Computers & Graphics 71 (2018) 14–22

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Real-time GIS-based snow cover approximation and rendering for large terrains☆

Benjamin Neukom[a,b,*], Stefan Müller Arisona[a,b], Simon Schubiger[a]

[a] Institute of 4D-Technologies, University of Applied Sciences Northwestern Switzerland FHNW, Switzerland
[b] Esri R&D Center Zürich, Switzerland

ARTICLE INFO

Article history:
Received 13 April 2017
Revised 11 October 2017
Accepted 16 October 2017
Available online 16 November 2017

Keywords:
Real-time visualization
Snow approximation
GIS
GPGPU
Game engine

ABSTRACT

Various terrain visualization techniques based on geographic information system (GIS) data already exist. One major drawback of existing visualizations is that they do not capture seasonal variations well. Besides vegetation variations, in colder areas this particularly also applies to snow cover. In this paper, we propose a real-time multi-scale snow cover approximation and visualization for large terrains. The computation runs on a large grid, calculates the snow/water equivalent based on precipitation data from a GIS and snowmelt based on a physically-based solar radiation calculation combined with a degree-day snowmelt approach using level of detail (LOD). The snow visualization is divided into two parts: Zero thickness snow cover textures are generated for distant views. For close up views the terrain's height field is modified using displacement maps and tessellation to produce thick snow covers. The GPU-based data-parallel computation and the visualization run on the GPU in real-time on a modern desktop computer. The implementation is tested using a real area in the Swiss Alps, with a size of 14.16 by 12.88 km, a grid resolution of 222 × 206, and a time step of 1 h. We compare the rendered results spanning several months with a time series of photographs from webcams for visual accuracy.

1. Introduction

Visualizations of elevation data from geographic information systems (GIS) already exist with sophisticated techniques [1–3]. The most common way to enhance realism of such renderings is to project aerial photography onto the digital elevation model (DEM). As previously pointed out by Premoze [4] several problems arise from this approach: For example, the orthoimage may still contain shadows which leads to visual artifacts if visualizations for different times of the day are created. Also, the visualizations do not capture seasonal variations and remain static. If a season specific rendering is desired, the user needs to adapt the landscape by hand. In colder climates, snow drastically changes the appearance of landscapes during winter. Premoze solves this by simulating snow cover and then rendering it over the original aerial imagery. His method aims at distant views and produces zero thickness snow textures for static images rendered by a ray-tracer and does not run in real-time. Other snow visualizations and simulations exist [4–6] but they are either not suitable for large terrains or do not run in real-time.

In this paper we build upon Premoze's work and propose a method to simulate and visualize snow cover for large terrains in real-time with different level of details. Our goal is to create a GIS-based snow cover approximation, which uses generally available input data such as high-resolution height maps, aerial imagery and temperature/precipitation time series, does not require manual intervention, runs at interactive framerates and still produces physically plausible as well as visually pleasing renderings. To the best of our knowledge, this is the first paper that realizes real-time multi-scale snow cover computation.

To achieve our goals, we compute snow accumulation by assuming precipitation below a certain temperature threshold to be snowfall. Our method uses different levels of details (LOD) for improved performance and produces thick snow covers for close-up views and zero thickness textures for distant views. The thick snow cover is generated by displacing the landscape according to the amount of snow generated from the computation. After the accumulation, the snow is redistributed according to the relationship described by Blöschl et al. [16] to account for snow depletion based on slope and wind. To compute snowmelt, a degree-day approximation similar to the one used by Premoze et al. [4] is run on the GPU. The snowmelt simulation calculates the solar radiation

☆ This article was recommended for publication by Bedrich Benes.

* Corresponding author at: Institute of 4D-Technologies, University of Applied Sciences Northwestern Switzerland FHNW Switzerland.
E-mail addresses: BNeukom@Esri.com (B. Neukom), SArisona@Esri.com (S. Müller Arisona), simon.schubiger@fhnw.ch (S. Schubiger).

https://doi.org/10.1016/j.cag.2017.10.003
0097-8493/

Table 1
Comparison of previous work. In the performance column, R stands for real-time, NR for non-real-time, and I for interactive. In the physically-based and large-scale column, PA indicates partially realized techniques. In the snow model column, H stands for heuristic and P for particle-based. As shown, the majority of previous works focused on small-scale, non-real-time methods. The methods proposed by Premozě et al [4] and Foldes and Benes [5] work for large scenes but do not run in real-time.

	Performance	Artist interaction	Physically-based	Large-scale	Snow model
Premože et al. [4]	NR		PA	PA	H
Fearing [6]	NR			PA	P
Haglund et al. [7]	R	√			P
Feldman and O'Brien [8]	NR		√	√	P
Ohlsson and Seipel [9]	R	√			H
Moeslund et al. [10]	NR		√		P
Wang et al. [11]	R		√		P
Saltvik et al. [12]	IN		√	PA	P
Foldes and Benes [5]	NR			√	P
Hinks and Museth [13]	NR				P
Festenberg and Gumhold [14]	NR	√			H
Reynolds et al. [15]	R				H
Ours	R		PA	√	H

Fig. 1. (A) The scene after the first snowfall with snow already melting and not sticking to the ground well (as described in Section 4.2). (B) The same scene after more snowfall, with the ground almost fully covered in snow. (C) The scene in late January with even more snow and the tree trunks almost fully buried using displacement maps as described in Section 4.3.

for each cell at the given time of the year by approximating the sun's position as described by Swift [17] and uses the solar radiation as an index for the snowmelt calculation. The inputs for the computation are temperature and precipitation.

In summary, our main contributions are:

- A snow cover approximation for large terrains based on GIS data with visual validation using webcam time series from the Swiss Alps. Using our snow cover approximation only the initial degree-day factor has to be adjusted by hand, a process which took about 10 minutes to achieve realistic snow covers for a whole season.
- An LOD implementation for the snow computation which calculates high LOD snow covers close to the camera and a lower LOD spanning the whole scene.
- An implementation running in real-time using a game engine.

Fig. 1 illustrates the results of this work. On the left-hand side the scene is shown after first snowfall with snow already melting again. In the middle the scene is shown with the ground fully covered in snow and on the right-hand side the displacement of the snow is shown with tree stumps covered in snow the The computation of one timestep took about 0.3 ms and can be rendered with over 100 frames per second on a modern desktop computer.

This paper is organized as follows: In Section 2 we give a comprehensive overview of existing snow simulations. Section 3 lays out the theoretical foundation for our approach. In Section 4 we describe how the approximation is implemented on the GPU to achieve real-time performance. In Section 5 we present the results of our work. We describe the data used for the computation and then compare the results to real photographs to assess visual accuracy. We also evaluate the performance with varying time steps. The paper concludes with Section 6 where we also provide possible future directions.

2. Related work

The related works can roughly be categorized into two categories: Heuristic models that approximate snow accumulation instead of simulating particles to accumulate snow. Particle-based that models simulate the behavior of snow particles and their accumulation on objects.

2.1. Heuristic models

Heuristic models approximate snow accumulation instead of simulating particles. Premože et al. [4] introduced a system which uses techniques from geology to simulate snow accumulation and snowmelt for mountainous terrains. They assume rain under a certain air temperature to be snow and use a degree-day simulation for the snowmelt. Their method produces zero-thickness snow and is intended for distant scene views. Our work is based on Premozě's idea of using a degree-day simulation for snowmelt. Foldes and Benes [5] proposed another approach by using ambient occlusion to compute the shape of the snow in ditches or secluded areas for large terrains. Their method does not deal with falling snow or wind but rather with ablation and snow melting and is therefore better suited for distant views of large scenes.

Other research is inspired by shadow buffer techniques to approximate the accumulation of snow. Ohlsson and Seipel [9] proposed a per-pixel method similar to shadow mapping, using a depth buffer to determine how much snow a surface should receive. The final amount of snow is then calculated depending on the surface slope. As the last step, 3D noise is applied for convincing lighting of the snow. This method provides good results for smaller scenes in real-time. For larger scenes, their method is not suitable as the depth buffer map would require a very high resolution. Reynolds et al. [15] also presented a method that uses shadow mapping to compute occlusions for snow accumulation. Their main

contributions are the addition of mapping multiple accumulation buffers to each object. This allows simulating a more dynamic, moving scene where snow accumulates. To visualize snow height, they do not generate new polygonal meshes as described by Ohlsson but rather use tessellation shaders and displacement maps. In our visualization implementation, we also use displacement maps similar to Reynolds' method.

Festenberg and Gumhold [14] proposed a geometrical approach which improves Haglund et al. [7] height map model and is inspired by real world observations. Their model is based on the observation that a smooth curve is formed near the edges of snow covered surfaces. The form of this curve depends on various factors like snow depth or different weather conditions, but the basic shape always remains the same. In their implementation, they use a height span map described by Onoue and Nishita [18] to compute the snow cover probability distribution. Their method produces good looking results, with good performance for small scenes. It is also ideally suited to animate accumulated snow, since the snow cover probability distribution is only calculated once and can be used to determine a range of snow heights. However, the algorithm does not work well with certain geometries like small crests or small planes and its initial parameters have to be chosen carefully for any given scene to produce good results.

2.2. Particle-based models

Particle-based methods model snow cover by simulating particles and then accumulating snow where particles collide with objects. One of the most sophisticated particle based methods was described by Fearing [6]. His method is a two-phased particle based simulation which produces thick layers of snow by generating new meshes. The *accumulation phase* computes how much snow a surface receives by shooting particles towards the sky to compute possible occlusions. Each surface therefore has independent control over its sampling rate and density. This phase allows for phenomena such as flake flutter, the influence of wind and the snow's ability to stick on uneven surfaces. The *stability phase* moves snow away from physically unstable areas. This is achieved by simulating small, simultaneous avalanches which move the snow to more stable areas by calculating the angle of repose between neighboring triangles. Thus, the snow is redistributed to neighboring areas if the angle is too steep. Fearing's method produces visually beautiful scenes with a minimal amount of interaction by a designer, but it cannot be used for a real-time simulation due to its complexity.

Haglund et al. [7] introduced a method which uses a two-dimensional height matrix that stores the current snow depth. When a snowflake hits the ground the value at the corresponding entry in the height matrix is increased. The height matrices are then used to triangulate and render the area with snow cover in real-time. The combination of appropriate blending and Gouraud shading gives the impression of snow slowly accumulating on a surface. One drawback of their method is that the height value matrix has to be manually placed and therefore requires much work for large scenes. Also, it is not suitable for arbitrary landscapes but rather for surfaces without steep slopes, since snow redistribution is not handled in their simulation. Saltvik et al. [12] have extended Haglund's method with a dynamic wind simulation and implemented it on general purpose graphics processors (GPGPU) to achieve interactive framerates. Their simulation runs in real-time and therefore simulating a whole season is not feasible.

Several studies build on Fearing's particle simulation and snow stability test. Feldman and O'Brien [8] use the fluid simulation described by Fedkiw et al. [19] to model a more sophisticated wind field as used by Fearing. They use the wind field to compute how snow moves through the air and is accumulated on the objects. To displace the accumulated snow, they apply Fearing's avalanche method. Moeslund et al. [10] improved Fearing's method by modelling the physical properties of snowflakes and their movement more precisely. They also simulate a wind field similar to Feldman. Wang et al. [11] simulate snow using wind fields for very small scenes in real-time. Hinks and Museth [13] proposed a method that builds on the work of Moeslund et al.; however instead of using polygonal meshes to represent snow, they use level set surfaces which are inherently smooth. Explicit geometries as used by Moeslund or Fearing tend to produce sharp edges and are computationally expensive when handling unstable snow areas.

Additional methods model the heat exchange within a scene to model snowmelt. Muraoka and Chiba [20] described a particle-based method using virtual temperature and air fields for snow accumulation. They describe a technique for changing the shape of the snow cover where snowmelt is affected by sunlight, ground temperature and radiation heat from surrounding objects. Temperature values are stored in a voxel space. Their system provides a detailed simulation for close up scenes but is not suited for a real-time visualization as the computational costs are too high.

Our method does not use the particle-based approach but we nonetheless incorporate some ideas proposed in this Section. We also use a grid-based approach described by Haglund et al. and extend his method, similarly to Reynolds [15] to use displacement instead of meshes to generate thick snow.

3. Computation

In this section, we describe how the approximation of snow accumulation, snowmelt and redistribution of the snow according to a given terrain are computed.

3.1. Snow accumulation

The first phase of the computation is snow accumulation. The terrain is divided into cells. The cell size can be adjusted to fit the terrain data. We chose 64m as the side length of the cells. Each cell stores its snow water equivalent (SWE), the amount of water contained in the snowpack per m^2. Similarly to Premozě, the amount of accumulated snow is determined by assuming precipitation below a specific air temperature threshold to be snow and by calculating the snow water equivalent for every cell. This threshold temperature is typically set to 0°C. To achieve smoother snow cover transitions we linearly interpolate between snow and rain at the air temperature threshold as described by Barringer [21]. Air temperature and precipitation are passed into the computation at a base location with a lapse rate. For the examples in this paper, we adopted a temperature lapse rate of 0.6°C per 100 m of altitude and a precipitation lapse rate of 10 mm per 1000 m of altitude [22].

3.2. Snowmelt

The second phase of the computation is snowmelt. The most common ways to simulate snowmelt in geology are the empirical degree-day method and the more physically-based energy balance approach. The degree-day method uses air temperature as an index to calculate snowmelt. As described by Rango and Martinec [23] the degree-day method is still in heavy use since simulations based on it provide accurate results and do not rely on a lot of input data.

The physically-based energy balance models have become increasingly complex in recent years. At the core of the physically-based snowmelt procedures are the energy balance equations [24]. In order for these simulations to work, they require accurate data and interpolation mechanisms for wind speed, air temperature, air

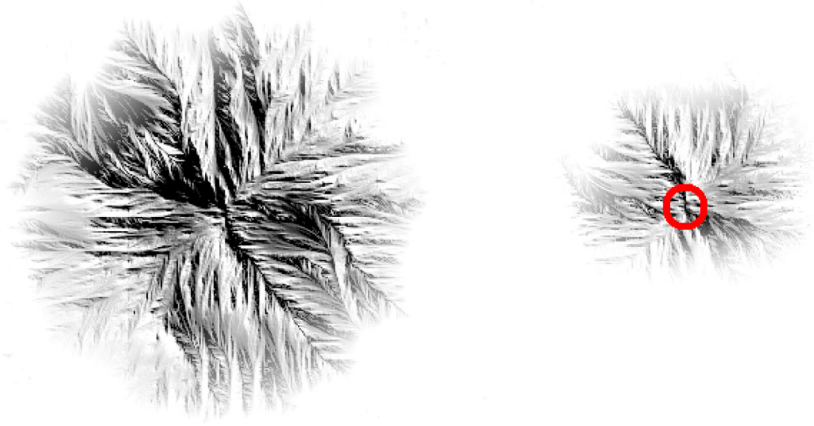

Fig. 2. Two textures with the amount of snow coded as grayscale values from an artificially generated mountain. Note that darker intensities represent more snow cover. (A) The mountain after fresh snow has fallen, fully covered in snow. (B) The same mountain with snow melted considerably. We can see the influence of the aspect for the snowmelt. The red circle marks the top of the mountain. In the northern hemisphere south facing slopes receive more solar radiation and therefore snow melts faster. The visual result is comparable to Maréchal et al. [29] radiation calculation, where they used a (non-real-time) physically-based thermal simulation to calculate snowmelt. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

pressure, humidity and precipitation [25]. In our approach, we decided to use the use the empirical degree-day method and combine it with a physically-based solar radiation calculation as described below. Computing the energy balance equations would have resulted in poor performance and the amount of input data needed to compute snowmelt for large terrains is not generally available.

Normally the degree-day method is used with a daily time step, but as described by Hock [26] it can be adapted to an hourly time step by adding a radiation component. Using an hourly time step provides accurate results for small time frames. The basic expression relating the snowmelt M to the temperature index C_m and degree-days is as follows:

$$M = C_m(T_{air} - T_{melt}) \tag{1}$$

where M is the snowmelt in mm. C_m is the degree-day factor in $\text{mm}^{\circ}C^{-1}d^{-1}$, which is the amount of melt that occurs for a positive degree-day. T_{melt} is the critical temperature at which point snow starts to melt. It is typically around 0°C. We again applied linear interpolation for the melt threshold to provide smoother snow transitions. C_m is calculated as follows:

$$C_m = R_i * (1 - A) \tag{2}$$

The radiation index R_i is the radiation that a surface with a given slope and aspect receives, normalized to a surface with a horizontal alignment. We used the algorithm proposed by Swift [17] to calculate the daily total solar radiation. Premože et al. [4] used a daily time-step in their simulation [4]. We decided to use an hourly time-step to provide more accurate results for smaller time frames. Solar radiation follows a diurnal pattern and reaches its peak during the afternoon. Monteith [27] proposed a sine-curve approximation for hourly radiation calculation. The radiation index at a given hour of the day t is calculated as follows:

$$R_i(t) = \frac{\pi R_i}{2} * sin\left(\frac{\pi t}{D}\right) \tag{3}$$

where sunrise is at $t = 0$ and sunset is at $t = D$. The albedo A used in Eq. (2) is important for estimating the amount of radiation the snowpack absorbs. The albedo decreases with time as proposed by Eggleston et al. [28]. They noted that the albedo varies from around 0.8 for freshly fallen snow to 0.4 during melting. Eq. (4) approximates this variation with an exponential function:

$$A(t) = 0.4 * (1 + e^{-0.2t}) \tag{4}$$

where t is the time since the last snowfall.

3.3. Snow redistribution

The third computation phase distributes accumulated snow. Premože et al. [4] deposit snow by elevation. However, they are not concerned with the final snow height since they only render flat textures where snow accumulates only [4]. We are interested in the snow height and therefore need a more accurate snow redistribution mechanism. Fearing [6] used a snow stability test which moves snow away from unstable positions. However, the results are computed iteratively and the test is therefore unsuitable for real-time application with large terrains.

Wind and inclination are generally accepted as the major driving forces in snow redistribution [16]. Unfortunately, a wind simulation as proposed by Feldman and O'Brien [8] or Moeslund et al. [10] is not feasible for large terrains in real-time. We therefore decided to use an empirical interpolation that uses terrain slope and curvature to approximate the final water equivalent. The curvature is used to approximate snow depletion on mountain tops which are caused by wind as well as snow accumulation in gullies. The interpolation function we used was proposed by Blöschl et al. [16]:

$$f(w) = w * (1 - f(\theta))(1 + a_3 * \kappa) \tag{5}$$

where w is the snow water equivalent, a_3 is an empirical coefficient which is used to weight the effect of the curvature on the final water equivalent interpolation, θ the slope and κ the curvature. We used 50 as suggested by Blöschl. The function $f(\theta)$ defines the influence of slope on the water equivalent and is given by:

$$f(\theta) = \begin{cases} 0 & \theta < 10^{\circ} \\ \theta/60^{\circ} & \text{otherwise} \end{cases} \tag{6}$$

The curvature of the terrain is computed in a pre-processing step by calculating the second derivative of the terrain as suggested by Zevenbergen and Thorne [30]. By using this interpolation scheme, we account for snow accumulation in gullies as well as snow depletion on steep slopes and tops. In Fig. 4 we can see the results of the interpolation: Steep slopes and mountain tops accumulate much less snow than flat areas and gullies.

4. Implementation

Modern game engines are considered suitable for non-game virtual worlds since they provide features such as high-performance rendering and high-quality dynamic lighting out of the box [31].

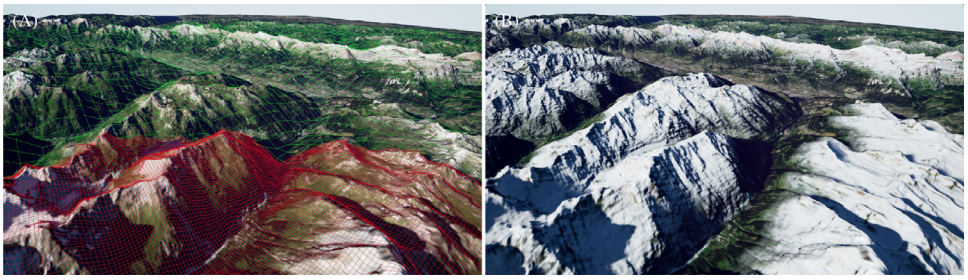

Fig. 3. (A) Grid level of detail: The red grid (166 × 118 cells, 6.695 × 4.7825 km) represents the higher level of detail closer to the camera while the green grid (166 × 118 cells, 53.56 × 38.26 km) represents the large low level of detail grid for the snow cover computation. When the camera moves, new cells in the high LOD grid (red) have to be update to include the snow level computation for the current current time step. The lower LOD grid (green) is not affected by the camera and therefore does not need to be recomputed when the camera moves. (B) The scene covered in generated snow using the LOD as described in Section 4.1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We decided to use the Unreal Engine to implement our approach. Using Unreal Engine, landscapes can be imported from digital elevation models and rendered at interactive framerates with adaptive LOD using tessellation. Unreal Engine also provides a device independent abstraction over the underlying graphics pipeline which we used to implement our computations. In Algorithm 1,

Algorithm 1 Snow computation pseudocode which runs on the GPU.

```
1:  function ComputeCell
2:      if Precipitation > 0 then
3:          LastSnowfall ← 0.
4:          if T_air > T_rain then
5:              Albedo ← 0.4
6:          else
7:              SWE ← SWE + Precipitation ∗ Area
8:      if SWE > 0 then
9:          if LastSnowfall >= 0 then
10:             Albedo ← 0.4 ∗ (1 + exp(0.2 ∗ LastSnowfall))
11:         if T_air > T_melt then
12:             R_i ← SolarRadiationIndex(t)
13:             C_m ← R_i ∗ (1 − Albedo)
14:             M ← C_m ∗ (T_air − T_melt)
15:             SWE ← SWE − M
16:             SWE ← InterpolateSWE()
17:
```

the pseudo code which was derived from the previous Section is shown. This code is executed for each cell and for each time step. As shown, each cell is independent of neighboring cells so the simulation can be run in a highly parallel way and can therefore be implemented as a compute shader that executes on the GPU.

The compute shader writes the amount of snow-water equivalent coded as grayscale values into a texture of the dimension as the computation grid. This generated texture is later used for the displacement of the snow height, and the fragment shader modifies the aerial image to include the generated snow. Refer to Fig. 2 for two examples of such a texture. As previously pointed out, our method visually compares to the physically-based approach of Maréchal et al. [29] but in contrast runs in real-time.

4.1. Level of detail

To improve performance for large scenes, we added two levels of detail (LOD): The low LOD consists of a coarse grid, which spans over the whole simulation area. The high LOD is calculated in a grid of high resolution covering the area near the camera. When the camera position or orientation changes, our implementation determines which grid cells from the low LOD enter the high LOD. For these cells, the snow level computation is carried out up until the current computation time step from the beginning of the simulation. Recalculation using the high resolution grid is necessary, because the results from the low LOD computation might not be accurate enough for close up views. As we will show in Section 5.4 these computations are carried out with low overhead. Cells which remain in the high LOD since the last update do not need to recalculate all time steps. The low LOD grid is not affected by camera position and orientation, and therefore no additional computations on this grid are required when the camera moves. Fig. 3 shows the two LOD grids for a large scene in the Swiss Alps (6.695 × 4.7825 km for the high LOD and 53.56 × 38.26 km for the low LOD). Using this LOD technique we can approximate snow cover for very large scenes and still achieve high frame rates. Refer to Section 5.4 for an in-depth discussion on performance of our implementation.

4.2. Snowshading

To shade the landscape with snow cover we use the snow height texture with the amount of snow coded as grayscale values. We linearly interpolate between the aerial image and a snow texture according to the snow amount from the grayscale texture. Furthermore, we use the snow water equivalent relationship described by Blöschl et al. [16] by linearly interpolating the amount of snow which gets rendered between the pixel normal vector and a vector which is horizontal to the ground. This interpolation reduces artifacts where the cell sizes are too large to capture small scale slope variations in the terrain. To improve the rendering of wet and warm snow we use the albedo from the simulation to decrease the angle at which snow sticks to the ground. As described by Trabant et al. [32] this angle can reach values as low as 5° for wet snow with a temperature of over 0°.

4.3. Snow thickness

To produce the appearance of snow rising and disappearing throughout the season we used displacement maps [33]. Displacement maps modify the actual geometric vertex positions of a surface, and are normally used in combination with tessellation and tessellation shaders.

Displacement maps have already been used by Barré-Brisebois [34] or Yanyun et al. [35] to modify snow cover on the ground interacting with surrounding objects. We do not generate new meshes for the snow as described by Haglund or Fearing, but rather use displacement maps to modify the height of the landscape which gives the appearance of snow accumulation. Reynolds already used this technique for small-scale snow accumulation patterns and we adopt it for larger scales [15]. In contrast to Haglund's

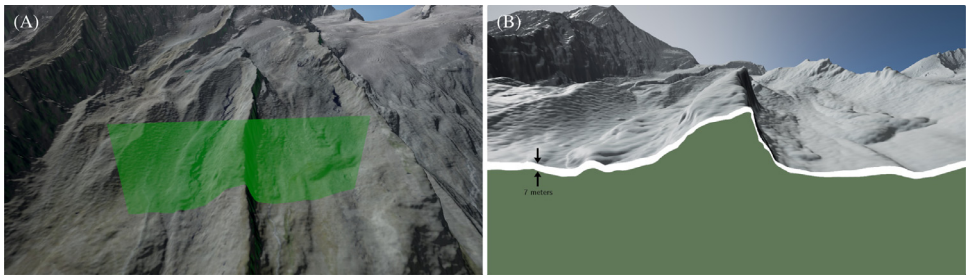

Fig. 4. On the left-hand side the original terrain with a projected aerial image is shown. The green plane marks the cross section used for the illustration on the right-hand side, which shows the terrain with with snow cover: The green area represents the surface without snow accumulation. The white stripe represents the accumulated snow, with a thicker snow level on flat areas and a thinner on steep slopes. According to this stripe, the landscape was displaced as described in Section 4.1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

method, in which the height matrices for the final snow meshes have to be hand placed, our solution does not require additional manual interaction by a designer.

We use the gray-scale texture with the snow height generated from the compute shader as a displacement map. Each vertex from the landscape is then displaced according to its normal vector to match the simulated snow height. To increase the performance, displacement maps are only applied close to the camera, i.e. on the low LOD grid. For distant views of a scene the displacement is not visible and can therefore be omitted. For our test scenes, additional tessellation was not needed as the landscape mesh already contains enough vertices to produce good results for the displacement. If tessellation is needed for other scenes, it can easily be added to the existing solution.

Fig. 4 illustrates the results of the snow displacement. On the left hand side, the original scene at an altitude of 3500 meters, without snow cover is shown. The green plane represents the cross section shown on the right-hand side, which is used to visualize the displacement. On the right-hand side, the lower green area represents the surface without snow accumulation. We can see that on the steeper slopes less snow is accumulated and therefore the surface is not displaced as much as on the flatter ground.

Fig. 1 illustrates how displacement and snow shading techniques affect the look of the scene. With displacement, the rising and sinking snow can be visualized throughout a whole season with accurate snow heights.

5. Results

In this Section, we present the results of our snow cover approximation. We first describe the input data for testing and evaluation, and describe how we set up the initial conditions for the computation. The results of the simulation are then compared to photographs to assess the visual accuracy. At the end of the Section we discuss the performance.

5.1. Input data

Elevation and climate data are globally available. We chose a region around the village of Zermatt in Switzerland. The region features rugged terrain with steep slopes and mountains reaching above 4000 m altitude where snow does not fully melt during summer. We used the swissALTI3D digital terrain model with a 2 m spatial resolution. The digital elevation model was acquired using LIDAR sensors with an accuracy of $\pm$ 0.5m up to 2000 m altitude and $\pm$ 1-3m above 2000 m. Additionally we used the SwissImage aerial imagery with pixel sizes of 2.5 m to texture the digital terrain model. The hourly precipitation and temperature data are from a station located in Zermatt. The data is collected from the SwissMetNet automatic meteorological network database. The time frame of our test case is from October 2015 until April 2016 [36].

5.2. Initial conditions

To provide accurate initial conditions, we need to estimate the snow line for permanent snow cover. For the Swiss Alps the snow line is somewhere around 2500 m–2800 m depending on the area's aspect [37]. We again used Blöschl's approximation to estimate the initial amount of snow [16].

$$w = (a_1 + a_2 z)(1 - f(\theta))(1 + a_3 * \kappa) \tag{7}$$

where $f(\theta)$ is the same as in Eq. (5) and z is the altitude. We used this approximation to estimate the water equivalent above 2700m altitude and used the value 2.5 for a_1 and 0.001 for a_2. The coefficient a_3 was set to 0.5. In the first row of Fig. 5 the initial approximation is shown.

5.3. Visual accuracy

To assess visual accuracy, we used real photographs of a Zermatt gondola station web camera and compared them to the results of our implementation [38]. Note that only the high level of detail was computed for this visualization as the area was small enough and no additional level of detail was needed.

In Fig. 5 we can see results of our approximation from 12.10.2015 to 21.04.2016 compared to photographs. The first row is on 12.10.2015 at the beginning of the simulation. The snow at the top of the mountains is approximated as described in Section 5.2. The second row is on 18.10.2015 after the first snowfall. We can see that the snow line was accurately simulated. The third row shows the results on 16.01.2016 with the scene fully covered in snow after heavy snowfall in early January. The last row shows the results on 21.04.2016 with snow beginning to melt as the temperature rises. Trees in our visualization were placed by hand and the city model integrated using OpenStreetMap data [39].

To achieve the results shown in Fig. 5, only the initial degree-day factor and the cell size had to be tweaked manually, which took around 10 minutes to find good values. For further steps during the computation, no additional manual tweaks were necessary.

5.4. Performance

We first show the performance of the snow cover approximation without LOD and then show the scalability using LOD. For all tests we used a desktop computer with an Intel i7-4770K CPU, 16GB of RAM and an NVIDIA GeForce GTX 960 GPU.

5.4.1. Snow cover approximation

To test the performance of the snow cover approximation without LOD we used the same scene as in Section 5.3. The dimensions

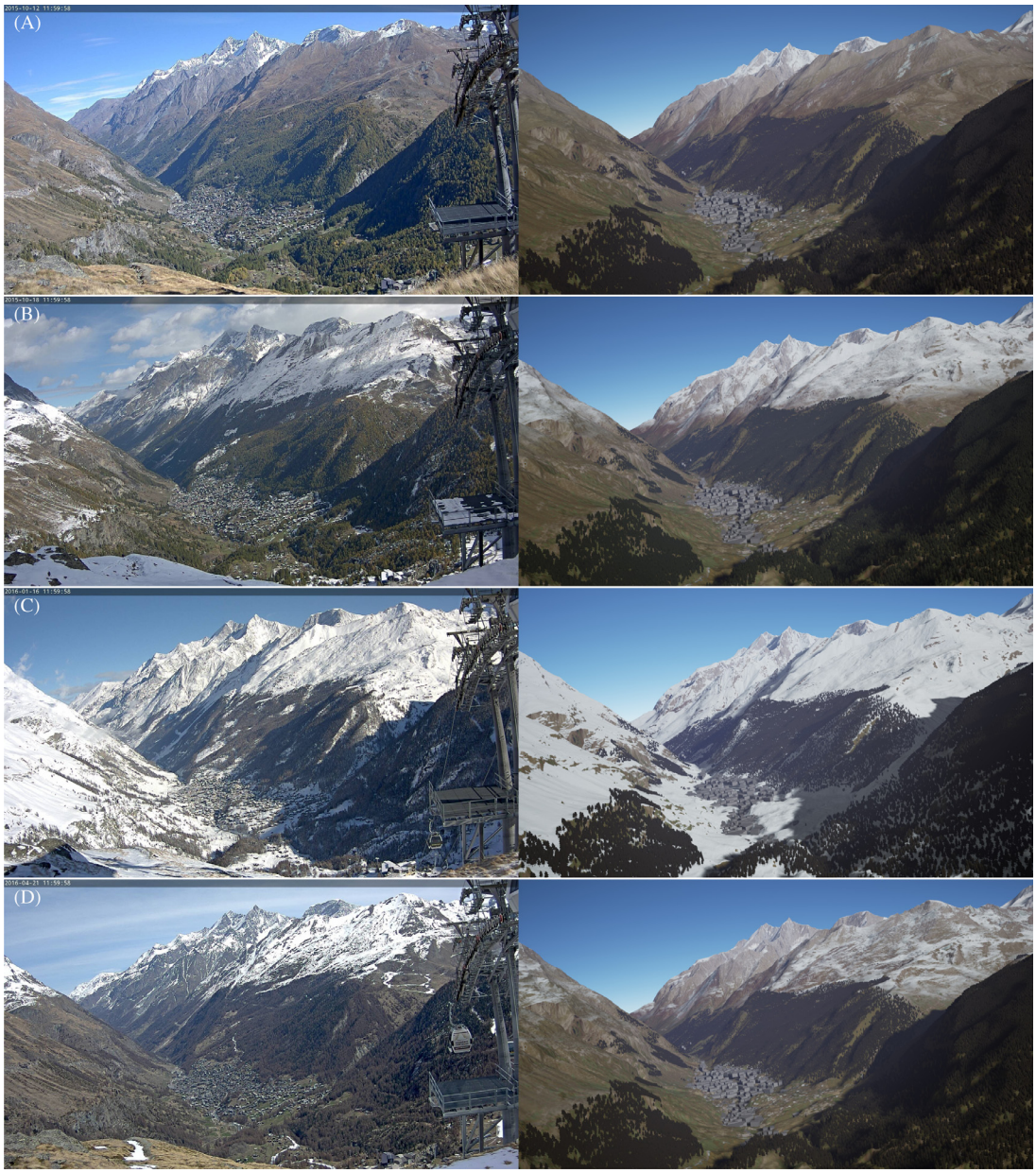

Fig. 5. Comparison of webcam images (left) of Zermatt with our simulation (right). Trees were placed by hand and the sun position was approximated from the webcam image. (A) Start of the computation on 12.10.2015 with only the initial snow visible. (B) The results on 18.10.2015 after the first snowfall. The snowline was accurately computed. (C) The results on 16.01.2016 after the first heavy snowfall with the whole scene covered in snow. (D) The results on 21.04.2016 with snow beginning to melt as the temperature rises. We can see the effect the albedo has on the snow shading as described in Section 4.2.

of the simulation grid are 222×206 and each cell covers an area of 4096 m^2.

In Fig. 6 the runtime compared to the number of steps (hours) per iteration is shown. We used an average of 10 iterations to calculate the values for each step size. The time was calculated by querying the GPU clock frequency and then querying the clock times before and after executing the compute shader, so only the GPU time is measured.

For large time steps, we can see an apparent performance gain compared to smaller time steps (computing 1000 steps per iteration is not 1000 times slower than computing one step 1000 times). This is because we only need to interpolate the water equivalent once instead of 1000 times. Also, we need to execute the compute shader less often to compute the same time span. Overall, we can see that the computation runs at interactive frame rates even for large step sizes per iteration and at real-time frame rates for smaller step sizes per iteration. Computing large time steps becomes relevant when the LOD is used as described in the next section.

5.4.2. Scalability

The computation scales to very large areas when using a level of detail approach as described in Section 4.1. For the scene in Fig. 3, the low level of detail (green) spans over 166 × 118 cells (53.56 × 38.26 km). Updating the grid took 0.2 ms on average. The high level of detail (red) spans over 166 × 118 cells (6.695 × 4.7825 km). The average time to update the high LOD when the camera remains constant was 0.19ms. When the camera moves, some cells of the high level of detail need to be recalculated from the beginning of the simulation, as they have never been

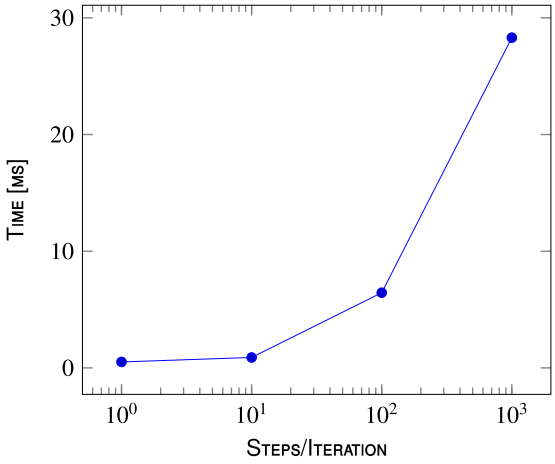

Fig. 6. Runtime for each iteration of the simulation with different step sizes (hours) per iteration. A step size of 1 means computing 1 hour and a step size of 1000 means computing 1000 hours in one iteration. Computing larger time steps (up to 1000 hours if in mid-season) becomes necessary for the high LOD when the camera is moving and new cells have to be computed up until the current time step.

calculated. In the worst case (when the camera moves very fast) this results in recalculating all cells. After a time-period of two months (1460 time steps) this resulted in an average 14.5 ms update time in our test scene, which is still considered interactive. Recalculating all cells is the worst case. When moving the camera slower, the cells that remain inside the high LOD only need to calculate 1 time-step. Without the described LOD strategy, the same scene required on average 18.9ms to update one time-step.

6. Conclusion and future work

We have shown how to implement a snow cover approximation with convincing results. The computation runs in real-time on the GPU and uses a degree-day approach. We run the computations with an hourly time step and use solar radiation approximations to compute snowmelt. To redistribute snow, an approximation based on terrain curvature is used. The performance of our computation is within interactive framerates even for large time frames (>1000 hours). The results compared with real photographs show that the approximation provides visually accurate results. The snow line remains accurate even for late winter.

Improvements for the snow cover approximation could be made by adding a wind simulation. The interpolation scheme we currently use does not account for large scale snow redistribution effects caused by wind. But this would also add the need for accurate wind input data and drastically reduce the performance, as large-scale wind simulations such as those described in [40] still lack in performance. The snowmelt computation could be improved by not only calculating the energy index based on the slope and aspect of the cell, but also include shadowing effects from the neighboring terrain. Further we could modify the calculation grid by using a dynamic tree structure to create larger grid cells where the terrain does not change much and therefore save memory and possibly gain performance. This would also remove the need to tweak the grid size parameter by hand at the beginning of the computation.

Another level of detail could be added when the whole earth's snow cover should be approximated. This level of detail could use crude approximations of snow lines for a given longitude and latitude without considering rainfall data and only for closer views our described approximation could calculate the snow cover with finer detail.

Additionally, if sufficient data were available our approximation could be tested by not only comparing it visually to photographs but to real snow height data as well. Right now, we are mostly concerned with the visual accuracy but if our approximation were to include a wind simulation for large scale snow redistribution effects comparing the results to measured data would be beneficial.

Finally, visualization improvements could be made by including falling snow, such as suggested by Langer et al. [41], or dynamically moving snow masses as described by Stomakhin et al. [42], for example to render effects caused by avalanches or plowing streets. Adding foliage, trees or other 3D models at appropriate places such as described by Amara et al. [43] would be another enhancement. They use GIS land cover data to render large amounts of trees in real-time. If no land cover data is available the aerial imagery could be classified in a pre-processing step as described by Huang et al. [44]. They used support vector machines to accurately predict land covers from aerial imagery.

Acknowledgements

Thanks to all the reviewers who have helped with their comments continually improve this paper. We also thank Taisha Waeny for proofreading the paper.

References

[1] Hoppe H. Smooth view-dependent level-of-detail control and its application to terrain rendering. In: Proceedings of the visualization'98.. IEEE; 1998. p. 35–42.
[2] Losasso F, Hoppe H. Geometry clipmaps: terrain rendering using nested regular grids. In: Proceedings of the ACM transactions on graphics (TOG), 23. ACM; 2004. p. 769–76.
[3] Asirvatham A, Hoppe H. Terrain rendering using gpu-based geometry clipmaps. GPU Gems 2005;2(2):27–46.
[4] Premože S, Thompson WB, Shirley P. Geospecific rendering of alpine terrain. In: Rendering techniques? 99. Springer; 1999. p. 107–18.
[5] Foldes D, Benes B. Occlusion-based snow accumulation simulation.. In: Proceedings of the vriphys. Citeseer; 2007. p. 35–41.
[6] Fearing P. Computer modelling of fallen snow. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co.; 2000. p. 37–46.
[7] Haglund H, Andersson M, Hast A. Snow accumulation in real-time. In: Proceedings of the SIGRAD, 2002; 2002. p. 11–15.
[8] Feldman BE, O'Brien JF. Modeling the accumulation of wind-driven snow. In: Proceedings of the ACM SIGGRAPH 2002 conference abstracts and applications. ACM; 2002 218-218.
[9] Ohlsson P, Seipel S. Real-time rendering of accumulated snow. In: Proceedings of the Sigrad Conference; 2004. p. 25–32.
[10] Moeslund TB, Madsen CB, Aagaard M, Lerche D. Modeling falling and accumulating snow. In: Proceedings of the second international conference on vision, video and GraphicGraphics. Citeseer; 2005.
[11] Wang C, Wang Z, Xia T, Peng Q. Real-time snowing simulation. Vis Comput 2006;22(5):315–23.
[12] Saltvik I, Elster AC, Nagel HR. Parallel methods for real-time visualization of snow. In: Applied parallel computing. state of the art in scientific computing. Springer; 2007. p. 218–27.
[13] Hinks T, Museth K. Wind-driven snow buildup using a level set approach. In: Proceedings of the eurographics Ireland workshop series, 9; 2009. p. 19–26.
[14] Festenberg Nv, Gumhold S. A geometric algorithm for snow distribution in virtual scenes. In: Proceedings of the eurographics workshop on natural phenomena. The Eurographics Association; 2009. p. 15–25.
[15] Reynolds DT, Laycock SD, Day A. Real-time accumulation of occlusion-based snow. Vis. Comput. 2015;31(5):689–700.
[16] Blöschl G, Kirnbauer R, Gutknecht D. Distributed snowmelt simulations in an alpine catchment: 1. model evaluation on the basis of snow cover patterns. Water Resour. Res. 1991;27(12):3171–9.
[17] Swift LW. Algorithm for solar radiation on mountain slopes. Water Resour. Res. 1976;12(1):108–12.
[18] Onoue K, Nishita T. An interactive deformation system for granular material. In: Proceedings of the computer graphics forum, 24. Wiley Online Library; 2005. p. 51–60.
[19] Fedkiw R, Stam J, Jensen HW. Visual simulation of smoke. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques. ACM; 2001. p. 15–22.
[20] Muraoka K, Chiba N. Visual simulation of snowfall, snow cover and snowmelt. In: Proceedings of the seventh international conference on parallel and distributed systems: workshops. IEEE; 2000. p. 187–94.
[21] Barringer J. A variable lapse rate snowline model for the remarkables, central Otago, New zealand. J Hydrol(NZ) 1989;28(1):32–46.

[22] Rolland C. Spatial and seasonal variations of air temperature lapse rates in alpine regions. J Clim 2003;16(7):1032–46.
[23] Rango A, Martinec J. Revisiting the degree-day method for snowmelt computations1. JAWRA J Am Water Resour Assoc 1995;31(4):657–69.
[24] Liston GE. Local advection of momentum, heat, and moisture during the melt of patchy snow covers. J Appl Meteorol 1995;34(7):1705–15.
[25] Liston GE, Elder K. A distributed snow-evolution modeling system (snowmodel). J Hydrometeorol 2006;7(6):1259–76.
[26] Hock R. A distributed temperature-index ice-and snowmelt model including potential direct solar radiation. J Glaciol 1999;45(149):101–11.
[27] Monteith J. Light distribution and photosynthesis in field crops. Annals Botany 1965;29(1):17–37.
[28] Eggleston K.O., Israelsen E.K., Riley J.P. Hybrid computer simulation of the accumulation and melt processes in a snowpack 1971 Reports. Paper 501.
[29] Maréchal N, Guérin E, Galin E, Mérillou S, Mérillou N. Heat transfer simulation for modeling realistic winter sceneries. In: Proceedings of the computer graphics forum, 29. Wiley Online Library; 2010. p. 449–58.
[30] Zevenbergen LW, Thorne CR. Quantitative analysis of land surface topography. Earth Surf Process Landf 1987;12(1):47–56.
[31] Trenholme D, Smith SP. Computer game engines for developing first-person virtual environments. Virtual Real 2008;12(3):181–7.
[32] Trabant D, Armstrong R, McClung D. Snow avalanches. In: Proceedings of the cold regions hydrology and hydraulics. ASCE; 1990. p. 147–76.
[33] Cook RL. Shade trees. In: Proceedings of the ACM Siggraph computer graphics, 18(3); 1984. p. 223–31.
[34] Barré-Brisebois C. Deformable snow rendering in batman?: Arkham origins. GDC; 2014.
[35] Yanyun C, Sun H, Hui L, Wu E. Modelling and rendering of snowy natural scenery using multi-mapping techniques. Comput Anim Virtual Worlds 2003;14(1):21–30.
[36] Federal Office of Topography. swisstopo Onlineshop. 2016. https://shop.swisstopo.admin.ch/; [Online; accessed 28-October-2016].
[37] Adam S, Pietroniro A, Brugman MM. Glacier snow line mapping using ers-1 sar imagery. Remote Sens Environ 1997;61(1):46–54.
[38] Zermatt Bergbahnen A.G.. Cam-Bild Aroleid. http://www.matterhornparadise.ch/de/wetter/webcams; 2016. [Online; accessed 28-October-2016].
[39] OpenStreetMap. https://www.openstreetmap.org; 2017. [Online; accessed 28-March-2017].
[40] Schmid T.M.. Real-time snow simulation-integrating weather data and cloud rendering. Master's thesis; NTNU2016.
[41] Langer MS, Zhang L, Klein AW, Bhatia A, Pereira J, Rekhi D. A spectral-particle hybrid method for rendering falling snow.. Render Tech 2004;4:217–26.
[42] Stomakhin A, Schroeder C, Chai L, Teran J, Selle A. A material point method for snow simulation. ACM Trans Gr (TOG) 2013;32(4):102.
[43] Amara Y, Meunier S, Marsault X. A gpu framework for the visualization and on-the-fly amplification of real terrains. In: Proceedings of the international symposium on visual computing. Springer; 2007. p. 586–97.
[44] Huang C, Davis L, Townshend J. An assessment of support vector machines for land cover classification. Int J Rem Sens 2002;23(4):725–49.