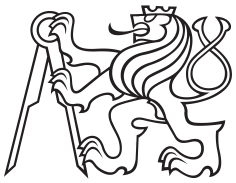
Bachelor's Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Reshaping the surface of snowy landscape by wind erosion

Emese Szabó

Supervisor: Ing. Jaroslav Sloup
Field of study: Open Informatics
Subfield: Computer Games and Graphics
May 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Szabó** Jméno: **Emese** Osobní číslo: **469879**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Přetváření povrchu zasněžené krajiny vlivem větrné eroze

Název bakalářské práce anglicky:

Reshaping the surface of the snowy landscape by wind erosion

Pokyny pro vypracování:

Prostudujte fyzikální princip přetváření povrchu zasněžené krajiny vlivem větrné eroze a následným transportem a sedimentací sněhu. Nastudujte metody simulace takto vznikajících morfologických tvarů používané v počítačové grafice. Na základě nastudované literatury navrhněte a implementujte aplikaci, která umožní simulovat vybrané typy morfologií zasněženého povrchu (příčné duny, barchany, čeřiny, sastrugi). Implementaci založte na metodě [1, 2], kterou rozšíříte o časově proměnné parametry sněhu (např. přilnavost) tak, jak jsou popsané v článku [3].
Pokuste se pomocí implementované metody vygenerovat co nejvíce tvarů sněhového podloží a vizuálně zhodnoťte jejich věrohodnost porovnáním se skutečnými fotografiemi těchto útvarů.

Seznam doporučené literatury:

[1] N. Wang, B.G. Hu: Real-time simulation of aeolian sand movement and sand ripple evolution: a method based on the physics of blown sand. Journal of Computer Science and Technology, 27(1), 2012, pp.135-146.
[2] N. Wang, B.G. Hu: November. Aeolian sand movement and interacting with vegetation: A GPU based simulation and visualization method. In 2009 Plant Growth Modeling And Applications (pp. 401-408). 2009, IEEE.
[3] S. Filhol, M. Sturm: Snow bedforms: A review, new data, and a formation model. Journal of Geophysical Research: Earth Surface, 120(9), 2015, pp.1645-1669.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jaroslav Sloup, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jaroslav Sloup podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studentky

Acknowledgements

I would like to express my gratitude to my supervisor Jaroslav Sloup for his advice, corrections and helpful comments. I would also like to thank my family, friends and my dear boyfriend, who are all very unlikely to ever read this thesis, but I am still grateful for the support, kind words, and not bothering and letting me do what I had to.

Declaration

I declare that I have created the presented thesis independently and that I have quoted all used sources of information in accordance with the Methodical instructions about ethical principles for writing academic theses.

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Prague, May 23, 2019

Abstract

Simulation of snowy landscape transformations by wind erosion is not a common topic of computer graphics. While sand movement simulation, which is very similar and also challenging, is a widely explored and studied subject, as the great number of published scientific papers proves.

This work proposes a slightly simplified implementation of Wang and Hu's physically based sand simulation method. Then, the idea of transforming the proposed model into snow simulation is introduced, including the necessary changes and new parameters that derive from the comparison of sand and snow in real environments.

The method is extended with a hard, unalterable surface under the sand, which might be crucial for the formation of some sand and snow patterns. In the conclusion, the achieved results are presented and compared with the reference method.

Keywords: particle system, snow, sand, wind erosion

Supervisor: Ing. Jaroslav Sloup
Katedra počítačové grafiky a interakce,
Fakulta elektrotechnická
ČVUT v Praze

Abstrakt

Simulace přetváření zasněžené krajiny vlivem větrné eroze není úplně běžné téma v oblasti počítačové grafiky, zatímco simulace přenosu písku, která je svou podstatou velmi podobná a stejně náročná, je široce zkoumanou a studovanou oblastí, o čemž svědčí i velké množství publikovaných vědeckých studií.

Tato práce se zabývá návrhem a implementací mírně zjednodušené verze fyzikálně založené metody Wanga a Hua pro simulaci písku. Následně je představena myšlenka transformace uvedeného modelu pro simulaci sněhu, včetně potřebných změn a zavedení nových parametrů vycházejících z porovnání chovaní písku a sněhu v reálných podmínkách.

Metoda je rozšířena o tvrdé, nezměnitelné podloží pod pískem, které může být podstatnou součástí formace některých písečných i sněhových útvarů. V závěru práce jsou uvedeny dosažené výsledky a porovnány s referenční metodou.

Klíčová slova: částicový systém, sníh, písek, větrná eroze

Překlad názvu: Přetváření povrchu zasněžené krajiny vlivem větrné eroze

Contents

Figures

Chapter 1

Introduction

Snow bedforms are geological phenomena that exist in many shapes, forms, and sizes. They exhibit various features that are also recognizable in deserts or even river bottoms. The most common patterns are dunes and ripple marks (see Figure 4.2). As Filhol and Sturm [1] point out, there is little scientific interest in these features. Only a few studies have been published, yet they are rarely cited compared to the studies of the equivalent sand features. They suggest that the reason for such sparse interest may be the low population in the areas where snow bedforms occur, or rather that the snow features have a much shorter lifespan than the sand features.

Figure 1.1: Example of sastrugi, an erosional bedform, with author (Clea Bertholet) for scale, photo looking downwind. Photo by Kelly Kochanski [2].

In the field of computer graphics, little to no research can be found on snow in connection to wind erosion. There are efforts to model fallen snow on various surfaces, such as Fearing's [3] combination of an accumulation and stability model. This essentially simulates a snow cover, with no real focus

on wind transportation, yet not entirely omitting it either.

The generation of sand covered landscape though is a widely studied topic of computer graphics. Researchers have developed various methods of sand (or dust) simulation, some of which are only appearance-based, but most of them are based on physical parameters. Some methods make use of a height field [4, 5] or a height map [5], others include a particle system [5, 6] to simulate the granular properties of sand. Another interesting option is simulating sand as a fluid [7] rather than independent particles, which approximates very fine grain sand.

Wang and Hu [8, 9] focus on the wind erosion on sand surface. They take the physical approach and discuss the physics of blown sand. Their work involves research topics from sand grain and sand bed modeling through rendering complete desert scenes.

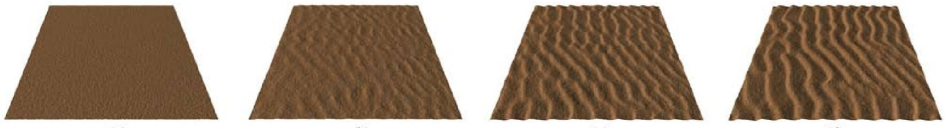


Figure 1.2: Evolution of sand ripples. Simulation results (left to right) after 0, 100, 200 and 400 seconds. Image taken from [9].

As they mention, their research is a kind of erosion study [9]. They used a height field to represent the simulated sand bed and an aeolian erosion model (see Figure 1.2). This thesis relies almost entirely on [8, 9], to a certain extent, implementing their sand bed and erosion model as a base for later transformation into snowy landscape simulation. The final goal of this work was to simulate snow features resembling Figure 1.1 for example.

The contents of the following chapters are organized as follows. Chapter 2 discusses the different forms of sand grain movement and the wind, according to [9]. Chapter 3 describes the sand model from [8, 9], which serves as a base of the implementation. In Chapter 4, the differences and similarities between sand and snow bedforms are discussed as well as the physical aspects of particles, the process of snow compression (sintering) and the necessary new parameters for snow simulation. The proposed implementation is described and illustrated with pseudo code in Chapter 5. Then, the achieved results and conclusions are presented and discussed in Chapters 6 and 7 respectively.

Chapter 2

Sand Movement

If wind with sufficient velocity blows over a sandy surface, some grains will get caught up in the air. Wang and Hu [9] assume that every grain has a *threshold velocity*, which, when exceeded, the grain will surely begin to move. They describe it as granular material in fluid, the fluid being the air. Therefore the movement itself is mainly affected by the density of air and the density of sand, and the shape and size of a given sand grain.

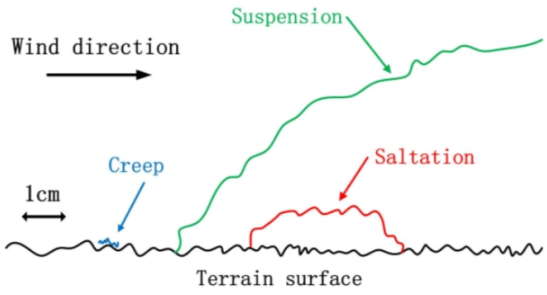


Figure 2.1: Three forms of aeolian sand grain movement. Image taken from [8].

There are three different ways of sand movement, namely *suspension*, *saltation*, and *creep* (see Figure 2.1). The first, *suspension* involves long distance movement, where the grains entrained in the air will either travel many kilometers before deposition or will not deposit at all for a significant period of time. Also, this type of transport will only affect the smallest grains, with diameters under 0.05 mm, and it has no significant effect on the resulting shapes. Therefore in a simulation as small-scaled as this, there is no point in taking suspension into account. *Creep*, on the other hand, will move bigger

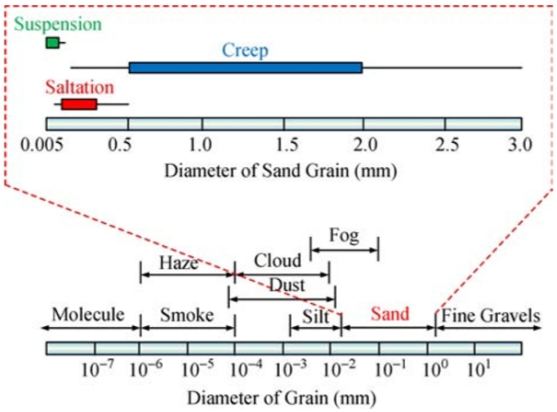


Figure 2.2: Diameter of sand grains. Image taken from [9].

grains, whose diameters range from 0.25 to 2 mm. These grains are often too heavy for the wind to be lifted, so they will only roll and slide along the surface. Wang and Hu [8] also describe sand collapses as a kind of creep caused by gravity instead of the wind.

The third form, *saltation*, is the main mode studied and simulated in [9], which affects grains with diameters mainly between 0.1 and 0.5 mm. It is kind of a jumping motion, where grains will get caught up in the air, but will only travel short distances before hitting the ground. Apparently, saltation is responsible for the transport of most of the sand in real environments. This kind of sand movement is able to create basically two different landscape features, sand ripples (or waves) and dunes. These forms sometimes appear in combination: ripples may form on the surface of sand dunes as an effect of changing wind conditions. The different qualities of these forms are further discussed in Chapter 4, where sand and snow features are compared.

Figure 2.3: Various sand ripples on the surface of dunes, with plants (left) and sand ripples resembling those simulated in [9] (right). Photos taken from [10] and [11].

2.1 Dune field morphology

The previously described grain movement also enables the formation of other patterns besides ripples, which also classify as dunes. Different wind conditions allow for different dunes to appear. Some of these features have exact analogies in snow, which are covered in Chapter 4. Here, a dune classification from [12] is adopted, where they focus on *free dunes*. These appear in almost entirely flat and vegetation-free, completely dry desert environments, which arguably simplifies the conditions for modelling them.

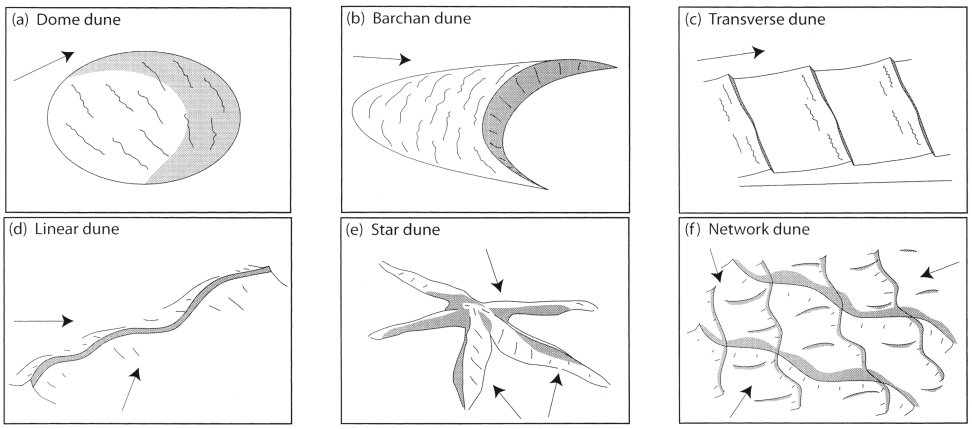


Figure 2.4: Schematic views of typical dunes: (a) dome dune, (b) barchan dune, (c) transverse dune, (d) linear dune, (e) star dune and (f) network dune. Arrows show the predominant wind directions. Figure 1 from [12].

According to Bishop *et al.*[12], free dunes can be separated into three categories, namely transverse, linear, and star dunes. Some of these can be seen in Figure 2.4. The transverse category includes barchan and transverse dunes, among others. A barchan dune has a crescentic plan-shape opening downwind [12], sometimes described as an arrowhead or a laundry iron [1]. Transverse dunes have their crests transverse to the wind and they usually move downwind. Under linear dunes, Bishop *et al.*[12] mention seif dunes and sand ridges, the former being a sinusoidal pattern with a sharp crest and the latter a partly vegetated linear dune that is larger than a seif [12] (see Figure 2.4(d)). The star dune category also includes network dunes (see Figure 2.4(e) and 2.4(f)), both are results of complex, changing wind conditions.

Figure 2.5, also adopted from [12], shows how the amount of available sand is related to the variability of wind directions. On the wind axis, the uni-directional end means that wind blows in one major direction nearly all the time. The other end, *complex* suggests wind changes between more than two distinct directions, or that there is no definite wind regime. Figure 2.5

strongly implies that the more complex the wind conditions and the more available sand there is, the more interesting and complicated the forming patterns become. According to this, the proposed implementation classifies as a mostly uni-directional wind model with the option of changing the sand availability.

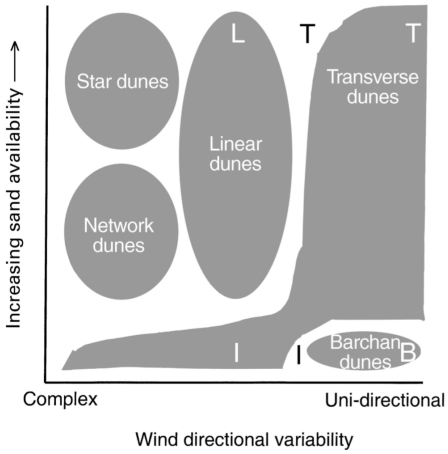


Figure 2.5: Dune type diagram with regard to sand availability and wind direction variability. B, I, L, T denote barchan, isolated, linear and transverse dunes respectively. Figure 2 from [12].

2.2 Wind

Simulating realistic wind conditions is a very complex and time-consuming task, therefore Wang and Hu[8] adopted a simplified wind field model. They assume that the wind direction does not change during the simulation. After omitting the constants they set to zero, the formula for the mean wind velocity at height z above the surface is

$$U(z) = \frac{u_*}{k} \ln\left(\frac{z}{z_0}\right) \tag{2.1}$$

where u_* is the friction velocity, k=0.4 is the dimensionless Von Karman's constant and z_0 is the aerodynamic roughness length, the height at which the wind speed reaches zero [8]. As they work with vegetation on the surface, they further calculate z_0. For a bare sand covered surface, they refer to Bagnold[13], who suggests to set $z_0 = \frac{d}{30}$, with d being the diameter of the grain.

In this implementation, it is possible to change the wind direction during the simulation as one wishes, but it does not change without interaction.

Chapter 3

Sand Model

The presented sand model is a simplified alternative of Wang and Hu's sand model in [8, 9] with a bit different approach to sand collapses and redistribution. Most of the model, such as grain initialization, transportation and collision are directly implemented as described in [8, 9]. As Wang and Hu, we assume a uniform particle diameter of 0.15 mm, and particles are generated with random starting velocities and directions.

3.1 Grain initialization

Wang and Hu [9] assumes that there is a lift off velocity $\mathbf{V}$ for each grain, when friction velocity u_* exceeds threshold friction velocity u_{*t}. The value of u_{*t} can be calculated as mentioned in [14]:

$$u_{*t} = \sqrt{0.0123(sgd + \frac{3.0 \cdot 10^{-4}}{\rho \cdot d})} \tag{3.1}$$

where s is the sediment to fluid density ratio, g is gravity ($\mathrm{m\,s^{-2}}$) and d the diameter (mm). [9] states that the initial velocity V_y is proportional to the friction velocity.

$$V_y = Bu_* \tag{3.2}$$

where B is between 0.8 and 2. The corresponding direction is 21° to nearly 90° with respect to the horizontal axis. To represent the direction of lift off velocity, spherical coordinates (ϕ, ψ) are used, where $0 \leq \phi \leq \frac{\pi}{2}$ is assigned randomly and $0 \leq \psi \leq 2\pi$ is the wind direction.

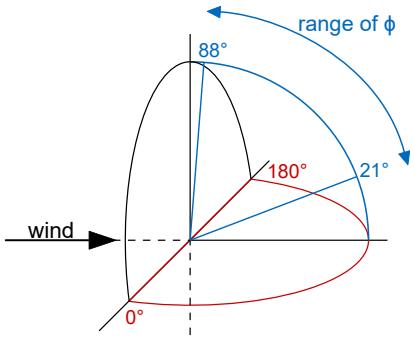


Figure 3.1: Illustrating the angle generation ranges. ϕ is the lift-off direction (blue) and ψ is given by the wind direction (red).

Finally, the lift off velocity $\mathbf{V}$ is given by

$$V_x = \|\mathbf{V}\| \cos\phi \cos\psi, \tag{3.3a}$$

$$V_y = Bu_* = \|\mathbf{V}\| \sin\phi, \tag{3.3b}$$

$$V_z = \|\mathbf{V}\| \cos\phi \sin\psi, \tag{3.3c}$$

where $\|\mathbf{V}\| = \sqrt{V_x^2 + V_y^2 + V_z^2}$ is the magnitude of $\mathbf{V}$. Sand grains are allocated uniformly on the surface at position $\mathbf{P}$

$$P_x = rand(), \tag{3.4a}$$

$$P_y = h(i, j), \tag{3.4b}$$

$$P_z = rand(), \tag{3.4c}$$

where h is the height map and *rand()* refers to the random number generator used [9].

3.2 Transportation

From the many different forces acting on grains in the air, they only consider gravity $\mathbf{F_g}$ and drag force $\mathbf{F_d}$ [8, 9].

$$\mathbf{F_g} = (0, -mg, 0), \tag{3.5}$$

$$\mathbf{F_d} = B\|\Delta\mathbf{V}\|\Delta\mathbf{V}, \tag{3.6}$$

where $\Delta\mathbf{V} = \mathbf{U} - \mathbf{V}$ is the difference between wind velocity and the velocity of flying sand. $\mathbf{F_d}$ and $\boldsymbol{\Delta}\mathbf{V}$ have the same direction. With the help of these forces, the change in velocity and position is calculated iteratively:

$$\mathbf{V_{curr}} = \mathbf{V_{prev}} + \frac{\mathbf{F_d} + \mathbf{F_g}}{m}\Delta t, \tag{3.7}$$

$$\mathbf{P_{curr}} = \mathbf{P_{prev}} + \mathbf{V_{curr}}\Delta t. \tag{3.8}$$

3.3 Deposition and Collision

When a grain reaches the ground, it can either deposit there, or collide with the other grains and push some into the air, or get caught up again [8, 9]. In our simulation, no other grains are actually being pushed into the air. The velocity of the incoming grain $\mathbf{V_1}$ is divided into a normal and tangential component [9]. Then the new, reduced velocity $\mathbf{V_2}$ is calculated as

$$\mathbf{V_{1n}} = (\mathbf{V_1} \cdot \mathbf{N}) \cdot \mathbf{N}, \tag{3.9a}$$

$$\mathbf{V_{1t}} = \mathbf{V_1} - \mathbf{V_{1n}}, \tag{3.9b}$$

$$\mathbf{V_2} = -\mathbf{V_{1n}} \cdot f_{atten} + \mathbf{V_{1t}} \cdot f_{frac}, \tag{3.9c}$$

where $\mathbf{N}$ is the terrain's normal vector, n and t mark the normal and tangential directions, and f_{atten} and f_{frac} are damping parameters to simulate momentum attenuation and friction force respectively [9]. If the outgoing velocity falls below a threshold, it is set to zero, to avoid infinite jumping.

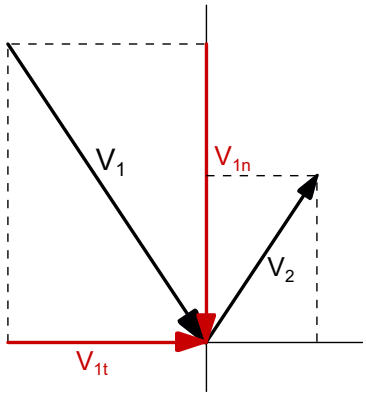


Figure 3.2: Illustrating the velocity decomposition, V_1 and V_2 with black, V_{1t} and V_{1n} with red arrows.

3.4 Collapsing

Every granular material has a specific angle of repose, that is the steepest angle of descent to which the material may be piled without sliding. For loose sand, this angle is around 34-35°, above which the built up sand pile will collapse, the excess grains sliding down the slope.

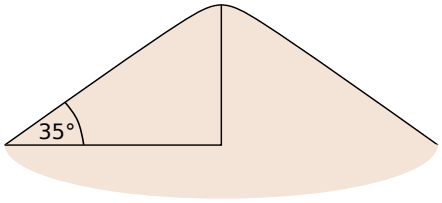


Figure 3.3: The angle of repose, 35° for loose sand. The steepest angle at which granular material stays stable, the excess material on the slope will slide down.

Wang and Hu[8] simulate this collapse by making the excess grains scatter around the deposition point. They formulate the process as

$$h_{curr}(i,j) = h_{prev}(i,j) + G\Delta h\widetilde{step}(\tan\theta, T\Delta h) \tag{3.10a}$$

$$\Delta h = \frac{2-\sqrt{2}}{4}\sum_{N_4} h_{prev}(i,j) + \frac{\sqrt{2}-1}{4}\sum_{N_8-N_4} h_{prev}(i,j) - h_{prev}(i,j) \tag{3.10b}$$

$$\widetilde{step}(a,b) = 1 \text{ when } b \geq a, \text{ and } 0 \text{ when } b < a \tag{3.10c}$$

where h always marks the height at *(i, j)* and N_4, N_8 are the neighbouring points on the grid (see Figure 3.4). T is a scale factor and G is related to gravity. They do this on GPU through selective filtering [8].

(a) : N4 **(b) :** N8

Figure 3.4: N_4 and N_8 neighbourhoods (red) of a grid point (black).

As the proposed implementation is not on a GPU, a simplified method was chosen for checking the height difference and making the sand to scatter. First, it was done by a shallow recursion with limited nesting depth, where the function checks the eight neighbouring grid points and calculates the height differences, then the current angle of the slope. If it exceeds the angle of repose (set to 30° and even 25° for testing purposes), the excess sand is distributed to the neighbour with the biggest height difference. This concept was later changed to better mirror the method in [8, 9], where collapsing is done on GPU [9]. With the recursion omitted, the `CallRedistribute` function calls `Redistribute` once for each grid vertex, effectively smoothing the terrain to a certain extent.

Chapter 4

Snow

Snow bedforms come in many different shapes and sizes, and exhibit various patterns. Some of them, like dunes and ripple marks, are also found in deserts or river bottoms. There is a surprisingly low scientific interest in snow bedforms, whereas they appear on approximately 8-11% of the Earth's surface, including the ice of Antarctica and Greenland [1]. Sturm and Filhol [1] also point out that the importance of these snow covers extends beyond the shapes, they influence the surface roughness in the Arctic and Antarctic, largely affecting the regional energy balance, ice melt rate, the exchange of air between the atmosphere and firn, etc.

4.1 Sand and snow

The key differences between sand and snow are in the spatiotemporal stability and the granular properties of the materials. Sand features almost entirely form through saltation, but snow bedforms are also largely affected by creep. Although suspension can occur, just like with sand, this has no significant effect on the formation of bedforms.

Despite the differences, ripple marks, for example, do form in both sand and snow and they look very much alike. Also, ripples may appear underwater, on beaches or river bottoms.

Snow bedforms are spatially and temporally variable shapes. There is certainly more variability to them then there is to sand features. Some dunes in deserts may be hundreds of years old, whereas their snow counterparts form within a few hours and hardly ever last longer than a few months [1]. This spatiotemporal variability creates the different key characteristics of snow bedforms, according to which they are depositional or erosional. As opposed to sand, these patterns are usually lower and flatter. For example, sand barchans can be 10 m high, 100 m wide and 150 m long, but snow barchans only range from 3 to 20 m in length, 5 to 12 m in width and are only about 0.1 to 0.5 m tall [1]. This is where the materials' characteristics come into play.

4.2 Particles

Another big difference is in the size and density of particles. Knowing these parameters allows for calculating the mass of the simulated grains, which is essential in the equations from [9].

According to Sturm and Filhol [1], the density of snow particles is 917 $\mathrm{kg\,m^{-3}}$ (the actual density of ice) and 2650 $\mathrm{kg\,m^{-3}}$ for sand (essentially the density of pure quartz). Although this assumption ignores the fact that real sand has other constituents apart from quartz, it is not inherently wrong.

For example, in [15], two sand samples were collected (for unrelated purposes) in the United Arab Emirates and their densities were measured. The samples were mainly constituted of calcite and quartz. Table 4.1, adopted from [15], shows the particle density of the samples.

Sample	1	2
Bulk density ($\mathrm{kg\,m^{-3}}$)	1493.14	1588.20
Particle density ($\mathrm{kg\,m^{-3}}$)	2612.00	2656.58
Porosity	0.43	0.40

Table 4.1: Density values of the samples. Table taken from [15].

In the same study, they also separated the samples through a series of sieves to see how many grains are of certain sizes. Figure 4.1 from [15] shows that around 60% of the sand in each sample stopped between sieves of dimensions 0.1 and 0.2 mm. This also confirms that it is safe to assume the simulated

grains have a diameter of 0.15 mm, as these conditions do occur in real sandy environments.

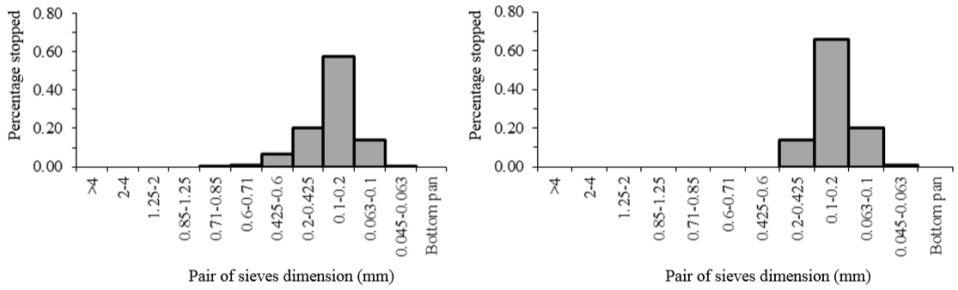


Figure 4.1: Size distributions for sample 1 (left) and sample 2 (right) from [15].

With the density and grain diameter set, we can calculate the weight of our particles. Given these values, they should weigh only about

$$2.65\,\mathrm{mg\,mm^{-3}} \cdot 0.014\,\mathrm{mm^3} = 0.0371\,\mathrm{mg} \sim 0.04\,\mathrm{mg}. \tag{4.1}$$

4.3 Sintering

Sintering is the process by which snow particles bond at temperatures below the melting point [16]. This process is completely absent in sand but plays a major role in the formation of snow bedforms. It is the reason for the huge age (and height) difference between sand and snow dunes.

First, depositional features start to form thanks to saltating snow particles in the wind. Sturm and Filhol [1] think of sintering as a timer - it starts at the moment the grain is deposited. If it remains in the same position long enough, it becomes bonded. They also state how this *long enough* is a function of many values and that this bonding rarely takes more than 24 hours. After the bond occurs, the bedforms become erosional as it becomes harder for the wind to lift individual particles into the air.

4.4 Bedforms

There are seven different snow bedforms that are recognized: transverse, barchan and whaleback dunes, pits, crag and tails, ripple marks, and sastrugi

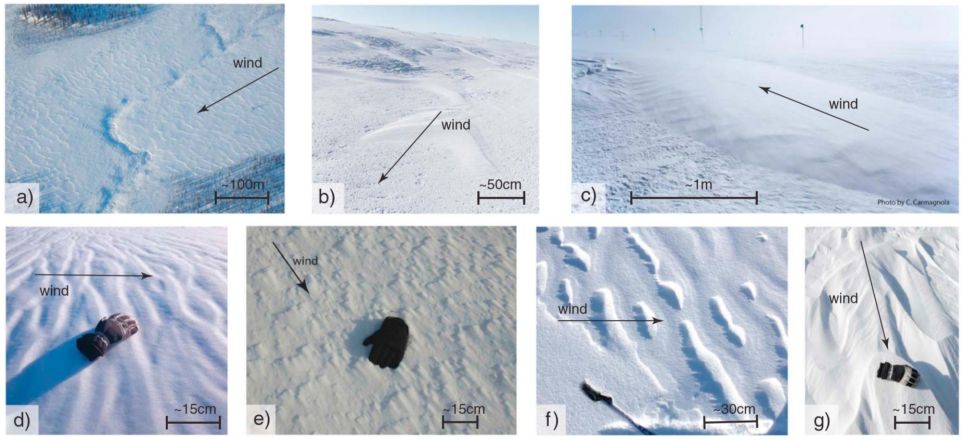


Figure 4.2: Photographs of typical bedforms encountered at the surface of the snow. The black arrow indicates the wind direction. (a) Snow waves (transverse dunes) viewed from an airplane, (b) barchan dunes surrounded by small pits, (c) a whaleback dune (photograph by C. Carmagnola), (d) ripple marks, (e) crag and tail features formed behind ice nucleus, (f) pits with the handle of a ski pole for scale, and (g) sastrugi (showing bedding or lamellae) with a glove for scale. Picture adopted from [1].

(see Figure 4.2). Each of these shapes forms in different weather conditions, and in certain situations some will never occur. Take ripple marks as an example. They are wave-like bedforms, typically small with wavelengths around 5 to 20 cm. It has been estimated that they only form at certain wind speeds (see Figure 4.3).

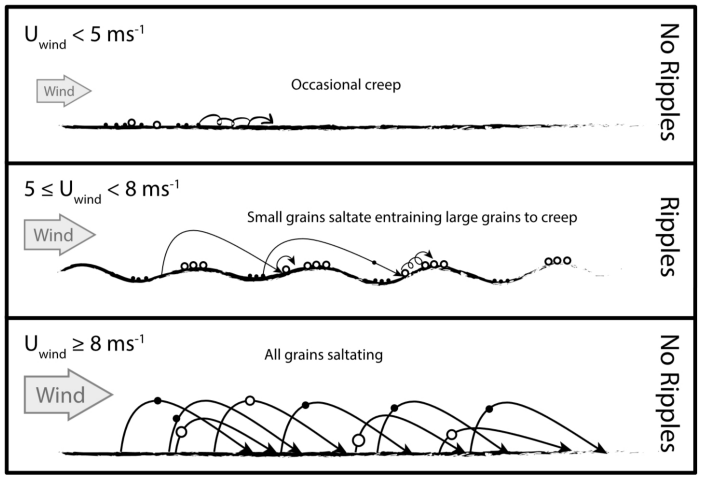


Figure 4.3: Illustrating necessary conditions for ripple marks to form. Image taken from [1].

As mentioned in Chapter 2, some of the sand patterns have analogous forms in snow, namely ripple marks and barchan and transverse dunes. Figure 2.5 shows these forms in the uni-directional end of the wind scale, the linear dunes being in the middle, where the bi-directional wind is supposed to be.

This is probably similar for the snow features of the same forms, considering Sturm and Filhol [1] mostly discuss patterns that formed in winds of one major direction.

4.5 New parameters

To modify and extend the sand implementation to simulate snow instead, new parameters have to be introduced. The overall weather has to be considered, not only the wind direction and speed. The changes of temperature play a major role here. Not only does it affect how long it takes for the snow to bond in sintering, but the snow particles themselves should be different in size and shape.

There is little to no comprehensive or universally applicable data on the perfect conditions for a certain pattern to form. And some of the existing studies have incomplete measurements or simply can not be compared (see [1]).

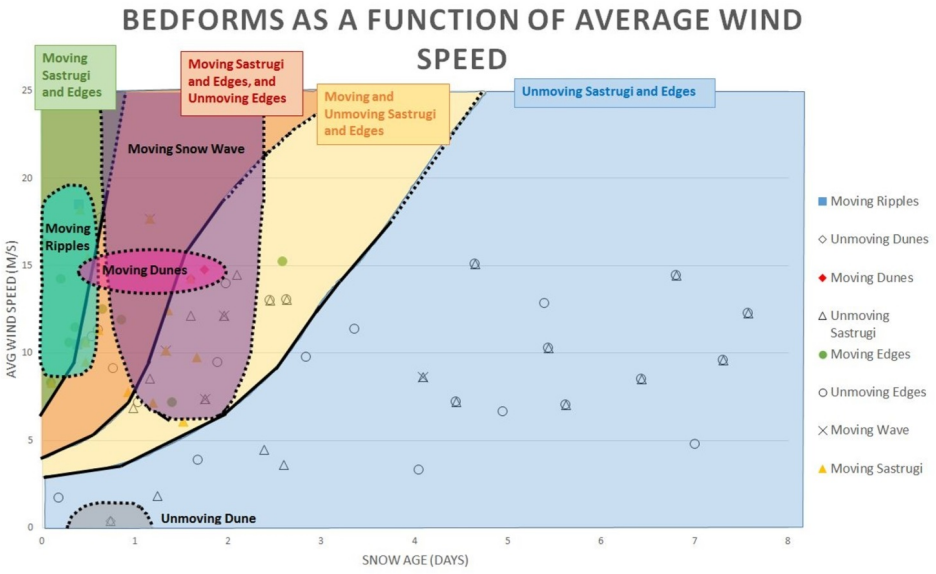


Figure 4.4: Bedforms as a function of average wind speed. A generalized interpolation of the data, in which all bedforms seen in the data are plotted, with shading to show where the points could fall. Thick lines represent separation between movement of bedforms, and dotted lines represent speculation as to where the data could fall. Figure adopted from [2].

However, there is a thesis on snow bedforms, describing and connecting them to the wind and time exactly as previously discussed. The study was

conducted in the winters of 2016 and 2017 in Niwot Ridge (Colorado, USA), an alpine ecology research site. The wind speeds there vary from 3-20 $\mathrm{m\,s^{-1}}$ with gusts around 4-25 $\mathrm{m\,s^{-1}}$ and the temperature can fall as low as $-20\,^{\circ}\mathrm{C}$. Bertholet [2] discusses the various bedforms observed and plots them as a function of wind speed and snow age. The figures are very informative as to when do which types of bedforms become stationary, and how much they move up to that point.

Figure 4.4 plots the observed snow features as a function of average wind speed in $\mathrm{m\,s^{-1}}$ and the snow age in days. Overall, the older the snow, the less the bedforms move. This is also generally true for the wind, the slower it is the less movement appears. The most frequently occurring features seem to be sastrugi and snow edges. These are terraced features with heights of a few millimeters, appearing as flat layers. These bedforms are not mentioned as a separate form by [1]. Bertholet [2] hypothesizes they are also eroded by wind.

Figure 4.5: Edges at Brainerd Lake in Colorado. Author (Clea Bertholet) for scale. Photo by Kelly Kochanski [2].

Contrary to Figure 4.3 from [1] showing ripples in 5-8 $\mathrm{m\,s^{-1}}$, Figure 4.4 shows moving ripples in wind speeds of about 8 to $18\,\mathrm{m\,s^{-1}}$. We can assume that the different temperatures in the observed locations are responsible for such a difference, as it affects particle size and the time it takes particles to bond in sintering.

Chapter 5

Implementation

This work is a simplified version of Wang and Hu's [8, 9] simulation method, using a height map and a particle system for simulating wind erosion on a bare sandy surface. The implementation described in this chapter serves as a base for expansion into the simulation of snow bedforms (see Chapter 4). The implementation is done in C++ using OpenGL 3.1. The visualization is done by displaying the height map using triangle strips in OpenGL, utilizing the code in [17].

```
int main() {
    //initialize GLUT (OpenGL UTility)
    //set glut functions like glutDisplayFunc, glutReshapeFunc, etc.
    //  to call our functions

    init();  //initializes shaders and shader attributes
             //calls generateTerrain()
             //sets up vertex arrays and buffers for drawing

    createMenu(); //creates the menu for changing simulation parameters

    glutMainLoop(); //calls the refresh function
                    //where the particle update and regeneration happens
                    //Redistribute for surface smoothing is called
                    //and init() is called to recalculate the triangle strips

    cleanup();  //deletes allocated buffers and shaders
    return 0;
}
```

Listing 5.1: Illustrating the main function of the application.

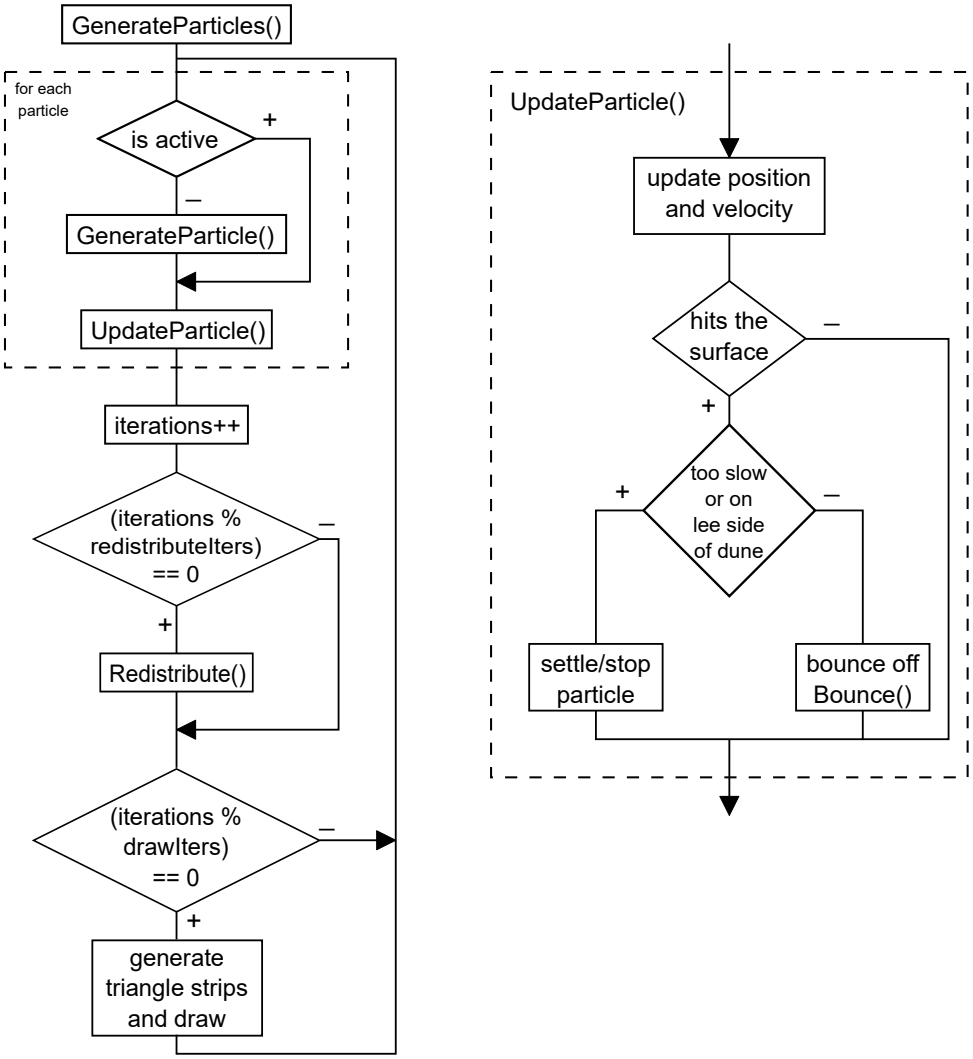


Figure 5.1: Simulation workflow.

Figure 5.1 illustrates the basic workflow of the simulation, with the main loop on the left and the `UpdateParticle` method on the right. Each function is further discussed in detail in the following sections.

After the initial particle generation is done, the program enters the main loop, which consists of regenerating inactive particles and updating the active ones, then the iteration counter decides, if there will be a call to `Redistribute` and to recalculate triangle strips before redrawing. The `UpdateParticle` function (see Listing 5.5) tests the particle for hitting the ground, then either recalculates its position and velocity, or stops and deactivates the particle. Recalculation happens in every call according to equations (3.7)-(3.8), but the flowchart shows only the exceptions: settling and bouncing.

5.1 Particles

Each moving sand grain is represented by a `Particle`. It is represented as a structure containing its position, velocity, mass, and diameter. And a boolean variable signalling if the particle is active or not. The uniform mass of a grain is set to 0.04 mg (see equation (4.1)) and the diameter of grains to 0.15 mm, just like in [9].

5.1.1 Generating particles

Particle generation happens differently in different phases of the program. At the start, the `GenerateRandom` function generates `NUM_PARTS` number of randomly positioned particles on the grid, which are returned in a `std::vector`. After this, as the program begins to iterate through this vector, each settled particle, those with zero velocity in every direction, gets reinitialized by the `GenerateParticle` function. This generates a new random location for the particle and a new starting velocity and direction.

```
//Velocity
function GenerateVelocity {
    y = B * frictionvel;
    normV = y / sin(phi);                //normV is length of V=(vx,vy,vz)
    x = normV * cos(phi) * cos(psi);     //calculate direction from wind (psi)
    z = normV * cos(phi) * sin(psi);     //and random lift-off angle (phi)
    return (x, y, z);
}

//Position
function GeneratePosition {
    x = rand();
    z = rand();                  //generate x, z
    y = map.getValue(x, z);      //get height at (x, z)
    if (below hard terrain)      //no sand at (x, z)
        discard;

    norm = computeNorm(x, z);    //calculate normal
    if (dot(norm, wind) > 0)     //if facing downwind
        discard;
    else {                       //if facing upwind
        map.setValue(x, z, y - diameter);
    }
    x, z = getCoordFromIndex(x, z); //make coordinates from index
    return (x, y, z);
}
```

Listing 5.2: Pseudo code of new velocity and position generation.

5.1.2 Update

The `UpdateParticle` function takes one particle at a time. First, it checks if the particle is within field boundaries and converts the current position of the particle into map indices, then it finds the triangle (in terms of OpenGL-drawn triangles) the particle is currently above or under. Then it checks if the particle has reached the surface (`IsUnder` function). In case it has not, it will recalculate the particle's position and velocity based on Δt (equations (3.8) and (3.7)). Otherwise, the particle bounces off the surface with reduced velocity calculated according to equation (3.9). If the particle is too slow at the moment of reaching the terrain, its velocity is set to zero (and it is marked as inactive), to prevent near infinite jumping.

```
//Update particle
function UpdateParticle {
    actual = GetIndex(p.position);      //make indices from coordinates
    CoordCheck(actual);                 //check boundaries
    vertices = PosToTriangle(actual);   //find triangle

    if (IsUnder(p.position, norm, verts[0])) {  //is particle above or under the triangle
        if (p.velocity below threshold) {       //particle too slow
            nearest = FindNearest();            //find nearest vertex in triangle

            //add new grain to nearest vertex
            map.SetValue(nearest, map.getValue(nearest) + p.diameter);

            p.velocity = (0, 0, 0);     //set velocity to zero
            p.active = false;           //deactivate particle
        } else {
            Bounce(p);  //recalculate bounce velocity and new position
        }
    }
}
```

Listing 5.3: Pseudo code of the `UpdateParticle` function.

5.1.3 Collapsing

As the method described in [8] for redistribution of the excess sand was a bit more complicated than necessary and not useful for this implementation, a slightly different idea was introduced here. It is basically the same, but it does not distribute the excess sand between the neighbours, only chooses one with the biggest height difference. As the images in Chapter 6 prove, this creates a very similar smoothing effect to that in [9, 8].

```
//Redistribute
function Redistribute {
    for (i = x−1; i < x+2) {
        for (j = z−1; j < z+2) {
            if (i == x && j == z) continue;  //the same vertex

            //calculating tangent of slope angle as in a right triangle
            //opposite of angle − the height difference
            opposite = map.getValue(x, z) − map.getValue(i, j);

            //adjacent to angle − unit grid distance
            adjacent = (transverse) ? deltax ∗ sqrt(2) : deltax;
            tan = opposite / adjacent;

            //find biggest difference
            if (abs(tan) > abs(maxtan)) {
                maxtan = tan;
                maxopp = abs(opposite);     //maxopp − the biggest height difference
            }
        }
    }
    //correction: difference − allowed max = real excess sand
    maxopp −= repose ∗ adjacent;

    //if angle exceeds repose angle
    if (abs(maxtan) > repose) {
        if (maxtan > 0) {
            //(x, z) is higher − avalanche
            map.setValue(x, z, map.getValue(x, z) − maxopp);
            map.setValue(i, j, map.getValue(i, j) + maxopp);
        } else {
            //(x, z) is lower − fill the hole
            map.setValue(i, j, map.getValue(i, j) − maxopp);
            map.setValue(x, z, map.getValue(x, z) + maxopp);
        }
    }
}
```

Listing 5.4: Pseudo code of `Redistribute`.

First, the `Redistribute` function was implemented as a recursive method with a limited nesting depth. It was called for each grid vertex where a new grain had just landed - and also for each vertex where a new particle was (re)generated, therefore taken away and reducing the height at that point.

The function takes a grid vertex and checks the height difference with each of its neighbours. It calculates the tangent of the slope angle by taking the height difference between the two vertices as the *opposite* and the unit distance on the grid as the *adjacent* side relative to the angle as in a right triangle. This unit distance is readjusted for transverse neighbours (see Figure 5.2).

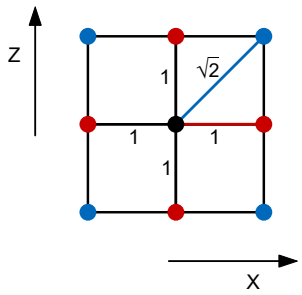


Figure 5.2: Neighbour distances on grid. Distance for direct neighbours (red) is the grid unit. Correction happens for indirect (or transverse) neighbours (blue) - the grid unit distance is multiplied by $\sqrt{2}$.

5.2 Height map

The height map is a simple two-dimensional array of `floats`, marking the height of the terrain at a grid vertex. As in [9], the simulation happens on a 256x256 grid. In [9], they also use a 512x512 grid for some of the shapes, but for our purposes, the smaller one seemed sufficient. Also, in the presented simulation, there is another grid of exactly the same size under the sand map functioning as the hard, unalterable terrain (e.g. rocks, see Figure 5.3).

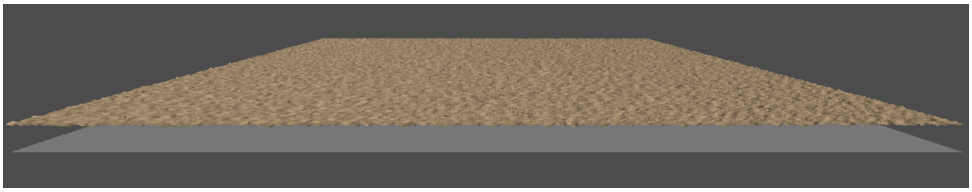
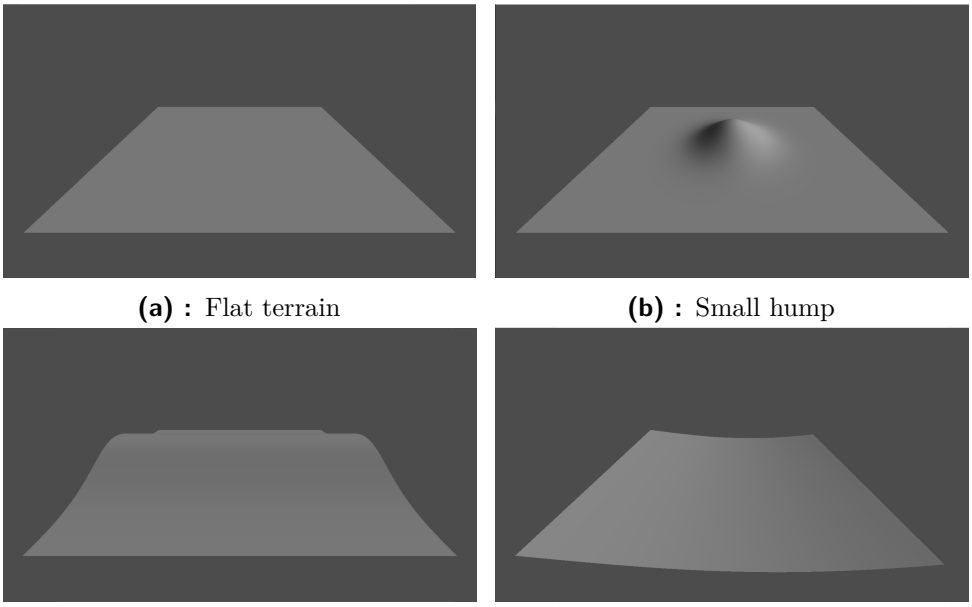


Figure 5.3: Two height maps, terrain and sand layer.

There are four terrains available in the application (see Figure 5.4). One is perfectly flat, setting each point to 0 and the sand map to `sandlevel`. The second has a smooth hump in the middle, which is calculated with a two-dimensional Gaussian function. The third terrain type resembles a gentle dune, created by another Gaussian function, which will be evenly covered with sand. The fourth is a slight valley-like wave calculated from a stretched sine function.

There is a separate class, `MyMap`, that handles most of the work around the grid. It takes care of the map initialization and the particle generation, stores the height fields, and provides functions for setting and retrieving terrain or sand height at a specified index. The map itself is cyclic - if a particle is carried away from above it, that particle appears on the opposite side and continues its movement. Therefore theoretically, the height map could be copied and positioned to simulate larger terrains, as in [9].



(a) : Flat terrain **(b) :** Small hump

(c) : Dune **(d) :** Valley

Figure 5.4: Showing the different terrain types available, without the sand cover. (a) Flat, (b) with a hump in the middle, (c) dune form, (d) gentle valley.

5.3 Visualization

The code from [17] was modified to visualize the changes of the terrain. The `GlutMainLoop` calls the refresh function, that is where the particle update happens. `UpdateParticle` is called for every active particle in the vector of particles. If a particle settled, that means it has its velocity set to 0 in every direction, it is reinitialized by `GenerateParticle`. The simulation is not actually real-time as Wang and Hu's [9]. Here, the terrain gets redrawn only after a certain number of iterations of particle updates.

5.3.1 Triangle strips and normals

At the start and on every redraw, the grid representing the sand cover is calculated in the `generateTerrain` function. The other grid, representing the terrain under is only calculated once at the start, or when its shape is changed from the menu. The height values for both grids are simply stored in two-dimensional `float` arrays, from which the triangle strips are created for drawing. The vertices of these triangle strips are saved in the `terrainVertices` and `underTerrainVertices` for sand and hard terrain

respectively.

```
//Terrain generation
//    px, pz − point to coordinates in
//    dx, dz − unit grid distance on x, z axis
//    vertIndex − index of a vertex; +0 for x, +1 for y, +2 for z
//    idx − triangle indices
//    gen − iteration counter
//    opt − terrain choice
//    attribs − number of vertex attributes: x,y,z, nx,ny,nz ... 6 (8 with texture x,y)

fucntion generateTerrain(float dx, float dz, int opt) {
    if (gen == 0) {
        Map.setDelta(dx, dz);       //set unit grid distance
        initTerrains(opt);          //fill in the height map of hard terrain
        Map.GenerateRandom();   //generate particles
        //generate triangle strips for hard terrain  − same as for sand below
        //calculate normals of hard terrain  − same as for sand below
    }

    pz = − 2.5f; //lowest z coordinate
    for (int z = 0; z < N; z++) {

        px = − 2.5f;  //lowest x coordinate, starting new strip
        for (int x = 0; x < N; x++) {
            vertexArray[attribs ∗ vertIndex + 0] = px;                  //x
            vertexArray[attribs ∗ vertIndex + 1] = Map.getValue(x, z); //y
            vertexArray[attribs ∗ vertIndex + 2] = pz;                  //z

            //save two indices to this strip
            triangleIndices[idx++] = vertIndex;
            triangleIndices[idx++] = vertIndex + N;

            vertIndex++;
            px += dx;   //move one grid unit, end of strip
        }
        pz += dz;   //move one grid unit
    }
    //calculate normals
    for (int iz = 0; iz < N; iz++) {
        for (int ix = 0; ix < N; ix++) {
            //norm = (nx,ny,nz);
            norm = sum(count);          //count − number of neighbouring triangles
                                        //sum the normals
            norm = normalize(norm);   //(nx,ny,nz) = (nx/length, ny/length, nz/length)

            vertexArray[attribs ∗ (iz∗N + ix) + 3] = nx;
            vertexArray[attribs ∗ (iz∗N + ix) + 4] = ny;
            vertexArray[attribs ∗ (iz∗N + ix) + 5] = nz;
            //vertexArray[attribs ∗ (iz∗N + ix) + 6] = textureX;
            //vertexArray[attribs ∗ (iz∗N + ix) + 7] = textureY;
        }
    }
}
```

Listing 5.5: The `generateTerrain` function.

The normal vector for each vertex is calculated as follows. The function takes all the normals of triangles the current vertex occurs in, sums them up and normalizes it. Then the `nx, ny, nz` values of the normal are saved as the next three vertex attributes after the vertex coordinates.

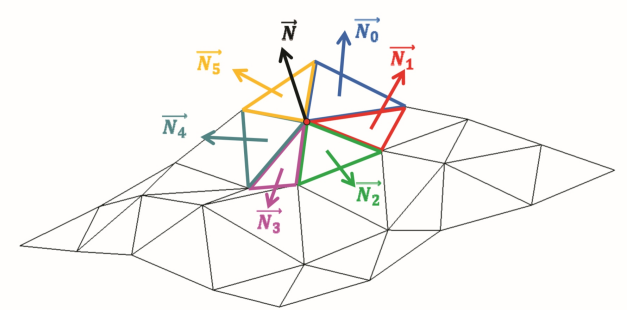


Figure 5.5: Neighbouring normals N_i are summed and the sum normalized to get N (black), the normal of the current vertex. Image taken from [18].

5.3.2 Shaders

The color of the terrain is calculated in the fragment shader, where different colors are passed for the hard terrain and the sand surface. Also, there is an option to switch between texture modes during the simulation: *clear* and *contours*. The clear mode only uses the current color passed to the shader, while with the contours mode chosen, it performs a height calculation, according to which it uses black instead of the color. The goal was to achieve something like in Figure 5.6, the lines marking certain heights of the terrain (see Figure 6.5 for results).

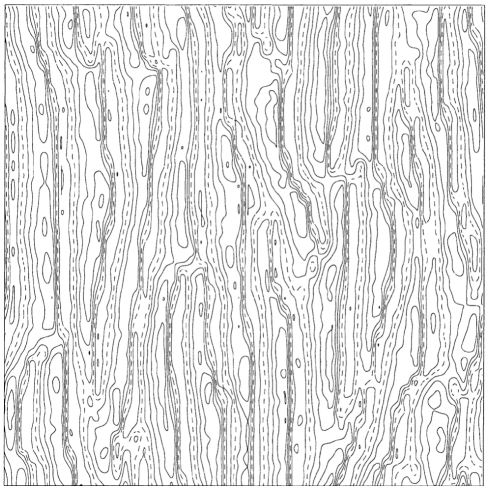
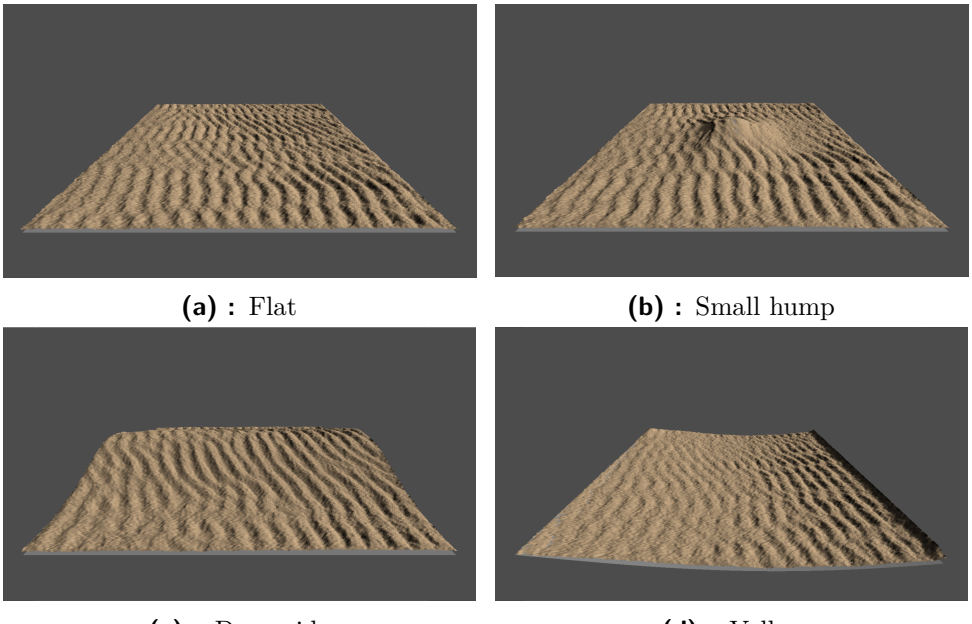


Figure 5.6: Terrain heights marked by black lines. Figure 5(c) from [12].

Chapter 6

Results

The proposed implementation was tested with different hard terrain types and wind velocities. Most of the images show the resulting patterns after up to 3000 iterations, which takes 210 seconds on average (NVIDIA GeForce GTX 1060 Max-Q, Intel Core i5-7300HQ, 2.5 GHz).



(a) : Flat **(b) :** Small hump

(c) : Dune side **(d) :** Valley

Figure 6.1: Various terrain types compared, each after 3000 iteration in 10 m/s western wind.

By default, the resulting images were generated in western wind (from right to left), a sand level of 500 grains and with 200.000 available particles, from which approximately 60-70.000 were active in each iteration. The sequences in Figures 6.2-6.4 show sand ripple evolution after 0, 1000, 2000, and 3000 iterations in that order, with the starting point being a completely flat surface in each case, but in stronger and stronger winds. The pattern gets more pronounced and their wavelengths increase in stronger winds, with higher crests and wider ripples. This is in agreement with Bagnold's observations [13], and consistent with the simulation results of Wang and Hu in [9, 8], who also compared their results to Bagnold's field observations.

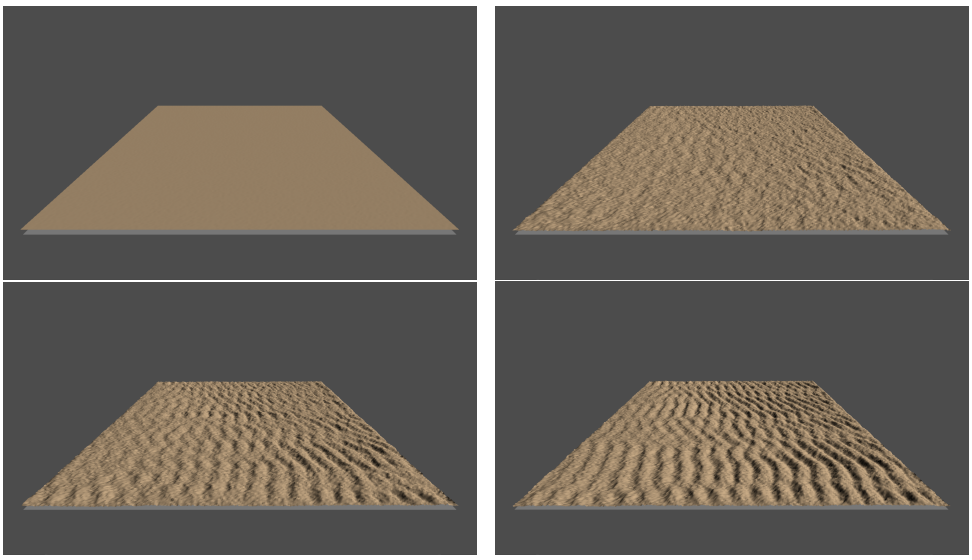


Figure 6.2: Sand ripples in default wind (10 m/s).

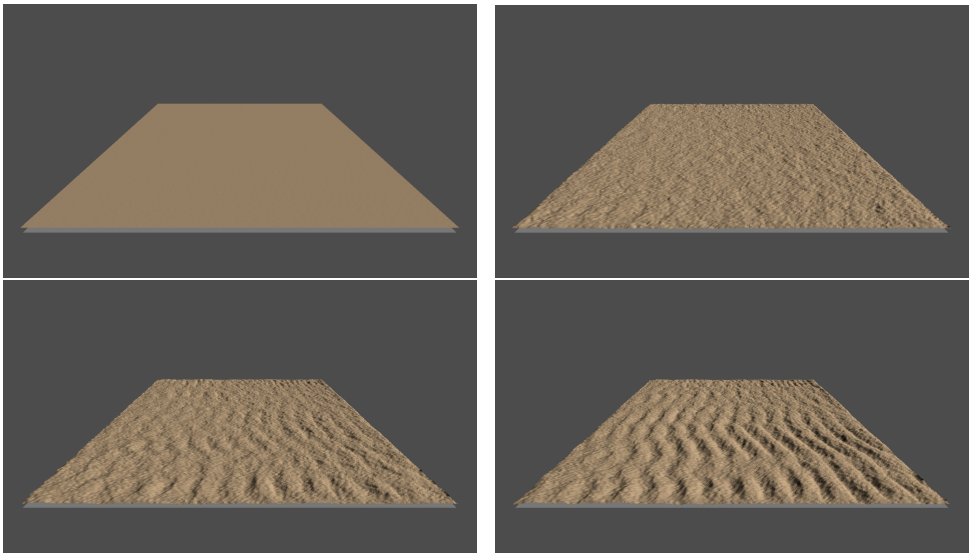


Figure 6.3: Sand ripples in stronger wind (12 m/s).

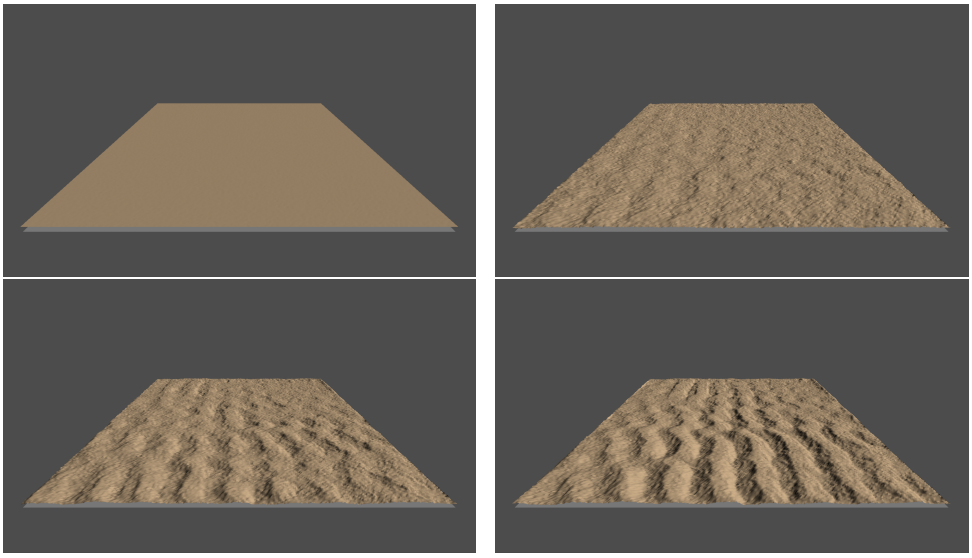


Figure 6.4: Sand ripples in strong wind (15 m/s).

As seen in Figure 6.1, the various unalterable terrain types under sand have an interesting effect on ripple formation as sand grains accumulate differently around obstacles and on flat surfaces. On an absolutely smooth surface, ripples form freely, whereas a hump in the middle causes the grains to settle on the leeward side from where they will hardly get caught up again by the wind. Slightly more oblique ripples form on the side of a dune, as gravity makes the grains to slide down on the slope. On the contrary, the ripples become flatter, the higher they are on the slope in a shallow valley.

As mentioned in Section 5.3.2, there is an option to show height contours on the sand surface, which can be seen in Figure 6.5. The dune shaped terrain was chosen for this, as the oblique ripples on the side of the dune make for an interesting sight. These lines can also be seen in Figure 6.6, which illustrates the avalanche effect. The hump in the middle is too high for the sand to stay on it too long, therefore it slowly flows down in a glacier-like fashion. Sand grains can also be seen largely accumulating at the bottom of the windward slope. The black parts are either a level area or the hard terrain with no sand to cover it.

According to the information about snow, in a similar situation to Figure 6.6, with a reasonably steep hump or other obstacles, snow would show less downward flow because of sintering. After a certain *long enough* period (discussed in section 4.3), the snow particles would bond strongly enough to become an erodable surface instead of a loose particle layer. It is safe to assume, that this would lead to the formation of sastrugi and snow edges. As for snow, the ripples on flat terrain would become flatter and also lower in height, probably transforming into sastrugi after some time.

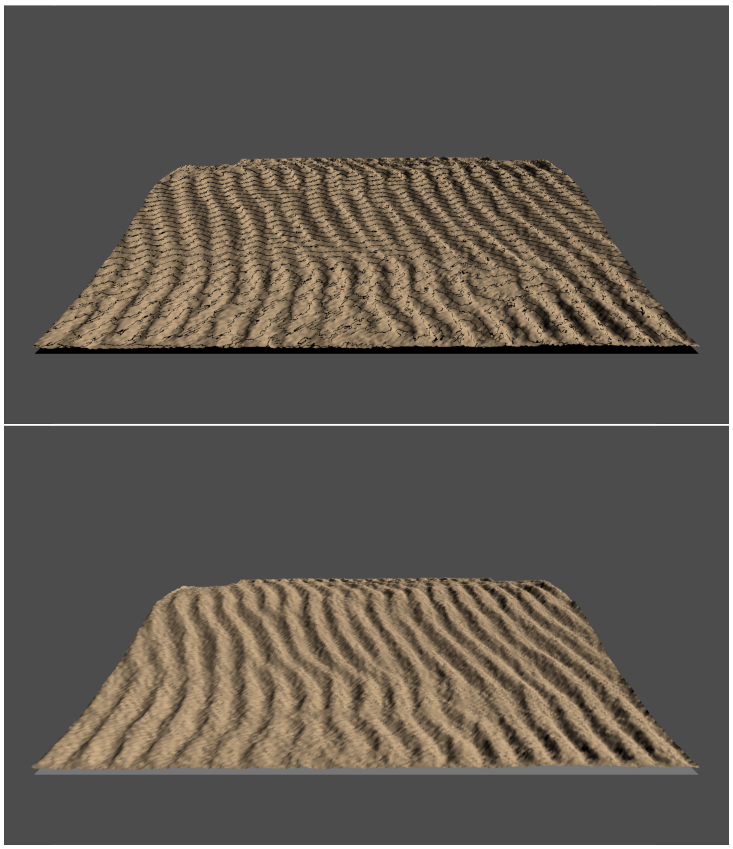


Figure 6.5: Comparing the same patterns with and without contours (both western wind, 10 m/s, 4000 iterations on dune shaped terrain).

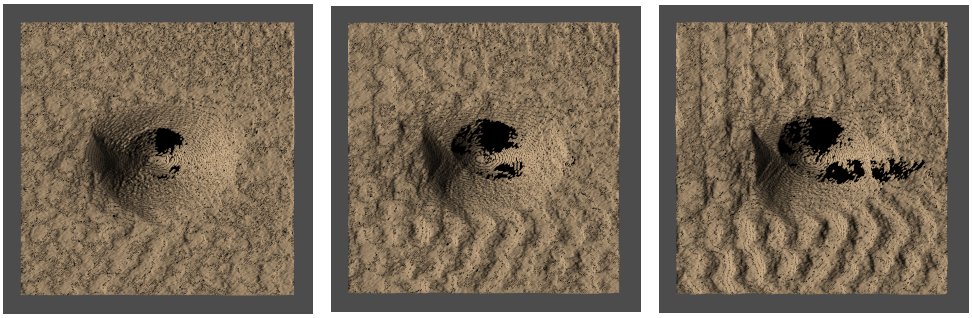


Figure 6.6: Showing the avalanche effect on a steeper hump, from top view allowing for better observation (western wind, 15 m/s, steep hump, after 1000, 2000 and 3000 iterations left to right). The black parts either mark a larger level area or the unalterable terrain showing from under the sand, where all grains have already fallen or been blown away.

Chapter 7

Conclusion

The initial aim of this thesis was to achieve credible snowy landscape simulations to be able to compare them with real snow features. However, this task proved to be more complex than expected.

First, I thoroughly studied Wang and Hu's [9, 8] sand simulation method pursuing the idea to simulate snow features created by wind erosion. This thesis presents an implementation of this method and then introduces the presumably necessary new parameters that would have to be included for snow. The Snow chapter gives a detailed summary of the information backing the assumptions made about snow and providing a comparison of sand and snow particles and their behavior.

The generated results prove that this implementation is functional and can yield reasonably satisfying results, although not as sophisticated as Wang and Hu's in [9, 8]. However, I compare ripple formation on various terrain types instead of introducing vegetation.

I believe this work provides an acceptable basis for further study of snow particle movement and wind erosion on snowy surface, allowing for the extension of my implementation.

Bibliography

[1] S. Filhol and M. Sturm. Snow bedforms: A review, new data, and a formation model. *Journal of Geophysical Research: Earth Surface*, 120:1645–1669, 2015.

[2] Clea Bertholet. Snow bedform growth as a function of wind speed and snow age. Undergraduate Honors Theses 1295, University of Colorado, Boulder, 2017.

[3] P. Fearing. Computer modelling of fallen snow. In *In Proceedings of SIGGRAPH 2000, New Orleans, LA USA*, pages 37–46. ACM Press/Addison-Wesley Publishing Co., 2000.

[4] R. W. Sumner, J. F. O'Brien, and J. K. Hodgins. Animating sand, mud, and snow. *Computer Graphics Forum*, 18:17–26, 1999.

[5] Onoue K. and Nishita T. Virtual sandbox. In *In Proc. the 11th Pacific Conf. Computer Graphics and Application*, pages 252–259, October 2003.

[6] Bell N., Yu Y., and Mucha P. J. Particle-based simulation of granular materials. In *In Proc. the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 77–86, July 2005.

[7] Y. Zhu and R. Bridson. Animating sand as a fluid. In *SIGGRAPH, July 31-August 4*, pages 965–972, 2005.

[8] N. Wang and B.G. Hu. Aeolian sand movement and interacting with vegetation: A gpu based simulation and visualization method. *Plant Growth Modeling And Applications*, pages 401–408, 2009.

[9] N. Wang and B.G. Hu. Real-time simulation of aeolian sand movement and sand ripple evolution: a method based on the physics of blown sand. *Journal of Computer Science and Technology*, 27:135–146, January 2012.

[10] Néstor Galina. https://flic.kr/p/8pzqzw.

[11] Sathish Jothikumar. https://flic.kr/p/bbtn9n.

[12] Steven R. Bishop, Hiroshi Momiji, Ricardo Carretero-González, and Andrew Warren. Modelling desert dune fields based on discrete dynamics. *Discrete Dynamics in Nature and Society*, 7:7–17, 2002.

[13] R. A. Bagnold. *The Physics of Blown Sand and Desert Dunes*. Methuen Co. Ltd, 1941.

[14] Nitsan Bar, Tov Elperin, Itzhak Katra, Hezi Yizhaq. Numerical study of shear stress distribution at sand ripple surface in wind tunnel flow. *Aeolian Research*, 21:125–130, June 2016.

[15] Alberto Crespo Iniesta, Miguel Diago, Thomas Delclos, Quentin Falcoz, Tariq Shamim, and Nicolas Calvet. Gravity-fed combined solar receiver/storage system using sand particles as heat collector, heat transfer and thermal energy storage media. *Energy Procedia*, 69, 09 2014.

[16] René O. Ramseier and Charles M. Keeler. The sintering process in snow. *Journal of Glaciology*, 6, 1966.

[17] Jaroslav Sloup, Tomáš Barák, 2019. Demo to display a terrain using triangle strips. B0B39PGR course, ČVUT, FEL.

[18] Petr Felkel, Jaroslav Sloup. Tessellations of simple 3D objects, 2017. Lecture notes, B0B39PGR course, ČVUT, FEL.