figure (a) shows a metal-coated pool table pocket with impact marks. The coating has been almost completely worn off from the edges and corners. Figure (b) shows how easily accessible outer parts of the kick scooter's frame have suffered a lot of impacts. Coating penetration has caused rust to appear.

(2000).

Cracking of a painted surface leads to peeling (Gobron and Chiba 2001b). Gobron and Chiba (2001b) separates two different kinds of peeling in their peeling model: complete and partial peeling. Partial peeling is particularly interesting for its distinct graphical appearance where the edges of the paint on the peeled area elevate or curl. Their peeling model is built on their previous crack model presented in Gobron and Chiba (2001a). A cracking and peeling model presented in Paquette, Poulin, and Drettakis (2002) is also focused on painted surfaces. Paquette, Poulin, and Drettakis (2002) determine that stress in the paint layer begins to build when the paint dries. Layer thickness, cleanliness and dryness of the surface being painted, and the method of paint application are major factors in initial stress buildup. Even after the paint is dry, it continues to weaken due to moisture, UV-light, temperature changes, pollutants, abrasion and impacts. (Paquette, Poulin, and Drettakis 2002)

2.1.2 Scratches and Impacts

Mérillou, Dischler, and Ghazanfarpour (2001b) divide scratches into two categories: microscopic scratches and scratches that are individually visible. Microscopic scratches cannot be seen as separate scratches but they contribute to the visual appearance of an object as loss of glossiness. Individually visible scratches, on the other hand, are larger but not so

large that they would contribute to the actual geometry of an object. They produce a visible anisotropic effect in reflectance and diffusion properties of the surface. (Mérillou, Dischler, and Ghazanfarpour 2001b)

Based on the ideas above, Mérillou, Dischler, and Ghazanfarpour (2001b) built a physical model of individually visible scratches by characterizing a scratch with a trajectory curve and an M-shaped cross section profile. The characteristic cross section profile was determined by measuring real scratches on metal surfaces. The profile shape is created when a scratching tool removes matter form a groove and displaces it on to the sides of the groove. (Mérillou, Dischler, and Ghazanfarpour 2001b) Bosch et al. (2004) expanded the work of (Mérillou, Dischler, and Ghazanfarpour 2001b) to encompass hardness of the scratched surface, orientation of the scratching tool, and the amount of force applied in the scratching process.

Impacts and scratches are closely related as they share the same birth mechanism: a physical contact with another object. Paquette, Poulin, and Drettakis (2001) have developed a user guided impact simulation that utilizes an impact tool to create impact marks. After selection of the tool and its trajectory, the simulation calculates a mesh deformation for the target object. Impacts can occur individually or in groups.

The literature concerning scratches and impacts tend to concentrate on what they should look like–not where they should appear. Perhaps it is not considered important or not interesting. In Blender modeling society, it is common to place scratches and impact marks on outer corners of objects. Outer corners are readily accessible for other objects, and due to necessarily small contact area applied pressure is greater than in contact with a flat surface.

Figure 1 shows two photographs of objects that have received scratches and impacts. Figure 1a shows how the amount of scratches and impacts is higher on the edges and corners of the metal plate, which implies that surface curvature may be a valid parameter in some cases. In figure 1b the three pipes of the frame of the kick scooter protect each other from the inside, but the out facing surfaces are damaged. This also serves as an example of a joint phenomenon where impacts have caused rusting of the underlying metal. Joint effects are not covered in this study as such, but the idea of rusting induced by coating penetration is used in section 6.

(a) (b)

Figure 2: Figure (a) shows protective corrosion on a building facade. Figure (b) shows destructive corrosion on a gate pole.

2.1.3 Dust

Hsu and Wong (1995) have presented a simple and believable model to describe dust accumulation. For initial dust accumulation they use surface slipperiness as a surface material parameter, and surface normal's projection onto dust source direction as a topological parameter. They make an important notion that even surfaces facing downwards accumulate some amount of dust, though not as much as surfaces facing up. They also take into account that the dust is swept away by wind and scraping objects.

2.2 Chemical

Chemical processes spontaneously occurring in nature are fairly complex phenomena, as they produce varying end-products depending on starting materials and environmental factors. Common features for each type of chemical weathering are related to the environment in which they can occur in the first place, and how said environment affects the speed and strength of the reaction.

2.2.1 Corrosion

Corrosion of metal is a chemical process which requires an electrolyte such as water. There is no need for an actual layer of water–high enough humidity is enough. Corrosion can be divided into protective corrosion where the corrosion stops after certain time, and de-

structive corrosion where material is removed from the object until it gets completely destroyed. (Mérillou, Dischler, and Ghazanfarpour 2001a) Examples of protective and destructive corrosion can be seen in figure 2.

Patination is protective corrosion: corroding metal develops a dense layer of corrosion products onto its surface. It prevents the underlying metal to get into contact with humidity and air greatly reducing the speed of future corrosion. Destructive corrosion, such as rusting, develops a porous layer which allows air and water to pass through it, which in turn keeps the chemical reaction going. (Mérillou, Dischler, and Ghazanfarpour 2001a) Whether certain object goes through protective or destructive corrosion process depends largely on the metal itself: copper usually forms a greenish blue protective layer whereas iron tends to rust thoroughly if given enough time.

When destructive corrosion is so strong that it affects the geometry of the object (in form of cavities and holes), it is called pitting corrosion (Mérillou, Dischler, and Ghazanfarpour 2001a). Jain, Kalra, and Kumar (2014) have created a simulation model for pitting corrosion in which they use a physio-chemical model to simulate the initialization and propagation of corrosion pits as a stochastic process. Frankel (1998) also notes that the initialization of the pits is a random process. In a more recent study Jain et al. (2016) have developed a corrosion simulation system that takes into account object's local geometry. They propose that the initiation of the corrosion is dependent on the surface curvature of the object by $e^{-\frac{1}{c}}$, where c is the surface curvature. Even if it is out of the scope of this study, one graphically interesting property of the corrosion pits is that they tend to form in certain shapes depending on the corroding metal (see Codaro et al. 2002, for details).

2.2.2 Fading

Light induced color fading is not a well-studied area in the field of computer graphics, although it has been widely studied in the field of conservation. Exposure to light over extended periods of time causes color changes in affected material. All dyes or pigments do not absorb light, and thus, do not fade. This causes the material color to shift towards some particular color. The proportion of colorants used in a color mixture defines the resulting

faded color, which is different for every mixture. (Kimmel et al. 2013) In terms of computer graphics, this implies that the resulting color should be defined for every material separately.

Kimmel et al. (2013) have developed a theoretical framework for physically based simulation of time-dependent spectral changes caused by absorbed light. They conducted empirical reflection measurements of paper exposed to a light source as a base for their simulation. Their model concentrates on yellowing of paper. In a more recent study Kimmel and Baranoski (2016) enhanced the model to work in interactive times.

2.3 Biological

There are many types of biological growth that affect the appearance of an object. Most of these species require at least a little light and water but, e.g., fungi do not need light to live. Perhaps the most common species that we see colonizing objects are different kinds of mosses and lichens. We are so accustomed to see them that they often go unnoticed if not paid special attention.

2.3.1 Lichen

Lichens are very resilient, and they can grow on regions too harsh for other species. They prosper on porous materials such as stone, wood and corroded metal. All lichens need indirect light and moisture to grow but direct sunlight is usually too strong for them. (Desbenoit, Galin, and Akkouche 2004) Figure 3 shows how lichen and moss grow readily on shady north-western side of the roof whereas the sunny south-western side is almost completely clean of any biological growth.

Lichen spores take root on surfaces that are easily accessible but protected from wind and rain that might sweep the seeds away (Desbenoit, Galin, and Akkouche 2004). This is a pretty demanding combination to model, and calculating it correctly most likely requires a flow simulation. Lichens grow in clusters where individuals near the center are older and bigger than the ones at the edges (Desbenoit, Galin, and Akkouche 2004). Desbenoit, Galin, and Akkouche (2004) have developed a simulation system, which has distinct phases for spore initialization and growth propagation. The initialization starts with spreading spores

Figure 3: Lichen and moss growing on north-western (left) side, while south-western (right) side is almost clean.

randomly on the surface. Spores in regions of high and low accessibility are removed leaving the ones on areas of medium accessibility. Spores are then aggregated into clusters forming fractal shapes that depend on simulation parameters. Lichen propagation is determined by simulating direct and indirect lighting of the scene. Sunlight is modeled as several separate light sources and moisture conditions are painted by the user. They used an atlas of textured lichen models to populate the scene.

2.3.2 Moss

Mosses are often highly specialized in specific environments (Anttila 2002, p. 37), so there are no simple rules to follow when determining good living conditions. All mosses do share the requirement of having a humid environment. Some species can survive in occasionally dry conditions, but most prefer continuously humid surroundings as they take their water and nutrients straight through their cell walls (Anttila 2002, p. 37). It is worth noting that there can be more than one species of moss even on a single rock, so defining growth parameters according to some single species may not be beneficial. As a curious example Anttila (2002, p. 134) mentions several species of moss that are specialized to grow on surfaces facing down.

For a general approach, we suggest using humidity and lighting conditions for both moss

and lichen growth. Figure 3 supports the plausibility of this approach as the moss and lichen grow mostly on same areas of the roof, though moss does not seem to be as sensitive to direct light as lichen.

3 Weathering Model

Our model consists of three distinct components: principal parameters, derived parameters, and effect generators. Principal parameters are assumed to exist a priori and to be provided by the implementation platform. In the example implementation presented later in section 5, they are provided by the renderer. Derived parameters are constructed from the principal parameters. Effect generators are responsible of creating the visual appearance of a weathering phenomenon, i.e., the weathering effect. They are controlled by principal and derived parameters. Effect generators are agnostic to the conditions given to them. The decision of which parameters should contribute to each effect is left for the user. Figure 4, illustrates some weathering effects and suggested parameters to be used in controlling them.

3.1 Principal Parameters

Height parameter is simply the z-coordinate of a shading point position. It is used to enhance moisture near the ground. *Surface curvature* tells about local curvature of the surface around a shading point. *Surface normal* is a unit vector that is perpendicular to the surface at the shading point. *Accessibility* tells how easily a spherical probe can reach a shading point on a surface. We assume accessibility to be similar to that of presented by Miller (1994), which comes in two flavors: local and global. Local accessibility is affected only by object's own geometry disregarding other objects. Object with global accessibility reacts to other objects as well. (Miller 1994)

3.2 Derived Parameters

Derived parameters are functions of principal parameters or other derived parameters. They are the drivers of our model. Most effects rely on derived parameters to decide whether, and to what extent, a surface should be weathered. Mathematical notation of this section is summarized in table 1.

In following sections, one may notice that most of the values and intermediate results are

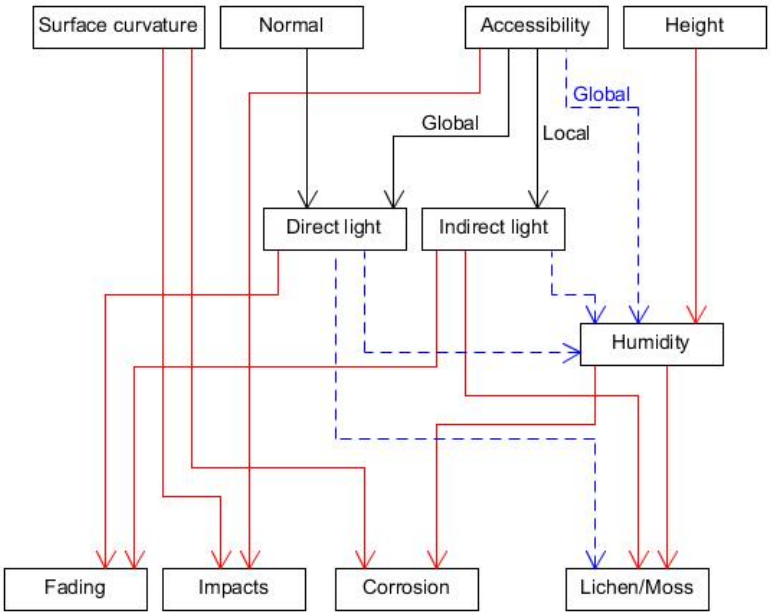


Figure 4: Relations between weathering effects and their parameters Solid red arrow means that the parameter has an increasing impact on the one it points to. Dashed blue arrows imply decreasing impact. Black solid arrows mean that the parameter is needed for calculations in some general sense.

either clamped or scaled to the interval $[0, 1]$. There are two reasons for doing so. First is to offer consistency with Blender's default nodes, which are most often restricted to this interval. Second, it is easier to combine different parameters together when they are confined into some known interval.

3.2.1 Direct Light

When thinking about strong direct light sources in our everyday life, the first thing coming to mind is most probably the sunlight. Sunlight is what bleaches our car paint and hair dyes by breaking the chemical bonds between colorant molecules. It warms the surfaces it shines on drying the up the morning dew and traces of nightly rain. This is all true by intuition, and intuition has a big role in whether a virtual world is considered believable or not.

The power of electromagnetic radiation per unit area (W/m^2) received from the Sun is called solar irradiance. In order to form a good model of solar irradiance one should take into

Table 1: Mathematical notation.

Notation	Explanation	Range
$\mathbf{v}$	Vector v	
$\|\mathbf{v}\|$	Euclidean norm of vector v	
$\hat{\mathbf{v}}$	Unit vector v	
$[]_a^b$	Clamping to an interval	$[a,b]$
z	Height	$[0,1]$
a_l	Local accessibility	$[0,1]$
a_g	Global accessibility	$[0,1]$

account latitude, longitude, altitude, day of the year, average cloudiness of the area, and so on. That kind of model would be far too complex to use, and too expensive to calculate for a visual model.

Let us, instead, model the yearly average of solar irradiation by thinking the Sun as a point light source that moves along a half circle, which is part of a hemisphere above an observation point. If we imagine our observation point to lie on the equator, the half circle cuts the imaginary hemisphere in half and the zenith point is straight up. If we knew some values for the intensity in different positions, we could fit some model into it, and integrate over the half circular path to calculate the overall amount of radiation. This is still a bit complicated, so let us take just three points along that half circle: one from the middle and two symmetrically from both sides.

Before defining our direct light model, we should get familiar with a few radiometric equations. *Intensity I* of a light source is the measure of radiant energy leaving from a point into direction of photon flux Φ per unit solid angle ω (Glassner 1994). It can be defined in differential form as

$$I = \frac{d\Phi}{d\omega} \qquad \left[\frac{\mathrm{W}}{\mathrm{sr}}\right]. \tag{3.1}$$

Irradiance E is a proportion of the photon flux Φ arriving at unit area A (Glassner 1994)

$$E = \frac{d\Phi}{dA} \qquad \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]. \tag{3.2}$$

When the Sun is setting, the angle between the Earth surface and the Sun is growing smaller, the area that its light shines on grows larger and the amount of radiation per unit area grows smaller. To account for the surface normal not pointing directly towards the light source we get *radiance L*, which is the power arriving at a surface per unit solid angle and per unit projected area

$$L = \frac{d^2\Phi}{d\omega\, dA\, cos\theta} \qquad \left[\frac{\mathrm{W}}{\mathrm{sr}\,\mathrm{m}^2}\right], \tag{3.3}$$

where θ is the angle between the surface normal and the direction to the light source (Glassner 1994).

We can discard the differential notation and define radiance with vectors. Let us think that the solid angle of intensity is so small it is reduced into a direction. The flux to that direction is now some scalar value. Something that has a direction and a magnitude can readily be represented as a vector, so we get that $I = \mathbf{v}$, where flux $\Phi = \|\mathbf{v}\|$ and $\omega = \hat{\mathbf{v}}$. As is common in vector representations in computer graphics, we flip $\mathbf{v}$ to point from the surface towards the Sun.

To define radiance, we need to know into how large area the incoming flux is projected to. For this purpose, a unit surface normal $\hat{\mathbf{n}}$ is enough to represent the irradiance on a surface. We can use a unit length normal, because the incoming flux is the one provided by intensity and contained in the magnitude of $\mathbf{v}$. In the definition of radiance, the term projected implies we are projecting the incoming flux onto some area, which we can express as a scalar projection from $\mathbf{v}$ to $\hat{\mathbf{n}}$

$$L = \mathbf{v} \cdot \hat{\mathbf{n}} = \|\mathbf{v}\|\|\hat{\mathbf{n}}\| cos\theta = \|\mathbf{v}\| cos\theta. \tag{3.4}$$

This formulation is actually Lambert's cosine law from the 18^{th} century. It tells us that the radiance depends only on the angle between the vectors, which is what we want–for a

constant flux, increase in the angle increases the projected area, which results in decreasing flux density, which is radiance.

We can use this idea separately for our three sun positions arriving to an equation for direct light

$$l_d(a_g, \hat{\mathbf{n}}) = \left[s a_g \left(\max(0, \hat{\mathbf{v}} \cdot \hat{\mathbf{n}}) + \max(0, \mathbf{v}^+ \cdot \hat{\mathbf{n}}) + \max(0, \mathbf{v}^- \cdot \hat{\mathbf{n}}) \right) \right]_0^1, \tag{3.5}$$

where $s \in [0,1]$ is just a user-specified scaling constant. The same constant will be heavily used from now on without further notice. Notation $[\ ]_a^b$ means that the result is clamped to closed interval $[a,b]$. Radiance received by a shading point is the sum of non-negative dot products over all three vectors. Non-negativity restriction must be placed because otherwise back-facing surfaces would be able to reduce the value of radiance. The global accessibility term a_g will be explained in detail later.

Vector $\mathbf{v}(r, \theta, \varphi)$ is our main sun vector defined in spherical coordinates, where r is radius, θ is the elevation from xy-plane in range of $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ radians, and φ is the azimuth measured from the positive x-axis. The azimuth represents the direction to the south (when in northern hemisphere), and the elevation is the angle between the ground and the Sun. The radius contributes only to the magnitude of the vector and is set to $r = 1$ for the zenith point vector $\mathbf{v}$.

Approximating sunlight with just one vector would provide difficulties with, for example, boxes that are orientated at angle φ, which, in the worst case, would cause only one face to receive sunlight and other five none at all. Our secondary sun positions, defined as $\mathbf{v}^+ = (k_1 r, k_2 \theta, \varphi + k_3)$ and $\mathbf{v}^- = (k_1 r, k_2 \theta, \varphi - k_3)$, greatly improve the situation. Constants $k_1, ..., k_3$ can be defined to approximate different kinds of environments. The first constant $k_1 \in [0,1]$ defines how much of the main vector's radiation power the side vectors should have. In essence, it takes into account atmospheric scattering of light, which is greater when the Sun is rising or setting, because the light has to travel longer distance before reaching the surface of the Earth. Constant k_2 controls the elevation of the side vectors. Using low values imply that the side vectors are selected near the Sun's rising and setting positions, which may be relevant for northern regions where the Sun does not set in the summer, whereas high

values mean that morning and afternoon radiation is emphasized. Constant k_3 is the spread of the side vectors. Setting it to some large value could be, again, used to model northern regions, lower values could be used when closer to the equator. There is no real need to restrict k_3 to any specific interval, as the side vectors will just orbit the main vector as the values increase or decrease, but for practical use, it is perhaps better to restrict it to a value less than half a circle.

Working with just a single shading point at a time is problematic, since we do not know anything about the surrounding geometry. This causes our model to interpret any surface facing south to receive sunlight–even if there is other geometry obstructing it. If we had an access to a ray tracer, we could cast a ray towards each sun position and null the value if none of them can be reached. As an ad hoc solution, we use global accessibility a_g to null the sunlight in hard to access areas. By using the global version of accessibility, we are also able to take other objects in the scene into account.

Although this solar model is very crude, it has some merits due to its simplicity. Manipulation of just three vectors can be done with a relatively small set of parameters, which should make it easy to understand and use. Also, because the calculations must be carried out for every shading point in the scene, the computational cost of the model needs to be relatively low. It is also worth noting that in order to build a catch-all model, we are neglecting all material properties and assume that all received radiation is absorbed.

Even though we named the equation as direct light, it is fairly easy to find other usages for it. It works with any kind of phenomena that has some directional component. For example, a vicinity of a seashore could provide salty water to nearby surfaces promoting corrosion (reader interested in corrosion in marine environments, see Cole et al. (2004)). This could be modeled by setting elevation $\theta = 0$ and disabling side vectors by setting $k_1 = 0$. Similarly, a busy street next to a building (an example used in Dorsey, Pedersen, and Hanrahan (2006)), which acts as a source for pollutants and dust, could be modeled by setting elevation to some negative value. In figure 20b a fairly narrow spread and low elevation was used to depict sunlight entering from windows of a room.

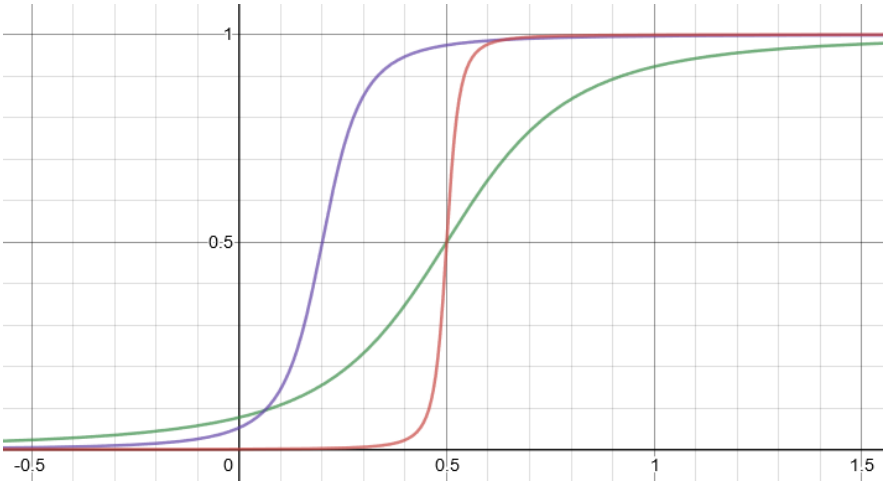


Figure 5: Plots of sigmoid function in equation 3.7. Green with $s_o = 0.5$, $s_s = 10$, red with $s_o = 0.5$, $s_s = 1000$, and purple with $s_o = 0.2$, $s_s = 100$

3.2.2 Indirect Light

There is always some ambient light that is scattered by the atmosphere and reflected from surfaces. It does not have a specific source and it can be thought to be emitted from everywhere. The simplest way to implement our *indirect light* would be to use ambient light as a constant value for every shading point. However, because of the importance of indirect light to biological growth, we should at least consider that deep crevices and holes do not receive as much ambient light as open areas. This can be done using the accessibility parameter again. The light is emitted from everywhere, so there is no need to account for other objects shadowing the object and the local version of accessibility can be used. The amount of indirect light l_i is defined as

$$l_i(a_l) = s\,S(a_l), \tag{3.6}$$

where function $S(x)$ is a sigmoid function.

We make heavy use of the sigmoid function from now on. It is used to scale any real value to range $[0,1]$ and to control how fast the transition from zero to one occurs. It can also be used to control the position of the transition. We define the sigmoid function as

$$S(x) = \frac{x - s_o}{2\sqrt{\frac{1}{s_s} + (x - s_o)^2}} + \frac{1}{2}, \tag{3.7}$$

where s_o is a constant offset that moves the point where the derivative of S reaches its maximum value. Constant s_s defines steepness of the slope. Figure 5 shows the sigmoid function plotted with few values of s_o and s_s.

3.2.3 Humidity

Presence of water is a major factor in many weathering phenomena such as rusting and biological growth. In case of rusting, as mentioned earlier, there is no need for an actual layer of water: some amount of humidity on the surface is enough. Organisms that do not have roots can usually take the water they need from the moisture in the air, so biological growth can be accounted for by using humidity as a parameter.

Our humidity model begins with the assumption that some base amount of humidity is always present. Due to moisture in soil, objects that lie on the ground for long periods of time tend to be wet. To take the moisture in soil into account, we use the height parameter. Combining it with the base humidity we get function α that increases humidity

$$\alpha(z) = S(1 - z) + h_b,$$

where h_b is a constant base humidity in the range of $[0, 1]$, and z is the z-component of the shading point's position vector. Because the humidity should rise when going closer to the ground, we use the complement $1 - z$. Sigmoid function is again used to control how far the effect of z is visible and how steep the transition is.

Average humidity of a surface does not only depend on how easily it gets wet, but on how well it dries up too. We model drying of the surface with two factors: how much light energy is absorbed and how well air can circulate in the vicinity of the surface. The idea behind using light energy as drying parameter is simple: the surface absorbs light energy transforming it into heat, which conducts to the water molecules. Molecules with more energy can break

free of their liquid bonds and transit into gaseous state.

Even without light, water has some rate of evaporation. Evaporating water makes the air near the surface more humid, which drops the evaporation rate. Evaporation rate can be held higher by continuously replacing the humid air with dry air. Air moves freely on open areas, so we can once again use accessibility to determine areas that should dry up faster. Based on measurements on real objects, Lu et al. (2006) found that on an arbitrary evenly wetted object, areas with small accessibility stay wet longer than areas with high accessibility.

To model drying of a surface, let β be a function of accessibility and surface normal

$$\beta(a_g, a_l, \hat{\mathbf{n}}) = c_1 S(a_g) + c_2 l_d(a_g, \hat{\mathbf{n}}) + c_3 l_i(a_l),$$

where $c_1, ..., c_3$ are scaling factors. They might seem redundant as the light equations already have internal factors defined, but having a second set of scaling factors here makes it easier to use a single light equation as an input for several other equations as needed. The first term $c_1 S(a_g)$ models airflow, the second term $c_2 l_d(a_g, \hat{\mathbf{n}})$ models direct light, and the last term $c_3 l_i(a_l)$ models indirect light. Indirect light should only have little effect on the humidity. For saving computational resources, it might be reasonable to use some small constant value instead of the actual indirect light model.

The amount of humidity h is now just a simple scaled difference of α and β clamped to the interval $[0, 1]$ as usual

$$h(z, a_g, a_l, \hat{\mathbf{n}}) = [s(\alpha - \beta)]_0^1. \tag{3.8}$$

3.3 Effect Generators

We use the term *effect generator* for entities that output a single value for texture selection, and when applicable, a three-component color vector for texture modification. All of them use the same base set of inputs: time t, durability d, environmental encourage e_e, and environmental discourage e_d. Other inputs depend on the actual implementation, which

will be discussed in section 5. Base inputs are divided into two pairs: time–durability and encourage–discourage.

The idea behind using durability in conjunction with time is that an object may consist of several different materials, which might weather differently, even if the same amount of time passes for both. For example, if we have an iron statue on a stone pedestal, the statue should begin to rust fairly quickly, in months perhaps, but developing a visible layer of moss onto the stone surface would most likely take years. Thus, we would set the durability of the effect generator generating rust patterns to a lot lower value than the generator for moss.

Time-durability relation is a function of time

$$t_d(t) = \frac{t}{d}, \tag{3.9}$$

where durability remains constant. As the durability is a property of a material, having a constant durability implies that an object made of that material has consistent quality throughout the entire object.

The second pair forms the overall environmental factor e as a function of a shading point $\mathbf{p}$:

$$e(\mathbf{p}) = e_e(\mathbf{p}) - e_d(\mathbf{p}). \tag{3.10}$$

In practice, this is the equation where we input all the parameters defined earlier.

3.3.1 Spotty

Spotty generator can be used for anything that should have a spotty look, e.g., rust, moss, or coat coloring of a dalmatian. We will formulate the Spotty generator using Musgrave noise, which can be thought of as having irregularly shaped islands rising from a sea.

If we think the spotty generator in terms of rust for a while, we have a few conditions that must be met. First, there should be spots of rust appearing in random places–these are the islands of the Musgrave function. Second, the spots should grow larger as time passes–this

is accomplished by lowering the sea level. Third, the spots should not move around as they grow–this is automatically true from the Musgrave noise definition covered next. The same conditions apply to our other example phenomena too.

Originally Musgrave noise was intended to be used in generation of eroding terrain (Musgrave, Kolb, and Mace 1989). Musgrave noise M is an iterative process where first, a Perlin noise function N (Perlin 1985) is calculated with some initial frequency, which is increased in each step an added to the total result with decreasing weight. One definition of M is

$$\begin{aligned} a_0 &= c_s(N(\mathbf{v}_0) + c_t) + c_0, \\ a_i &= c_s(N(\mathbf{v}_{i-1}\ \lambda) + c_t)\omega^i + a_{i-1}, \end{aligned} \tag{3.11}$$

where $\mathbf{v}_0$ is the initial coordinate vector for the Perlin noise. Constant c_s is an overall scaling factor, and c_t is used to shift the coordinate vector. Varying scaling factor $\omega \in [0,1]$ is raised to an increasing power in every step, which effectively lessens the contribution of later steps. Lacunarity λ represents gaps between used frequencies. The most important term for us, c_0, controls the sea level. (Musgrave, Kolb, and Mace 1989)

We use the Musgrave function to create spots controlled by time and environment just by altering the sea level. One can think about an uneven sea which reveals more islands when the effect is strong. The Spotty generator function g_s is defined as

$$g_s(t,e) = sM(c_0), \tag{3.12}$$

where $c_0 = et_d$ and all other factors in the Musgrave function are held constant.

3.3.2 Roughy

In the field of computer graphics, large amount of work has been contributed to model interactions between light and a surface. In the Phong reflection model (Phong 1975), specular reflections were introduced as a function of incident light direction and viewing direction.

Later, Blinn extended the model using the concept of microfacets for more realistic lighting (Blinn 1977). Blinn's model is nowadays called the Blinn-Phong model. In microfacet theory, a surface is considered to consist of tiny perfectly reflective mirrors that are arbitrarily aligned on the surface. The amount of light reflecting to the direction of the viewer depends on how great portion of the microfacets is oriented favorably. (Blinn 1977) In a sense, microscopic roughness of a surface measures how large portion of the microfacets is aligned along the surface. Small roughness creates sharp specular reflections whereas high roughness creates larger and duller reflections. The microfacet theory is still widely used in many different lighting models and renderers–including Blender's Cycles renderer (Steinmetz and Hofmann 2018). For practical purposes of this study, it is sufficient to know that Cycles shader nodes accept roughness values between zero and one ("Blender 2.81 Reference Manual" 2019).

In most cases, surface roughness of a real-world object would not remain constant over its time of existence, as the surface would be affected by various weathering phenomena. A car would be swept by sand and other particles carried by the wind, and sunlight would break the chemical bonds of the clearcoat perhaps rendering it more brittle resulting in microscopical damage. Even a boulder in a sea gets swept by debris carried by the water. All these physical and chemical weathering phenomena would contribute to the roughness of the surface.

We can model the change in roughness by defining some initial roughness for a new object letting it change as time passes. The Roughy generator is defined as

$$g_r(t,e) = [S(et_d)]_{l_{min}}^{l_{max}}, \tag{3.13}$$

where l_{min} is the lowest allowed roughness, which is usually the initial roughness, while l_{max} is the highest possible roughness. The upper bound is mainly for convenience: it only clamps the sigmoid function value. The limits can be omitted by setting $l_{min} = 0$ and $l_{max} = 1$ giving the sigmoid function full control over Roughy's behavior. It is worth noticing that, for any realistic surfaces, roughness should not be allowed to reach value zero, as it would mean a pure Lambertian surface, which does not exist in the real-world. The same applies for value one, which would be a perfect mirror.

For inverted effect, e.g. a rough stone getting polished by some mechanism, one can use a complement of the sigmoid value $1 - S(et_d)$. The limits still apply, but their interpretation has changed: now l_{max} is the initial value. Multiplying the input value with a noise function adds a lot of detail and character to the result, but because it is not an essential part of the Roughy generator, the decision to do so is left to the artist and not incorporated into the actual model. The same applies for the rest of the generators too.

3.3.3 Fade Towardy

Fade Towardy generator is used to depict phenomena that cause color changes in a surface. Color changes may occur in a material when its chemical properties change over time. One clear use case is yellowing of paper mentioned in section 2.2.2.

As the name suggests, Fade Towardy fades a base color towards some other color with a factor g_f. Following the formulation of generators presented earlier

$$g_f(t,e) = S(et_d). \tag{3.14}$$

If we have an original RGB (red, green, blue) color vector c_1, and some other color c_2 at the end of the color changing process, we can formulate a color mixture c during the process by

$$c = c_1(1 - g_f) + c_2 g_f. \tag{3.15}$$

Equation 3.15 is exactly how Cycles implement mixing of colors (Steinmetz and Hofmann 2018).

3.3.4 Desaturaty

Desaturaty can be thought as a different version of Fade Towardy in which, instead of fading towards some particular color, the saturation of given color is decreased. It can be used to depict fading of paint, for example. In contrast to Fade Towardy, which operates on RGB vectors, Desaturaty operates with HSV (hue, saturation, value) vectors. It takes an HSV

color and decreases the saturation value until some lower limit $l_{min} \in [0,1]$. We define the Desaturaty generator as

$$g_d(t,e) = [1 - S(et_d)]_{l_{min}}^{1}, \tag{3.16}$$

where the upper limit of one is what Blender interprets as the original saturation ("Blender 2.81 Reference Manual" 2019).

4 Virtual Worlds and Blender

Before presenting the actual implementation, we should take a quick look of the implementation tool selected for this study. We use an open source 3D-modeling software called Blender, which is maintained by Blender Foundation ("Blender Foundation" 2019). Utilizing an open source software contributes in our goal to make the model available for wide variety of people regardless of their monetary resources and programming skills. In this section, Blender's Cycles renderer (hereafter, Cycles for short) is briefly covered for the parts used in the implementation.

In aspects concerning solely Blender and Cycles, we rely on Blender manual ("Blender 2.81 Reference Manual" 2019) and Cycles Encyclopedia handbook (Steinmetz and Hofmann 2018). Although not being scientific sources, excluding analysis of the source code, these are probably the most accurate sources for actual behavior of the software. Blender has some naming conventions that differ from general computer graphic terminology, which we will point out along the way.

The whole purpose of computer graphics is that we want to see something that does not exist in the real world–we want to see a virtual world. The virtual world may be a video game with a purpose to entertain, it may be a simulation of a control room of a nuclear plant meant for training, or it may be an architectonic model of some historical building meant to educate. In order to see anything at all we obviously need light and a light detecting sensor. In the real world we sense light with our eyes, but in virtual world we use a camera. Last thing we need is something to be seen–an object. The object can be anything that interacts with light in some way: a brick, smoke, water, glass, etc. Surface of the object is a collection of faces joined by their edges.

A collection of lights, cameras and objects is called a scene. A typical scene has several light sources, several objects and a camera. A renderer is something that has perfect knowledge of the scene. Its purpose is to decide the color and intensity of the pixels in our computer screen. In other words, the renderer knows how a ray of light emits from the light source, how it interacts with the objects, and into which part of the camera the ray enters. The key

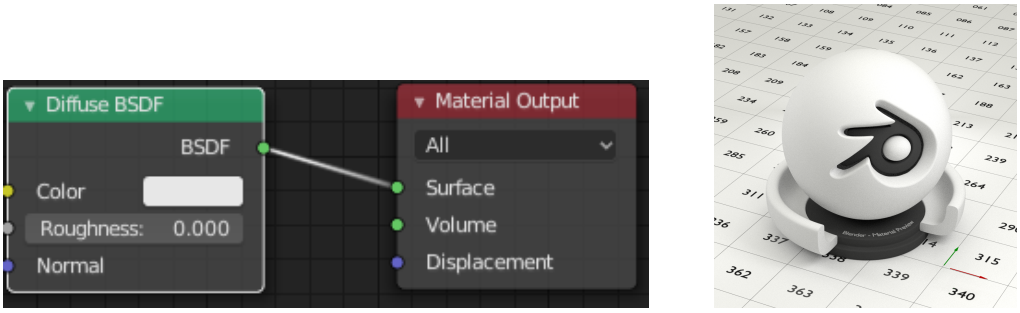
point here is how the light interacts with the object. The light source is fundamentally a very simple construct: it has some intensity, and some shape which governs the direction of the rays. Also, the camera is rather simple: it is a virtual sensor with array of virtual pixels that detect light.

The object, on the other hand, has endless ways of manifesting itself that can look drastically different. When a ray of light hits the surface of an object, it can reflect, scatter deeper into the object or transmit through it. In computer graphics, a *shader* is something that decides how rays of light interact with the surface of an object, but in Blender, it is called a material, whilst the term shader refers to a special shader node. There can be many (and usually this is the case) shader nodes in a single material. (Steinmetz and Hofmann 2018)

4.1 Path Tracing in Cycles

Cycles is a path tracer type renderer. The path tracing process starts by casting a ray through a pixel of the camera. This is called a camera ray. When colliding with an object, the type of the ray is changed according to the material of the surface it hit. If the surface material is, say, a glossy material, the ray is reflected, and its type is changed into a glossy ray. The ray might next hit a diffuse material in which case the ray will bounce into a random direction as a diffuse ray. The ray will keep on bouncing off the surfaces until maximum amount of bounces is reached, or the ray hits a light source, which terminates the ray. When the ray is terminated, intensity and color are returned to the pixel it was cast from. This is called a sample. The mean of all samples is used as the final color and intensity for the pixel. (Steinmetz and Hofmann 2018)

Usually a material has several shaders mixed together, a diffuse shader and a glossy shader for example. If this is the case, Cycles randomly picks one of them. The probability of picking one of the shaders can be controlled by a mixing factor. Also, shaders with brighter colors are more likely to get chosen. (Steinmetz and Hofmann 2018)



(a) (b)

Figure 6: A simple white diffuse material setup is shown in figure (a). Figure (b) shows how it is rendered on an object.

4.2 Nodes in Cycles

There exists close to one hundred nodes for Cycles materials and they have several input and output sockets creating endless amount of possible combinations. (Steinmetz and Hofmann 2018) It is impossible to cover even the most important ones in depth in this study, so an interest reader is encouraged to take a look at Steinmetz and Hofmann (2018) and "Blender 2.81 Reference Manual" (2019).

A material in Cycles needs at least one shader node and a Material Output node. Interconnected group of nodes is read from left to right: the dots on the left are node inputs, and the dots on the right are node outputs. The dots are color-coded so that the yellow dots are for colors, the green ones for shaders, the blue ones for vectors, and the gray ones for numbers. Since colors and vectors are both expressed as 3-vectors, they can be connected to each others' sockets, and usually produce some meaningful result. Vector components may have values outside of Cycles' default range $[0, 1]$, but colors cannot (or rather, they are clamped and values outside of that range do not convey any additional information). If a vector or color is connected to a value socket its first component will be used as the value. (Steinmetz and Hofmann 2018) Cycles materials are built and modified with Blender's Node Editor. One can think of it as a graphical programming interface for shader programming (shader in normal computer graphics sense). A minimal example of a material in the Node Editor is shown in figure 6.

Many inputs can take a map. Connecting a constant input to a socket will pass that value

for every shading point. However, connecting something that varies from point to point produces more interesting and complex behavior. For example, connecting a constant RGB-vector (red, green, blue) to a shader node's color input would produce an object with a solid color, but connecting a texture to it would result in surface with varying color. Nodes are connected to each other with lines called noodles. (Steinmetz and Hofmann 2018)

Nodes can be combined into node groups that externally look like nodes themselves. To provide an analogy to general programming, one can think node groups as subroutines. Arranging nodes into node groups have the same benefits as arranging program code into subroutines: reusability and exposing only the essential parameters.

4.2.1 Shader Nodes

Before introduction of the Principled BSDF node, the usual way of creating a material was to use two or more shader nodes and combine them with a Shader Mix node. Mixing factor of the node was usually controlled by a Fresnel node with an appropriate index of refraction. This would give the material realistic reflection properties that change according to viewing angle. Now all this hassle can be omitted with the Principled BSDF node. In this study, we will not concentrate on how to build the shaders that are used in the examples, but because Cycles material cannot exist without a shader node, it is good to know what they look like, at least.

Let us take a look at the most important inputs of the Principled BSDF node shown in figure 7. The Base Color input affects the diffuse color of the material. Metallic input is set to zero for dielectric materials and to one for metals. Metallic reflections are colored with Base color, whereas dielectric reflections are white. Roughness controls the microscopic roughness of the surface: values close to zero produce polished look, whereas values close to one look matte. Normal input is for changing the normal of the faces, which is used to fake geometry. (Steinmetz and Hofmann 2018)

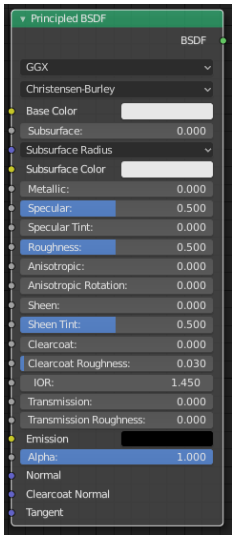


Figure 7: A Principled BSDF shader node.

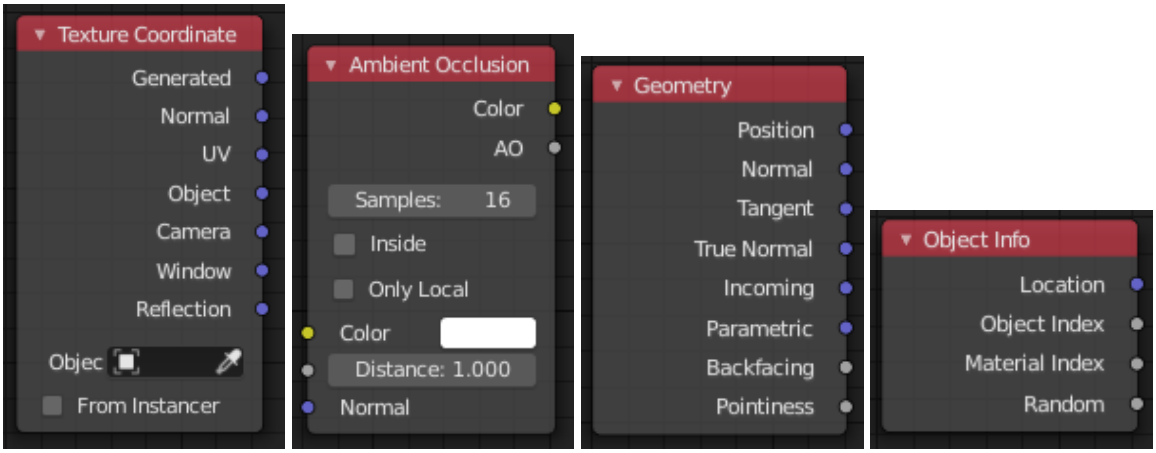
4.2.2 Input Nodes

The only purpose of the input nodes is to provide outputs for other nodes. They do not take any inputs themselves. The outputs are usually connected to texture nodes.

The Texture Coordinate node shown in figure 8a provides coordinates of the textures. This node can operate in object, world, UV, camera, and screen spaces. The Generated output provides texture coordinates in the range of $[0, 1]$ over the bounding box of an undeformed mesh. This is often used as a default input for texture nodes when no input vector is connected. (“Shader Nodes” 2019)

Ambient Occlusion (AO) node shown in figure 8b computes how much of the hemisphere above the shading point is occluded. Parameter Only local selects if other objects than the object itself are taken into account. (“Shader Nodes” 2019) This coincides with the idea of global and local accessibility presented in section 3.1 and is used as such from now on. The distance input defines the maximum distance of surfaces that are still contributing to the AO (“Shader Nodes” 2019).

The Geometry node shown in figure 8c provides information about the point being shaded. All coordinates are defined in the world space. True Normal is the unaltered normal of the point, while Normal output takes into account possible smooth shading and bump map-



(a) Texture coordinate node (b) Ambient Occlusion node. (c) Geometry node. (d) Object info node

Figure 8: Input nodes.

ping. Pointiness output approximates the curvature of the surface around the shading point. ("Shader Nodes" 2019) The Geometry node provides us with most of the principal parameters needed: position for height, the surface normal, and surface curvature. Often, usage of global position is not desired, in which case, Generated output of Texture Coordinate node is used for the height parameter.

The Object Info node shown in figure 8d provides information about an object instance. Random output gives a random number unique to the object. ("Shader Nodes" 2019) This is very useful if we use several copies of a single object with same material, and want the material to have a different pattern in each object. In practice we can use the random number to modify the shared material in some way.

4.2.3 Texture Nodes

Texture nodes are used for materials whose properties should depend on the location of a shading point (Steinmetz and Hofmann 2018). For a simple example, we could create an image-based texture where an image would be spread over an object. For procedural materials, there are numerous textures that provide some sort of pattern such as a checkerboard or brick wall pattern. In essence, procedural textures are mathematical functions that create an

(a) (b)

Figure 9: Figure (a) shows a Musgrave Texture node and (b) shows a Noise Texture node.

endless, seamless color field (Steinmetz and Hofmann 2018). For formal definition of noise functions and procedural noise, and classification of noise functions, see Lagae et al. (2010).

Using procedural textures instead of image-based ones offers some advantages. Procedural textures are endlessly scalable so even if the camera is close, or very far away, the texture is always sharp, which is not the case with image textures. Image textures are often too small to cover an entire object, so copies of the same image are tiled side by side, which often results in discontinuities. This is not a problem with procedural textures, since they are continuous in the whole 3D-space. Objects with image textures must be UV mapped in order to properly project a two-dimensional image onto a three-dimensional object, which is usually made by hand and can be very time-consuming. As procedural textures are already defined in 3D-space, there is no need for UV mapping.

All texture nodes have a Vector input which can be manipulated to control how the texture is laid on the object. All procedural textures have a Scale input that is used to adjust the size of the *texture*. Increasing the size of the texture results in pattern looking smaller because a larger pattern is projected onto the same area. (Steinmetz and Hofmann 2018)

In terms of control, the Noise Texture node shown in figure 9b is the simplest of the noise textures. It is an implementation of Perlin noise (Steinmetz and Hofmann 2018). For technical details about the original Perlin noise see Perlin (1985). The Musgrave Texture shown in figure 9a is very difficult to control and the resulting pattern is often all white or all black.

(a) (b)

Figure 10: Hue Saturation Value node is shown in figure (a) and Mix node in figure (b).

Depending on the selected function, some inputs may be ignored, and some may provide effects only if some other input is between a certain range (Steinmetz and Hofmann 2018).

4.2.4 Color Nodes

Color nodes are used to control and modify colors. The Hue Saturation Value node shown in figure 10a is used to alter the hue, saturation and value of the input color ("Blender 2.81 Reference Manual" 2019). The Mix node shown in figure 10b mixes two colors together as in equation 3.15. The mix is controlled by the Fac input so that Fac of zero outputs Color1 and one outputs Color2. ("Blender 2.81 Reference Manual" 2019)

4.2.5 Converter Nodes

The most important converter nodes are the Math node and the Vector Math node. Math node offers common arithmetic and trigonometric operations as well as a good range of other functions. It includes a checkbox for clamping the value to the range $[0, 1]$. Vector math node provides basic vector operations such as addition, subtraction, dot product and cross product, and vector normalization. Output is either a vector or a value–most operations support both. ("Shader Nodes" 2019)

5 Implementation

In this section, we will present how to apply the ideas developed in section 3 in practice. We concentrate on how to connect all the different parameters and generators to create visual results whereas the actual implementations of the equations in section 3 can be found from appendix A.

We use the Material Preview object of Cycles Material Test Scene [1] to demonstrate the usage of our model. The Material Preview object is often used in Blender community to demonstrate materials. It is a simple object, yet it has enough geometry to showcase most materials.

We changed the scene so that the red (pointing along positive x-axis) and green (pointing along positive y-axis) arrows can be seen in rendered images. Material Preview object's base and Blender logo use the same gray diffuse material in all images: only the material of the sphere and the ring below it use our custom materials.

Images rendered in grayscale show how raw values are distributed over the object. Black color corresponds to value zero and pure white to one. In these images, we use an Emission shader node, which does not react to the lighting of the scene. Images rendered with materials that aim to mimic real world materials utilize Principled BSDF shader node, but as creation of actual materials is out of the scope of this study, their implementations are not shown.

5.1 Derived Parameters

Usage of derived parameters is fairly simple, but visualizing them on actual geometry helps to understand their behavior. Later, when they are connected in more complex ways, their effect is more difficult to see. We have already shown in section 4 how principal parameters relate to default Cycles nodes, so there is no need to cover them again.

1. Downloaded from https://blenderdiplom.com/en/downloads/584-download-cycles-material-test-scene.html. Original version created by Robin Marin. Licensed under CC BY 3.0 https://creativecommons.org/licenses/by/3.0/

5.1.1 Direct Light

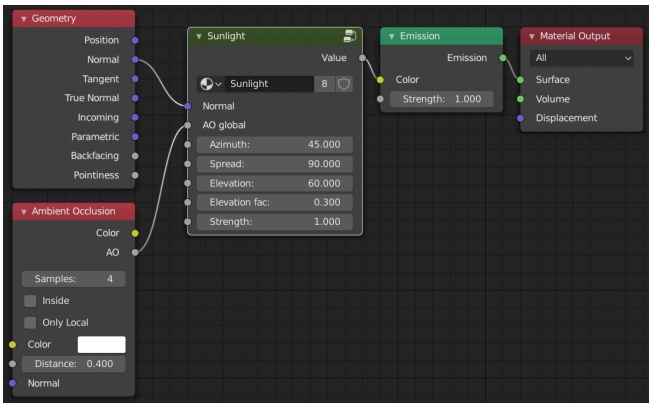
Direct light node group shown in figure 11a implements equation 3.5 and is renamed as Sunlight. From technical point of view, the most important settings are found from the Ambient Occlusion (AO) node. Sample count defines how accurate its value will be. AO is a costly operation in terms of computing time, so it should be kept as low as possible while still acquiring decent results. We have used sample counts from four to eight. Using less than four samples results in significant loss of accuracy, probably because the sampling ray can rarely reach another surface, which can be seen as mostly white result with low contrast. More than eight samples cause uncomfortably long evaluation times, which hinders the model's trial and error nature. The distance input of the AO node should be adjusted as model dimensions require, i.e., larger models need greater distance value to produce the same effect. Fairly large values should be used in order to produce large shadowed areas, as otherwise the only shadowed areas will be small crevices.

Figure 11 shows the node setup and two rendered images of the Sunlight. From the color difference between the top of the sphere and up-facing parts of the ring, it is clear that our trick with AO does shadow the ring. The difference in distribution of energy over the object in northern and equator regions can be seen in figures 11b and 11c.

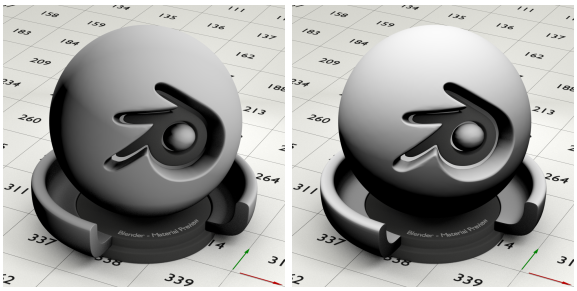
5.1.2 Indirect Light

Indirect Light node group in figure 12a implements equation 3.6. An AO node with Only local checkbox checked provides local accessibility needed for Indirect Light. Offset and contrast parameters control the sigmoid function in equation 3.7, so that Offset is s_o and Contrast is s_s.

Proper selection of the sigmoid parameters depends on the use case. If the indirect light is used for biological growth, where it is a vital parameter fighting against the discouraging effect of direct light, sharper transitions and low offset may be used. If, on the other hand, it is used as a discouraging factor for a Humidity node group, its effect should not be so pronounced.

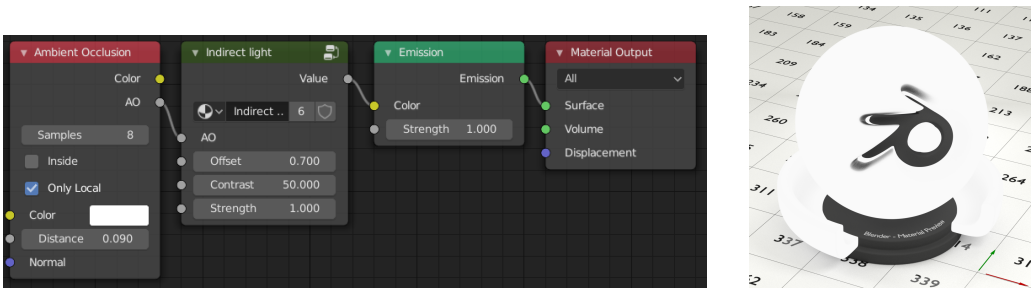


(a)



(b) (c)

Figure 11: Usage of the Sunlight node group. Both rendered images have 45° azimuth (south is towards the top right corner of the image) and strength of one. Subfigure (b) depicts the sunlight as in northern regions with low elevation, low elevation factor, and wide spread. Subfigure (c) depicts sunlight on the Equator with high elevation, high spread and high elevation factor.



(a) (b)

Figure 12: Indirect light node setup is shown in figure (a) and resulting rendered image in (b). Low distance value in AO node produces mostly white object with black in the smallest crevices.

5.1.3 Humidity

Humidity node group, which implements equation 3.8, has quite a few inputs as can be seen from figure 13a. Position is used for height parameter. Offset and contrast inputs control sigmoid functions, and inputs ending with *fac* correspond to scaling factors $c_1, ..., c_3$ in the equation. AO node that is passed straight into Humidity is used for the airflow term.

Using three separate AO nodes makes Humidity expensive to compute. The Sunlight and airflow both use global AO, but they need to have different distance value, as airflow should be restricted only on fairly small spaces, whereas sunlight should produce large shadow areas. Indirect light has only a small effect on the overall result (with factor of 0.1 as used in the example), so it could have been set to some small constant value to reduce the computational cost.

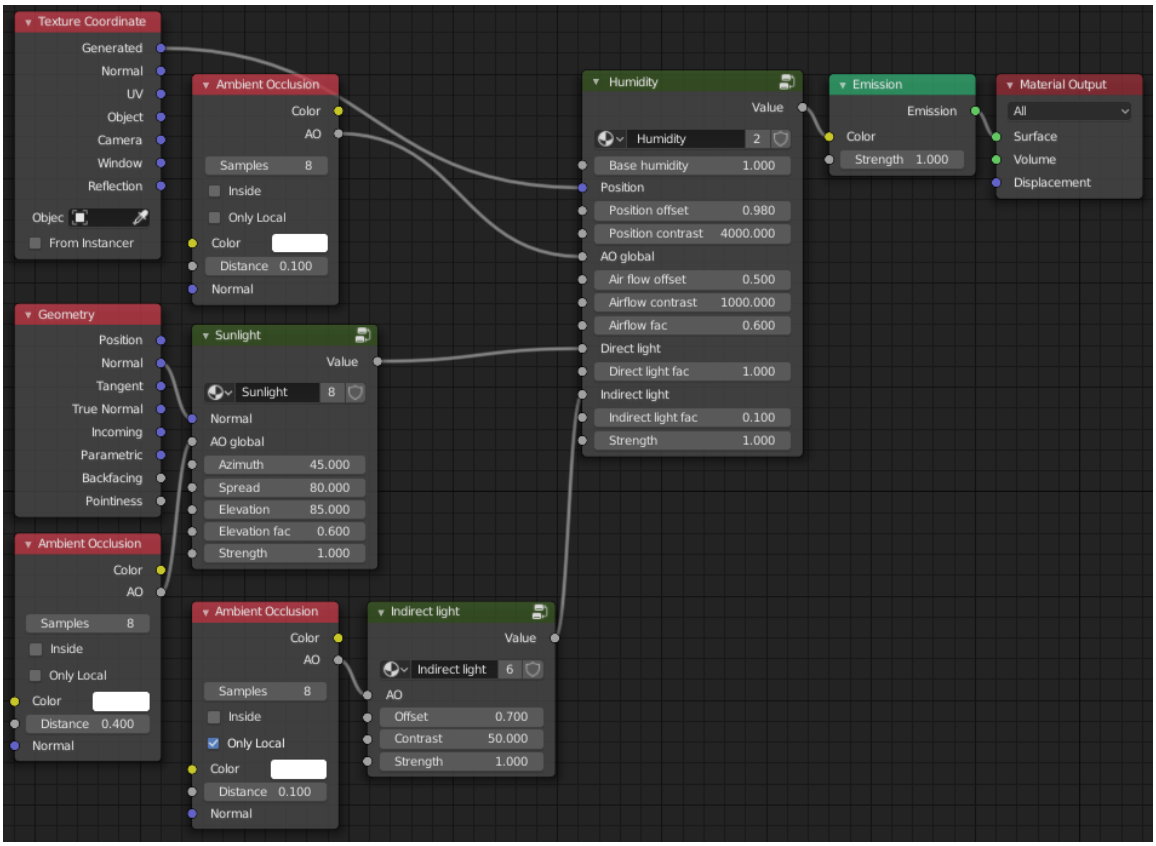
From the render result shown in 13b, it can be seen that the node group behaves as expected: the lowest portion of the ring has high humidity as if resting on ground. The top of the sphere has low humidity due to exposure to sunlight. Small notches of the Blender logo have relatively high humidity because the sunlight cannot reach them and the airflow is low.

5.2 Effect Generators

We can now inspect the visual appearance of the effect generators. We will move on from grayscale images and use materials that resemble real world materials to showcase applicability to some of the example use cases mentioned in section 3.3. Used base materials are: steel, stone, rust, moss, paper, and red paint. Variable names are again renamed into more reader-friendly form: t is Time, d is Durability, e_e is Env encourage, e_d is Env discourage, and s is Strength.

5.2.1 Spotty

Spotty node group shown in figure 14a implements equation 3.12. Spotty is used for mixing two shaders, Base Steel and Base Rust, together. Inputs Scale, Vector, Random, and Random scale are used to modify the texture coordinates passed to the Musgrave texture.



(a)

(b)

Figure 13: The usage of the Humidity node group and necessary inputs are shown in figure (a) and resulting rendered image in figure (b).

Random input shifts the vector mapping resulting in different appearance even if the object is duplicated. It is left unconnected in the figure as we only have one object. Random scale is a multiplier for the Random input, which is used to increase the shift in case the duplicate objects look too similar. In other words, scaling the Random input places the object further away of each other in the texture projection space.

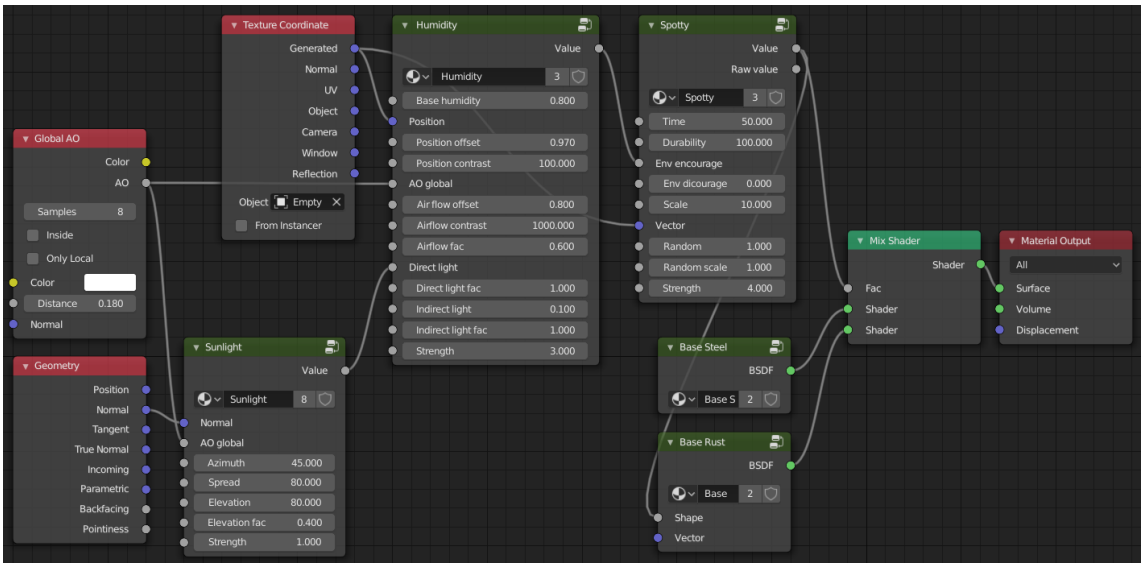
On a side note, one might notice that the edges of the rust spots in figures 14g through 14k are light reddish brown, whereas the center is of darker brown color. This behavior is similar to that of the rusted gate pole in figure 2b in section 2.2.1. This is one of benefits of using procedural textures: it is possible to affect the texture itself–not just how the two textures are mixed together.

5.2.2 Roughy

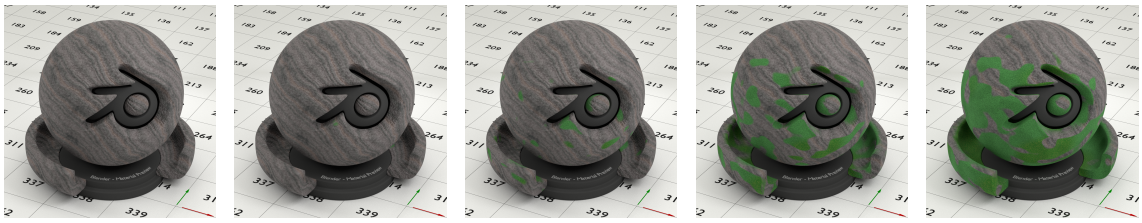
Roughy node group shown in figure 15a implements equation 3.13. Figures 15 and 16 offer two examples of how the Roughy generator can be used. In figure 15, Roughy is used to depict a stone object polished by water-carried particles over the course of hundreds of years. It utilizes two instances of the Roughy generator: one to decrease microfacet roughness (the lower Roughness generator in figure 15a), and the other to lower the height of a bump map. The bump map itself is hidden in the Base Stone material node group with the actual shader node and logic used to generate it.

Both Roughy generators use the same AO node multiplied with Perlin noise as an encouraging environmental factor. In this case, the AO node represents how easily the flow of seawater can access the surface. It can be noticed that the area in the large curve of the Blender logo is hardly smoother at all between figures 15c and 15d even though the main body of the sphere has smoothed out noticeably.

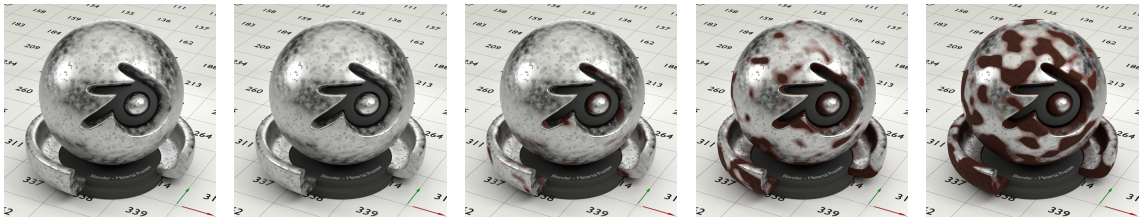
In the other example in figure 16, the object has been painted with glossy paint. Roughy increases the microfacet roughness of the material resulting in a more matte look. Encouraging environmental factor in this example is Sunlight set to shine from azimuth of 45° and 80° elevation. It is easy to see that the sunlight increases the roughness most on the top of the sphere while shadowed lower position of the object remains relatively glossy.



(a)

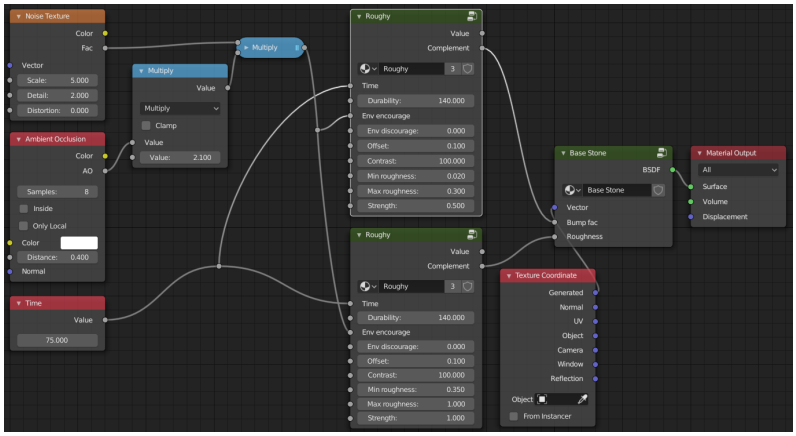


(b) $t = 0$ (c) $t = 25$ (d) $t = 50$ (e) $t = 75$ (f) $t = 100$

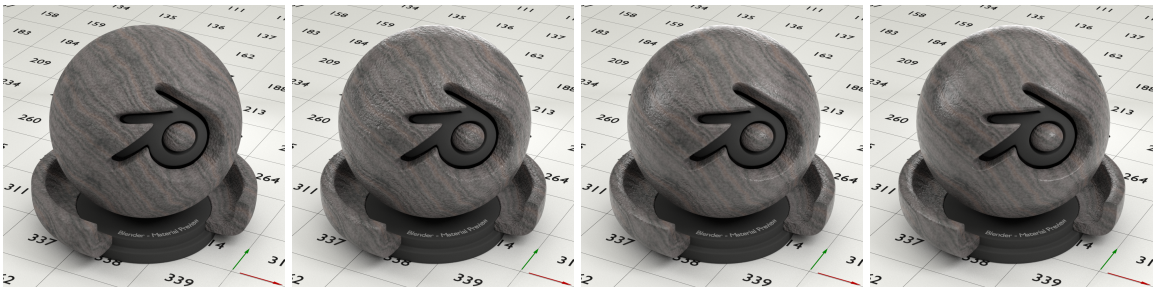


(g) $t = 0$ (h) $t = 25$ (i) $t = 50$ (j) $t = 75$ (k) $t = 100$

Figure 14: Figure (a) shows the node setup used to produce rusty steel in figure (i). In this setup, output value of Spotty is used to affect the appearance of rust. It can be seen as lighter edges of the rust spots. Figures from (b) through (f) show how moss created with Spotty grows over time. Figures from (g) through (k) show rusting steel. Note (the top of the sphere) how rust is not as sensitive to sunlight as moss.



(a)

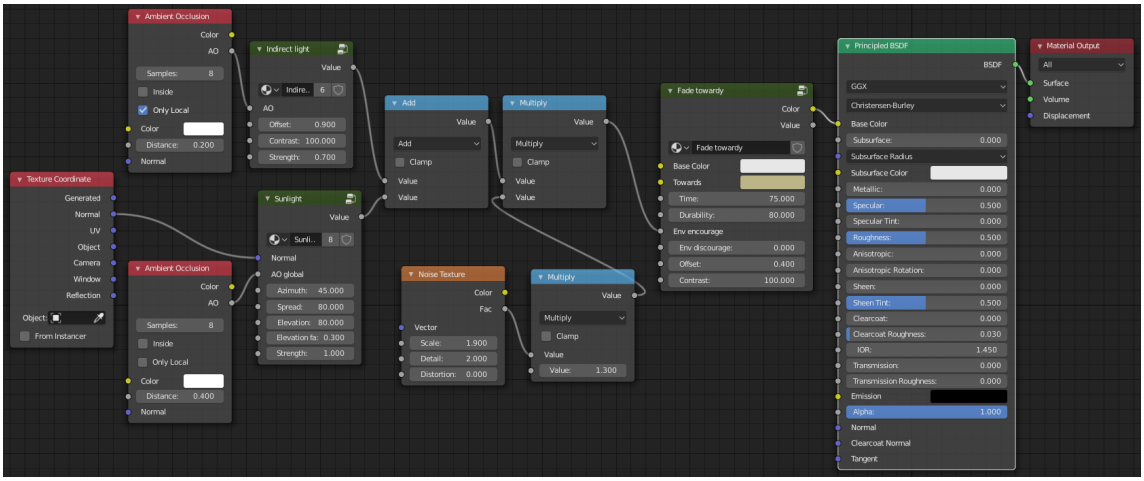


(b) (c) (d) (e)

Figure 15: A rough stone material polished by Roughy as if the object had been in beach water for hundreds of years. Node setup for figure (e) is shown in figure (a). Notice how the glossiness of the surface increases in figure (c) before the grain of the stone starts to smooth out in (d). This is achieved by using two instances of the Roughy generator: one to decrease microfacet roughness and the other to lower the bump map's height field.

(a) (b) (c) (d)

Figure 16: Glossy red paint affected by Roughy. Encouraging environmental factor used is sunlight, which makes the top of the sphere loose glossiness much faster than shadowed bottom parts. Node setup is not shown.



(a)

(b) $t = 0$ (c) $t = 25$ (d) $t = 50$ (e) $t = 75$ (f) $t = 100$

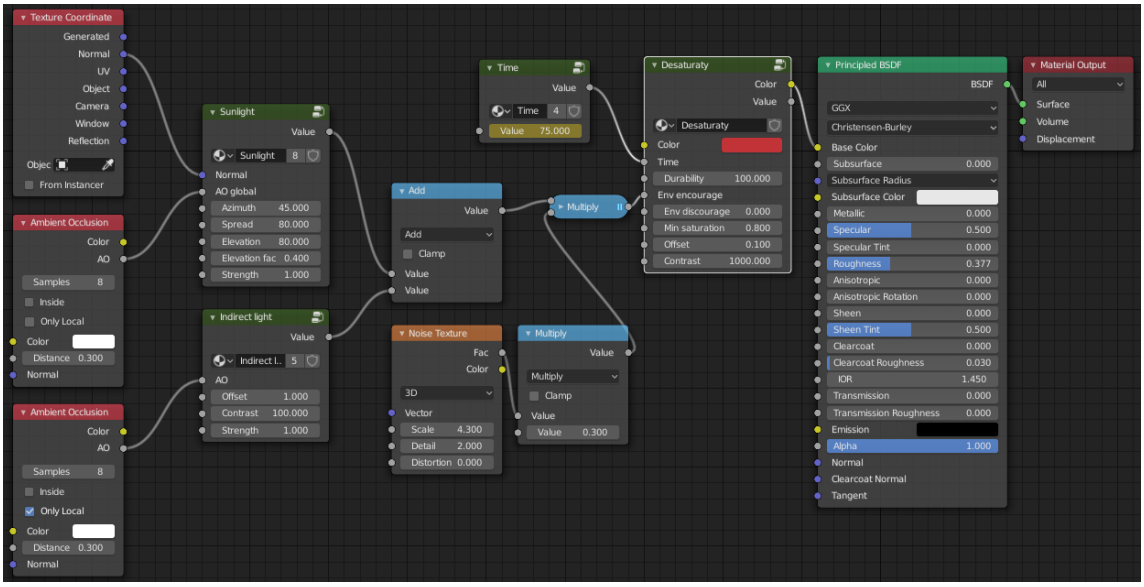
Figure 17: Usage of Fade Towardy generator and resulting rendered images when time increases. A Noise texture is used to disturb the Env encourage input to achieve an uneven result.

5.2.3 Fade Towardy

Fade Towardy is an implementation of equation 3.15. Its usage is is very straight-forward and illustrated in figure 17a. Used environmental encourage factors are Indirect Light and Sunlight disturbed by a Noise Texture node. Base color is set to pure white and the end color is light yellow. Rendering result is shown in figure 17e. Used shader is just a simple Principled BSDF. Note that the base color can be acquired from an image texture, which allows to yellow a printed paper for example. This kind of usage is demonstrated in section 6.2.

5.2.4 Desaturaty

The Desaturaty generator implements equation 3.16. Figure 18 illustrates its usage on the Material Preview object. It has same input parameters that were used on Fade Towardy



(a)

(b) $t = 0$ (c) $t = 25$ (d) $t = 50$ (e) $t = 75$ (f) $t = 100$

Figure 18: Usage of Desaturaty generator and resulting rendered images when time increases.

generator. Minimum value of saturation is limited to 0.8.

6 Results

In order to show that the model is applicable to real scenes–not just the material preview object–we use two iconic Blender benchmark scenes: the *Car Demo* scene and the *Class room* scene [1]. These scenes are often used to benchmark Blender's render performance on graphics cards and processors.

Object geometry in the scenes was not modified even if it would have been beneficial in some cases. Especially surface curvature requires certain amount of faces even on flat surfaces to recognize corners.

All images were rendered with Nvidia GTX 1070 graphics card. Main views of both scenes were rendered 1920 pixels wide and 1080 pixels high. Increase in render time was moderate: weathered images took roughly three times as much time as the images with original materials. The exact render times can be found from table 2.

6.1 Car Scene

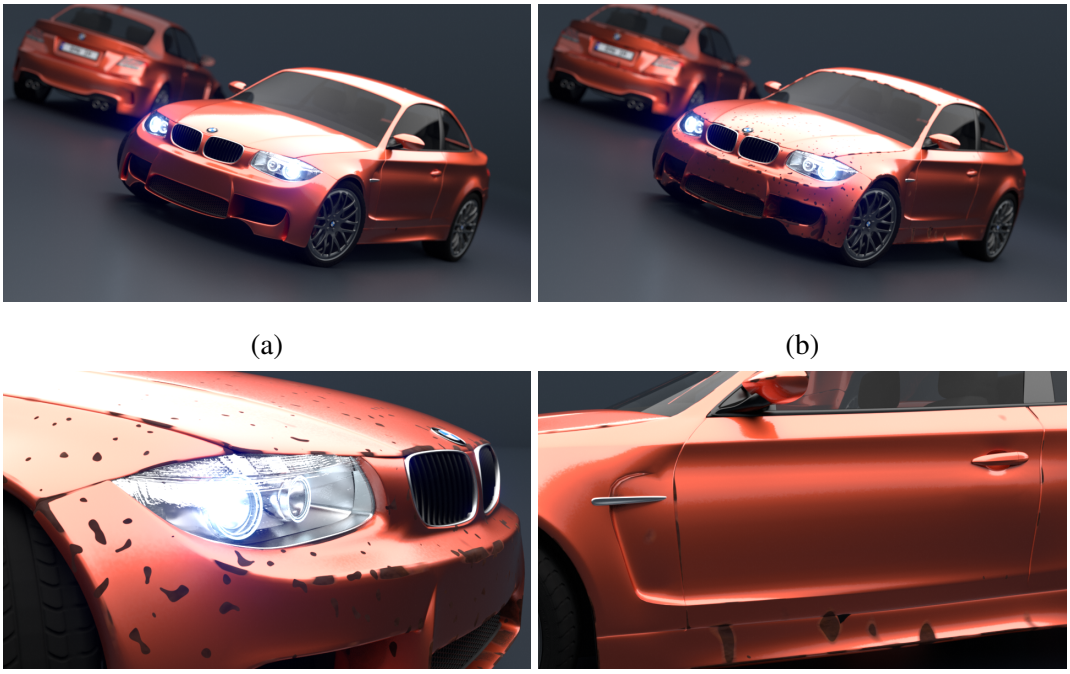
In the car scene, only one material was modified: the orange painted body of the car. Figure 19 shows the scene rendered with original and weathered materials.

Desaturaty was used to depict sunlight induced fading of paint. Its effect is subtle and fairly difficult to see without side by side comparison. Roughy was set to emphasize the front bumper and the hood of the car. Its effect is clearly visible from duller reflections on the

1. The *Car Demo* is created by Mike Pan and the *Class room* by Christophe Seux. Both are available on https://www.blender.org/download/demo-files/ under CC0 licence (https://creativecommons.org/publicdomain/zero/1.0/).

Table 2: Rendering times of the test scenes.

Scene	Original (s)	Weathered (s)
Car	115	284
Classroom	156	473



(a) (b)

(c) (d)

Figure 19: Car Demo scene rendered (a) with original materials and (b) with weathered materials. Figure (c) shows a closeup of the right headlight and (d) of the driver door.

hood.

Rust spots were created by using two separate Spotty generators. The first one is responsible for large spots on sharp edges and lower parts of the body, which can be seen in the closeup in figure 19d. Its encouraging parameters are humidity and surface curvature. The second one creates smaller spots to the front of the car as if repeatedly hit by small rocks while driving. Impact marks are clearly visible in figure 19c. The effect of impacts is not entirely restricted to the front of the car: a few very small impact marks can be seen on the driver's door.

Impact of surface curvature is clear from how much the edge of the hood has been rusted. It looks realistic when used as in figure 19c but if the scene is made much older, some problems arise: the rust does spread along the edge as it should, but it does not spread to flat areas next to it. This is because the values that depict high curvature are always very near to 0.5, and they are commonly scaled larger using a color ramp. Color ramp lacks fine control over its values, so better results might be achieved by constructing an adjustable function for scaling.

6.2 Classroom Scene

The classroom scene is a lot more complex than the car scene. We replaced more than twenty original materials on dozens of objects with weathered materials. Virtually every surface was modified excluding the smallest props such as the books on the teacher's desk. Regardless of the complexity of the scene, it could be rendered almost as fast as the weathered car scene. This is probably due to Blender being able to reuse same nodes and node groups over different materials even when set up differently. Figure 20 shows the scene rendered with original and weathered materials. There are quite a lot of details that are difficult to see in those main views, so some closeups are shown in figure 21.

Materials used in the original scene are mostly image-based. One clear disadvantage of using image-based materials can be seen in the rightmost column of chairs in figure 20a: every chair back has the same dark brown mark near its left side edge. If compared to corresponding weathered image, it can be seen that applying our procedural model to the chairs provides a varying pattern on every individual chair regardless of them being copies of a single chair object. This shows that our model is well-suited for duplicated objects and it supports a mixed approach where image-based methods and procedural methods are used side by side.

The car scene did not offer a good opportunity to showcase our Fade Towardy generator. In the classroom scene it is used to yellow the papers that hang on the wall. It is applied to the large map too, but as it is quite yellow to begin with and hangs in a shady notch, the effect is hard to notice. The papers on the left side wall have yellowed more than the ones near the window, when it should be the other way around. This brings out the problems with our simple sunlight model. It does not perform well in indoor scenes where accessibility is not good enough to approximate obstructing geometry.

Moss on the chairs concentrates near the metal frame as it considered more humid (see closeup in figure 21a). Moss on the wall concentrates on the crevices of the paneling, which is achieved by extracting the crevice information from an existing normal map image used in the original material (actual geometry of the wall is just a flat face). Even if the sunlight is not performing very well for the papers, it can be seen from the corner of the room under

(a)

(b)

Figure 20: Classroom scene rendered with original and weathered materials.

(a) (b)

Figure 21: Closeups of the classroom scene. Backs of chairs in figure (a) tend to grow moss more prominently near the metal frame. Figure (b) shows how paint is dropping off in large chunks revealing the plaster underneath.

the map, that it is correctly considered shady resulting in lush moss growth.

7 Conclusions

In this study, we have shown that a visual weathering model can be based on physical phenomena and implemented with commonly available software without requiring programming skills. Our model does not place any restrictions to object shape or size, and it can depict a wide variety of weathering phenomena with small number of parameters. Decoupling parameters and effect generators into generic interfaces makes it possible to easily build and test out new ideas. Computational cost of our model is reasonable.

As the model merely depicts where an effect should occur and with what strength, both procedural and image-based materials can be used. Procedural method allows for using the strength information to control the material when desired. By using procedural approach, the resulting weathering patterns are unique even for duplicated objects.

Adjusting the parameter values to achieve some specific result may be quite tedious, but once it is done, the age of the scene can be altered just by adjusting the time value. Also, a single material can be applied to similar, yet different, objects which further reduces the manual work required.

For future research, it would be interesting to convert these ideas onto other platforms such as game engines, where, once the parameters have been set, new objects with weathered textures could be generated into a scene. Parameters and effect generators in this study have been limited to only a few ideas: there are plenty of phenomena which were not implemented. Of these, some are fairly trivial, e.g., dust accumulation, which could be implemented based on surface orientation, and some are quite difficult–at least using the default node set of Cycles–like generation of realistic scratches. For biological phenomena, it would be interesting to use cell noise to create a situation where several colonies of species would compete for space. For creating more 3D-shaped phenomena instead of a thin film, such as thick blobs of moss, real displacement could be investigated. Utilizing surface curvature should be further researched to achieve better control over its spread over adjacent flat areas.

Bibliography

Anttila, S., editor. 2002. *Suomen sammalet : levinneisyys, ekologia, uhanalaisuus.* 2. korj. p. Suomen ympäristö. Luonto ja luonnonvarat. Helsinki: Suomen ympäristökeskus.

Bellini, R., Kleiman, Y., and Cohen-Or, D. 2016. "Time-varying weathering in texture space". *ACM Transactions on Graphics (TOG)* 35 (4): 141.

"Blender 2.81 Reference Manual". 2019. `https://docs.blender.org/manual/en/latest/index.html`.

"Blender Foundation". 2019. `https://www.blender.org/foundation/`.

Blinn, J.F. 1977. "Models of Light Reflection for Computer Synthesized Pictures". *SIGGRAPH Comput. Graph.* (New York, NY, USA) 11, number 2 (): 192–198. ISSN: 0097-8930. doi:`10.1145/965141.563893`. `http://doi.acm.org/10.1145/965141.563893`.

Bosch, C., Pueyo, X., Mérillou, S., and Ghazanfarpour, D. 2004. "A Physically-Based Model for Rendering Realistic Scratches". *Computer Graphics Forum* 23, number 3 (): 361–370. ISSN: 1467-8659. doi:`10.1111/j.1467-8659.2004.00767.x`. `https://doi.org/10.1111/j.1467-8659.2004.00767.x`.

Chen, Y., Xia, L., Wong, T.-T., Tong, X., Bao, H., Guo, B., and Shum, H.-Y. 2005. "Visual Simulation of Weathering by γ-ton Tracing". *ACM Trans. Graph.* (New York, NY, USA) 24, number 3 (): 1127–1133. ISSN: 0730-0301. doi:`10.1145/1073204.1073321`. `http://doi.acm.org/10.1145/1073204.1073321`.

Clément, O., Benoit, J., and Paquette, E. 2007. "Efficient editing of aged object textures". In *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa,* 151–158. ACM.

Codaro, E.N., Nakazato, R.Z., Horovistiz, A.L., Ribeiro, L.M.F., Ribeiro, R.B., and Hein, L.R.O. 2002. “An image processing method for morphology characterization and pitting corrosion evaluation”. *Materials Science and Engineering: A* 334 (1): 298–306. ISSN: 0921-5093. doi:https://doi.org/10.1016/S0921-5093(01)01892-5. http://www.sciencedirect.com/science/article/pii/S0921509301018925.

Cole, I.S., Muster, T.H., Lau, D., and Ganther, W.D. 2004. “Some recent trends in corrosion science and their application to conservation”. *National Museum of Australia Canberra ACT* 1:15.

Desbenoit, B., Galin, E., and Akkouche, S. 2004. “Simulating and modeling lichen growth”. *Computer Graphics Forum* 23 (3): 341–350. doi:10.1111/j.1467-8659.2004.00765.x. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2004.00765.x.

Dorsey, J., Pedersen, H.K., and Hanrahan, P. 2006. “Flow and Changes in Appearance”. In *ACM SIGGRAPH 2006 Courses*. SIGGRAPH ’06. Boston, Massachusetts: ACM. ISBN: 1-59593-364-6. doi:10.1145/1185657.1185723. http://doi.acm.org/10.1145/1185657.1185723.

Frankel, G.S. 1998. “Pitting corrosion of metals a review of the critical factors”. *Journal of the Electrochemical Society* 145 (6): 2186–2198.

Glassner, A.S. 1994. *Principles of Digital Image Synthesis*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1558602763.

Gobron, S. and Chiba, N. 2001a. “Crack pattern simulation based on 3D surface cellular automata”. *The Visual Computer* 17, number 5 (): 287–309. ISSN: 1432-2315. doi:10.1007/s003710100099. https://doi.org/10.1007/s003710100099.

———. 2001b. “Simulation of peeling using 3D-surface cellular automata”. In *Proceedings Ninth Pacific Conference on Computer Graphics and Applications. Pacific Graphics 2001,* 338–347. doi:10.1109/PCCGA.2001.962890.

Hirota, K., Tanoue, Y., and Kaneko, T. 1998. “Generation of crack patterns with a physical model”. *The visual computer* 14 (3): 126–137.

Hirota, K., Tanoue, Y., and Kaneko, T. 2000. “Simulation of three-dimensional cracks”. *The Visual Computer* 16, number 7 (): 371–378. ISSN: 1432-2315. doi:10.1007/s003710000069. https://doi.org/10.1007/s003710000069.

Hsu, S.-C. and Wong, T.-T. 1995. “Simulating dust accumulation”. *IEEE Computer Graphics and Applications* 15, number 1 (): 18–22. ISSN: 0272-1716. doi:10.1109/38.364957.

Jain, N., Kalra, P., and Kumar, S. 2014. “Simulation and Rendering of Pitting Corrosion”. In *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing,* 38:1–38:8. ICVGIP ’14. Bangalore, India: ACM. ISBN: 978-1-4503-3061-9. doi:10.1145/2683483.2683521. http://doi.acm.org/10.1145/2683483.2683521.

Jain, N., Kalra, P., Ranjan, R., and Kumar, S. 2016. “User Guided Generation of Corroded Objects”. In *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing,* 89:1–89:8. ICVGIP ’16. Guwahati, Assam, India: ACM. ISBN: 978-1-4503-4753-2. doi:10.1145/3009977.3010031. http://doi.acm.org/10.1145/3009977.3010031.

Kider, J.T., Jr. 2012. “Simulation of three-dimensional model, shape, and appearance aging by physical, chemical, biological, environmental, and weathering effects”. PhD thesis. https://search.proquest.com/docview/1170794456?accountid=11774.

Kimmel, B.W. and Baranoski, G.V. 2016. “Practical Acceleration Strategies for the Predictive Visualization of Fading Phenomena”. In *Proceedings of the 29th International Conference on Computer Animation and Social Agents,* 45–52. CASA ’16. Geneva, Switzerland: ACM. ISBN: 978-1-4503-4745-7. doi:10.1145/2915926.2915945. http://doi.acm.org/10.1145/2915926.2915945.

Kimmel, B.W., Baranoski, G.V., Chen, T.F., Yim, D., and Miranda, E. 2013. “Spectral Appearance Changes Induced by Light Exposure”. *ACM Trans. Graph.* (New York, NY, USA) 32, number 1 (): 10:1–10:13. ISSN: 0730-0301. doi:10.1145/2421636.2421646. http://doi.acm.org/10.1145/2421636.2421646.

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D.S., Lewis, J.P., Perlin, K., and Zwicker, M. 2010. “A survey of procedural noise functions”. In *Computer Graphics Forum,* 29:2579–2600. 8. Wiley Online Library.

Lu, J., Georghiades, A.S., Glaser, A., Wu, H., Wei, L.-Y., Guo, B., Dorsey, J., and Rushmeier, H. 2007. “Context-aware Textures”. *ACM Trans. Graph.* (New York, NY, USA) 26, number 1 (). ISSN: 0730-0301. doi:10.1145/1189762.1189765. http://doi.acm.org/10.1145/1189762.1189765.

Lu, J., Georghiades, A.S., Rushmeier, H., Dorsey, J., and Xu, C. 2006. “Synthesis of Material Drying History: Phenomenon Modeling, Transferring and Rendering”. In *ACM SIGGRAPH 2006 Courses.* SIGGRAPH ’06. Boston, Massachusetts: ACM. ISBN: 1-59593-364-6. doi:10.1145/1185657.1185726. http://doi.acm.org/10.1145/1185657.1185726.

Mérillou, S., Dischler, J.-M., and Ghazanfarpour, D. 2001a. “Corrosion: Simulating and Rendering”. In *Graphics Interface.*

———. 2001b. “Surface scratches: measuring, modeling and rendering”. *The Visual Computer* 17 (1): 30–45.

Miller, G. 1994. “Efficient Algorithms for Local and Global Accessibility Shading”. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques,* 319–326. SIGGRAPH ’94. New York, NY, USA: ACM. ISBN: 0-89791-667-0. doi:10.1145/192161.192244. http://doi.acm.org/10.1145/192161.192244.

Musgrave, F.K., Kolb, C.E., and Mace, R.S. 1989. “The Synthesis and Rendering of Eroded Fractal Terrains”. *SIGGRAPH Comput. Graph.* (New York, NY, USA) 23, number 3 (): 41–50. ISSN: 0097-8930. doi:10.1145/74334.74337. http://doi.acm.org/10.1145/74334.74337.

Paquette, E., Poulin, P., and Drettakis, G. 2001. “Surface Aging by Impacts”. In *Proceedings of Graphics Interface 2001.* Ottawa, Ontario, Canada. https://hal.inria.fr/inria-00510052.

Paquette, E., Poulin, P., and Drettakis, G. 2002. “The Simulation of Paint Cracking and Peeling”. In *Proceedings of Graphics Interface,* edited by W. Stuerzlinger and M. McCool, 10. Calgary, Canada: Canadian Human-Computer Communications Society. https://hal.inria.fr/inria-00606725.

Perlin, K. 1985. “An Image Synthesizer”. *SIGGRAPH Comput. Graph.* (New York, NY, USA) 19, number 3 (): 287–296. ISSN: 0097-8930. doi:10.1145/325165.325247. http://doi.acm.org/10.1145/325165.325247.

Phong, B.T. 1975. “Illumination for Computer Generated Pictures”. *Commun. ACM* (New York, NY, USA) 18, number 6 (): 311–317. ISSN: 0001-0782. doi:10.1145/360825.360839. http://doi.acm.org/10.1145/360825.360839.

“Shader Nodes”. 2019. https://docs.blender.org/manual/en/latest/render/shader_nodes/.

Steinmetz, F. and Hofmann, G. 2018. *The Cycles Encyclopedia.* 1.5.

Appendices

A Implementation Details of Node Groups

Implementation details of node groups used in this study are presented as screen shots of Blender's node editor. *Sigmoid* and *Spherical to cartesian* node groups are not shown as their implementation is trivial using math nodes.

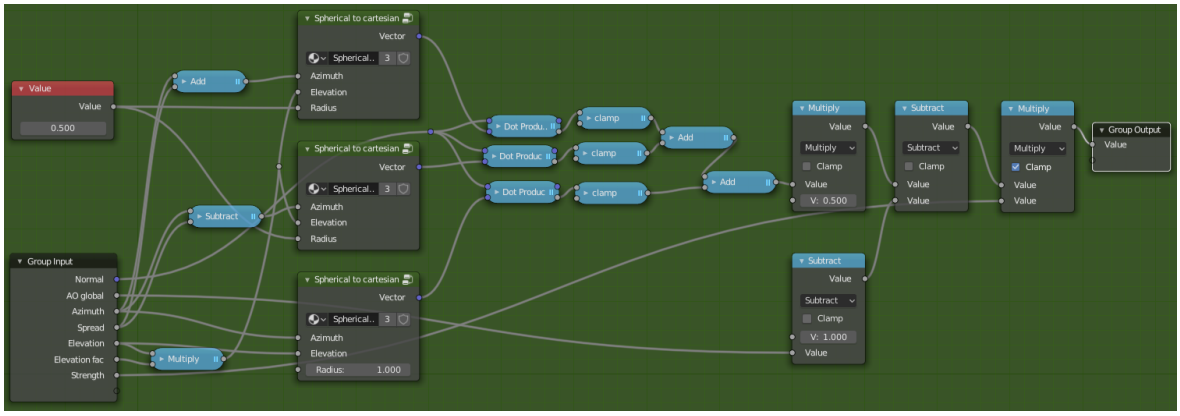


Figure 22: Sunlight node group implementation. Spherical coordinates are converted into cartesian coordinate system, so that they can be used by vector math nodes. Clamp operation is just a clamped multiplication by 1.

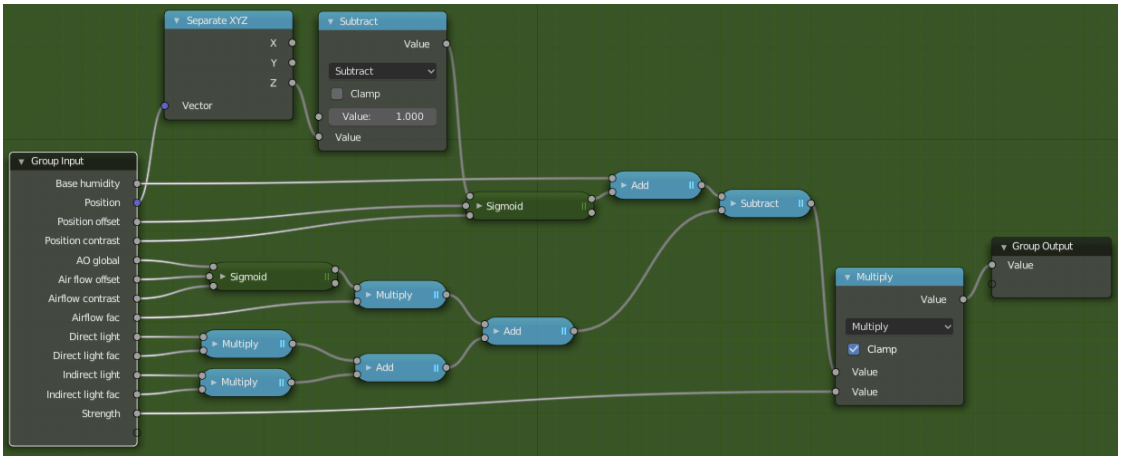


Figure 23: Humidity node group implementation. Position input takes the location of a shading point and separates its z-component. Position grows from zero to one when traversing upwards, so we subtract it from one to invert the values. Humidity could easily be expanded to point into any direction for more versatile use by using vector projection into given direction. Direct light and Indirect light inputs are controlled by a simple multiplication in the interval $[0, 1]$ and added together.

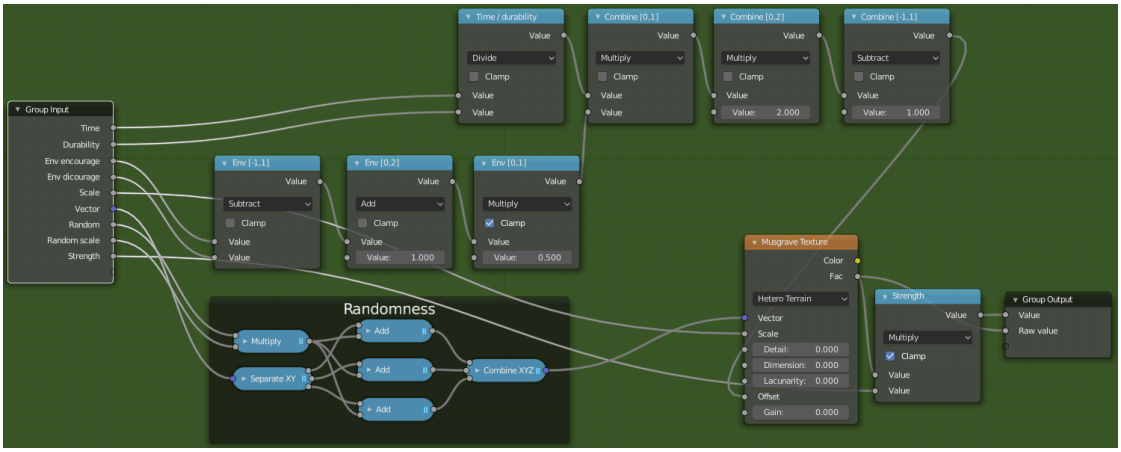


Figure 24: Spotty Generator implementation. Differences between object instances (duplicates) are created by separating the incoming Vector input and multiplying every coordinate with a random value. The final range of the Offset value (which controls the sea level) passed to the Musgrave node is defined by testing and not applicable to a general case. The range of the offset should be determined after all other Musgrave parameters are its range is known.

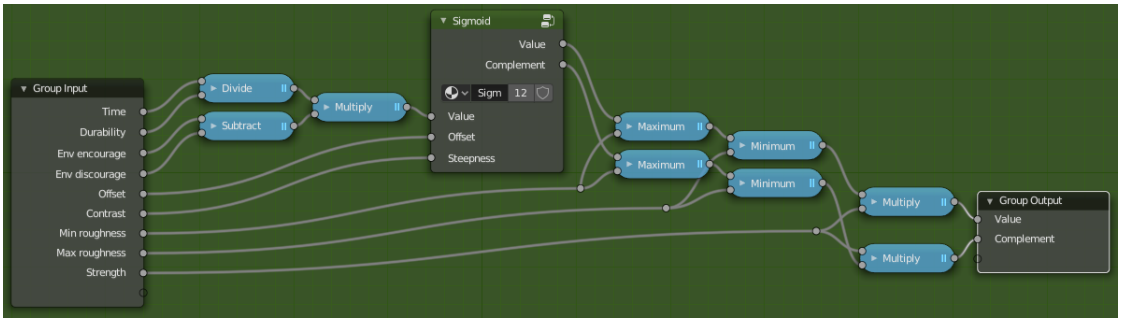


Figure 25: Roughy generator implementation. Maximum and minimum limit the value of sigmoid.

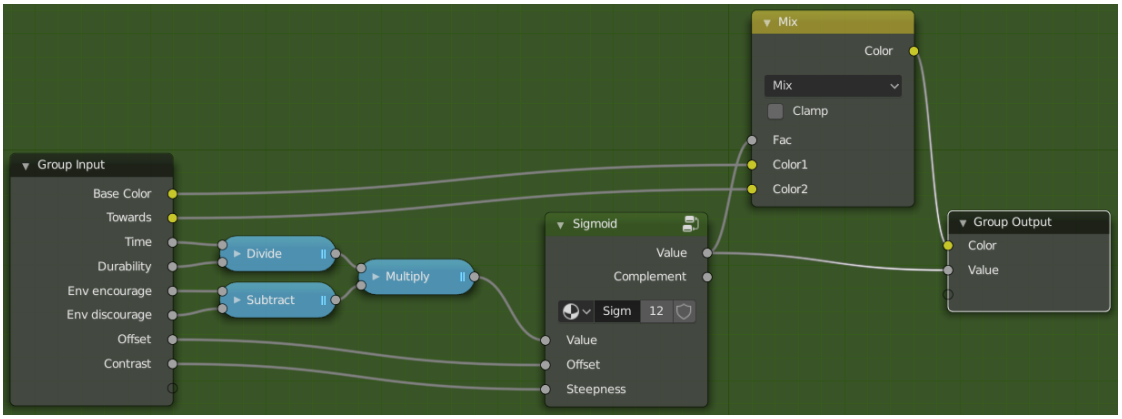


Figure 26: Fade Towardy generator implementation.

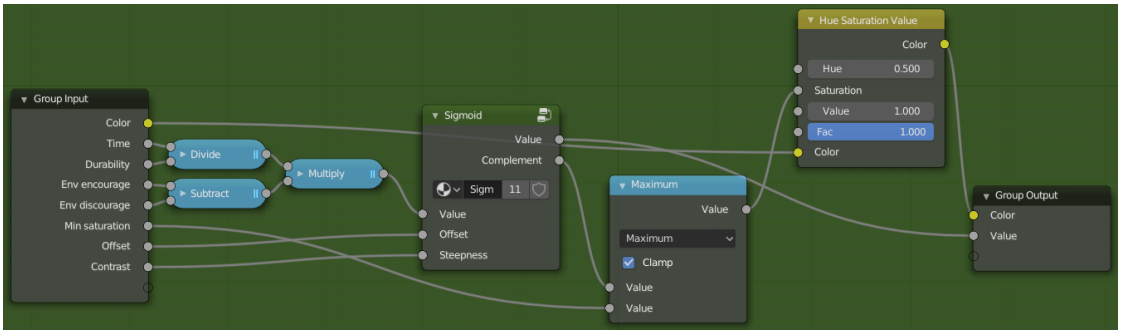


Figure 27: Desaturaty generator implementation. Maximum math node prevents the value of sigmoid to exceed *Min saturation*. Output Value is exceptionally the complement of the used value, to have a consistent look with other generators for debugging. In other words, it will show as white where the effect is strong.