A Naïve Approach to Snow Simulation

Kiran Wattamwar (kiranw) and Maria Karelina (maschka)

6.837 – Introduction to Computer Graphics – Final Project 2015

ABSTRACT

Our motivation for conducting this project was the recent popularity of Disney's Frozen. We were interested in modeling physical systems and wanted to create an optimal method for modeling a system of several snow particles efficiently, and implementing several techniques we found in recent papers on snow simulation. We specifically wanted to model snow accumulation in physical systems over time.

BACKGROUND

In this paper, we present a simplified approach to snow simulation based on various reference papers. It includes a particle system to model falling snow particles, while also updating a Cartesian grid to handling accumulation over time. This method reduces the total number of particles being stored in the system while increasing efficiency greatly.

1 | Particle System

PARTICLE GENERATION

Our particle generator operates such that it stores a constant number of particles once that constant has been reached. Upon initialization, a fixed number of particles are created. As time increases, the number of live particles increases and stays constant after some time. We found that 1000 is a reasonable value for this. Particles are managed in our particle class and store several properties. These include their positions in space, current lifespans, acceleration due to gravity, velocity, damping force, color, and an indicator of if they are alive or not. We use these member variables in various ways – positions are used for updating, with an integrator that steps forward in time using gravity, velocity and damping force to determine future velocities, and positions of the snowflakes. The boolean "alive" allows us to control the growth of visible particles in the system upon initialization. Though in most figures in this paper, our snow is white, in some simulations we varied the color of the particles with time to show where snow has accumulated. If a particle exceeds its lifespan, or intersects with the ground at its highest point of accumulation, the particle is inactivated and a new particle is initialized in the next time step. For this reason, the birth rate is dependent on how quickly intersections happen with the scene.

Particles are modeled using OpenGl primitive spheres. These can be generated quickly, with minimal overhead on our end.

TIME INTEGRATION

Snow is updated at each time step using a simple integration function. We consider acceleration due to gravity in this calculation, along with a damping coefficient to determine the adjusted position of the particle. Position is updated at each time step using the stored particle's velocity, scaled by the drag force. Velocities are updated using acceleration (gravity). Because we only worry about the motion of the particles in each time integration step, we can eliminate any motion associated with the ground mesh in this step for efficient calculation on a constant number of particles at each time.

2 | Cartesian Accumulation Grid

GRID IMPLEMENTATION

The grid itself is handled by two major components. The first is an array storing the highest vertices (along the y-axis) that compose the ground, indexed by their x-z coordinates, and storing their accumulation heights. The second is an array storing colors at those same x-z coordinates. On every drawing of the scene, we

couple these components together and render vertices based on this information.

COLLISION DETECTION

Collision detection in this system is relatively simple – we are able to detect it by comparing the y-position of a particle by the ground grid's height at the same x-z coordinate pair. Computing the difference between these heights becomes an efficient constant time calculation. If the particle is below the grid's surface, we terminate the particle's life and integrate that particle into the grid for accumulation. As we did on previous problem sets, we also include an `EPSILON` value to account for floating-point errors.

ACCUMULATION AFTER COLLISIONS

On the event of a collision between a particle and the floor grid, we first increase the height of the grid at the particle collision's x-z coordinate corresponding to the grid. We simply index into the mesh's coordinate array and update this by a small value, tuned iteratively until reasonable. We also update the grid's color at this point by adding another small value to its `(r,g,b,a)` values, and taking the minimum between this updated value and 1. In our current system, a point on the grid must endure 4 collisions with snow particles until it turns completely white. This creates a realistic effect that makes the magnitude of accumulation clear between sparse and dense areas.

3 | Wind

WIND AND INTERFACE

When the output window is open, wind and be controlled using the "w" key. This toggles it on and off.

WIND AND PARTICLES

The effect of wind on particles could be implemented by adjusting the physical properties of the system (namely, its velocity and acceleration, components in the direction of the wind). This could be modeled by adjusting the stored velocity in each particle or updating components in the wind's direction during time integration.

We found that a much simpler way to recreate the same effect with minimal overhead was to jitter the position of the particles by some factor in the relevant components using the direction of the wind. This means we need to access the particle's variables less and can add a layer of acute randomization to the system for a more realistic appearance.

WIND AND EXISTING ACCUMULATION

The effect of wind on accumulation also needs to be accounted for, as we expect wind to have a subtle effect on any collected snow, particularly if there are peaks. We expected this behavior to smoothen the peaks such that any harsh slopes are eliminated, and we see only steep inclines against the wind, and smooth declines extended over lengths that are higher in magnitude. We implement this by evaluated the derivative between all vertex pairs (in the direction of the wind). If the slope exceeds a reasonable threshold, we transfer a small amount of that snow to the closest vertex in the projection of the wind from the current vertex. We also adjust color using similar logic to account for small increases in the build up of snow.

For new accumulation, we do not need to change anything about updates. Since we run a wind-based smoothing function on every update, new accumulations are also controlled and cannot form unnatural peaks for the same reason our existing accumulation is softened over time with the wind.

4 | Realism

STABILIZATION

After snow was accumulated, a stabilization function was applied. This function would make sure that the snow wouldn't accumulate unrealistic peaks. Like many granular materials, snow has a certain threshold at which it can be accumulated vertically before the snow starts rolling or sliding off the peak. The maximum slope at which the limit is found within a granular

material, is called an angle of repose (AOR). We chose a threshold AOR value to create a more realistic accumulation of snow. In theory this simulation can change the AOR value. Depending on it's wetness, the snow become more stable, and this could be modeled by changing this value. In principle, a simulation in which the AOR is set to 0 would come close to simulating the accumulation of water. For our simulations, we picked a threshold that seemed most realistic for snow.

ELEMENTS OF RANDOMIZATION

We found that including several sources of randomization provided a much more realistic representation of snow in the simulation. In ours, we account for randomization at almost every level of the code.

We first introduce it by randomizing where particles are initialized in the x-z plane upon construction of a particle type. On each time step, we also randomly choose the radius of each particle – we do not preserve this radius for every unique particle. We decided this because our OpenGl sphere primitives are radially symmetric, and if rotated, do not change in appearance. Real snowflakes rotate as they fall because of air resistance and wind – to mimic this, we found that jittering the size of particles gives us a similar desired effect.

FLURRY SIMULATION

One behavior of the snow that we wanted to mimic was their twirling motion path as they fell. We considered several approaches to this, including position based sinusoidal functions to create spiral-like paths in the air. Though this is also extremely feasible, after implementation this seemed too controlled. We now use a jitter in the x and y components during integration that adds some randomized value within a reasonable range to the previous x-z positions of each particle. We can explain the real event of this by the air resistance acting on snow flakes, amplified by any trace of wind, but because we wanted to simplify our model, we were able to achieve this with some randomization and fine tuning of thresholds.

TERRAIN

In order to create a more interesting and realistic environment than a flat plane we decided to make a module in which terrain was generated. The terrain was generated using Perlin noise for both height and color. The Perlin noise is used to interpolate between to colors. A different set of Perlin octaves is used to create the terrain such that altitude of hills isn't' directly correlated with height.

OTHER PARAMETERS

Simpler and more intuitive additions like ambient and diffuse lighting, along with normal interpolation for a smoother looking mesh allow us to also achieve realism. In addition, a backdrop was added to the scene which adds a little more realism.

5 | Results

We were able to create a fairly convincing snow accumulation simulation. We made recordings of some variants of these simulations.

The simple terrain with minimal snow can be seen in figure 1. This shows some nice rolling hills, with some soft gradients in color. The terrain accumulates snow fairly quickly. The differences in snow aggregation can be seen with or without stabilization in *figure 2a* and *2b*. Clearly the valley in the terrain fills up more evenly with stabilization, which is expected. Smoothing however, is a fairly computationally heavy. OpenGL has a much higher frame rate without the stabilization function activated. The difference is recorded as well.

With wind, which can be toggled, the snow aggregation also creates some smoothing in the mesh, maybe a little too aggressively. This is a downside that hasn't been completely dealt with. However, past initial conditions, snow aggregates fairly realistically.

6 | Conclusion and Future Work

In general this simulation seems to have been somewhat realistic. In addition, OpenGL seems to be able to render the images in adequate time, making this simulation easy to reproduce for future animations. There are many aspects that can be developed further and improved. Even though, some may make computation time longer. It would be interesting to see how this technique would work with a larger Euclidean mesh, with a larger amount of triangles. Perhaps, it would allow for integration of interesting objects and meshes with more details. Another area of exploration is the shape of the falling snow particles, such that they are modeled with less symmetrical shapes, or clusters.

7 | Data and Graphics

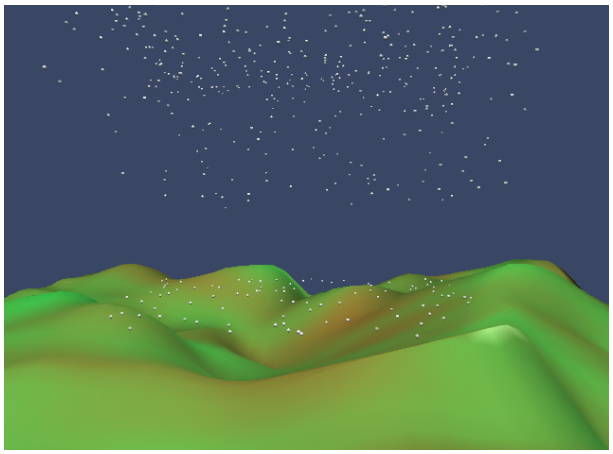

figure 1 – Minimal snow accumulation

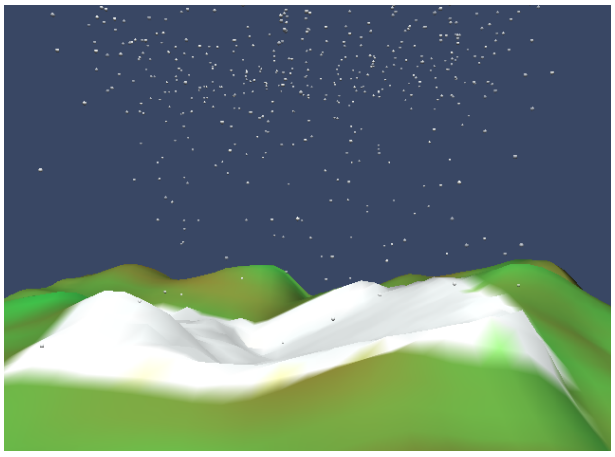

figure 2a – Short term snow aggregation over time with smoothing

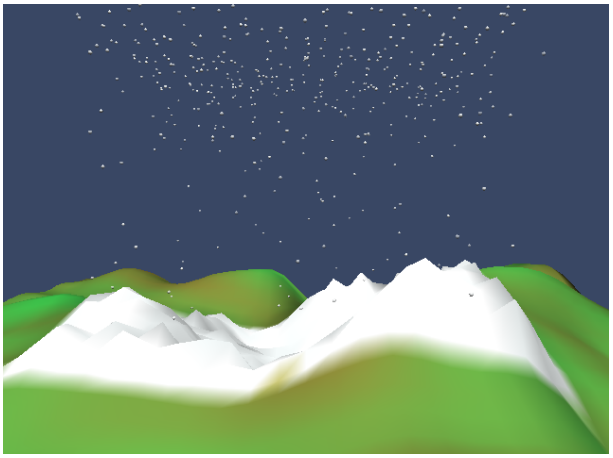

figure 2b – Short term snow aggregation without stabilization

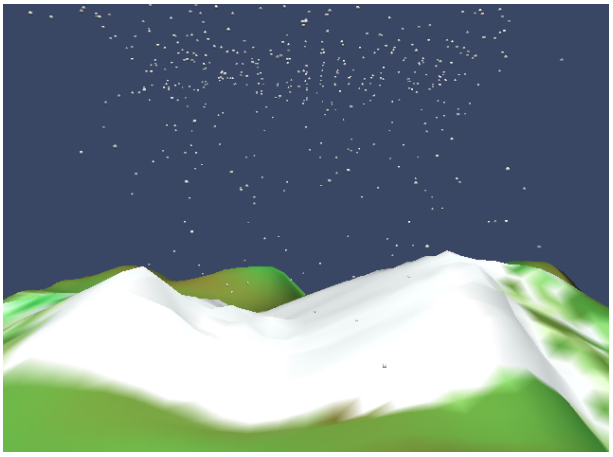

figure 3 – Long term snow accumulation with smoothing

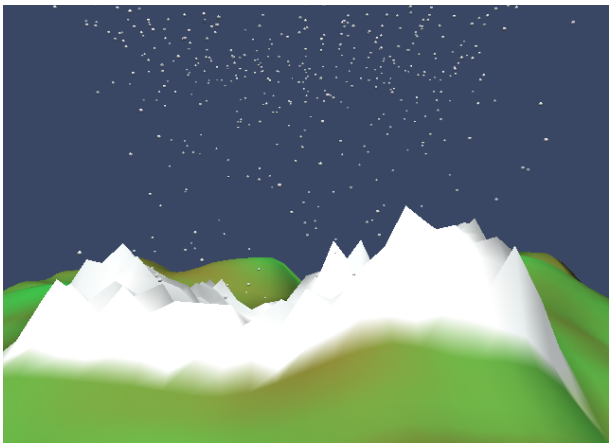

figure 4 – Long term snow accumulation without stabilization forms sharp uncharacteristic peaks

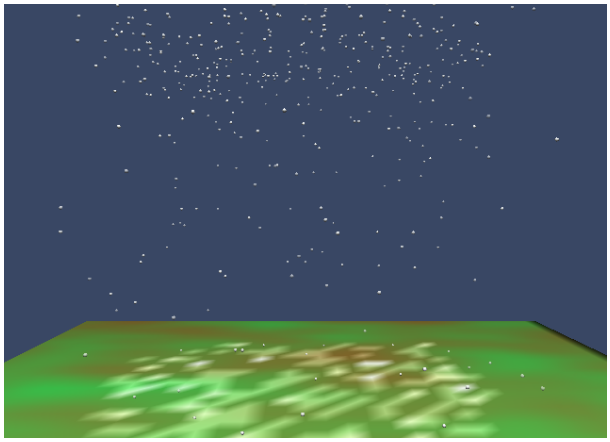

figure 5 – Minimal snow accumulation on a flat topology

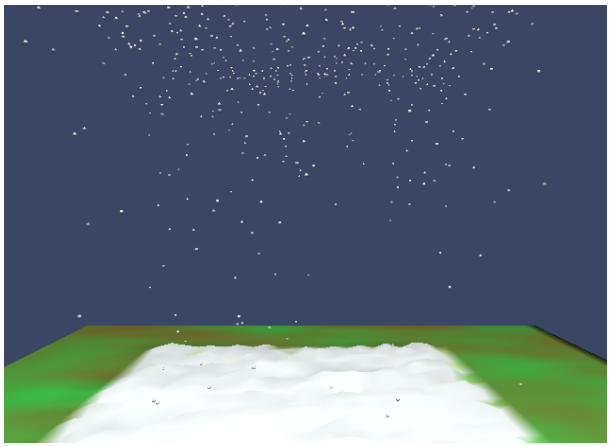

figure 6 – Snow aggregation on a flat topology

figure 7 – Short term accumulation in the presence of wind on a flat surface

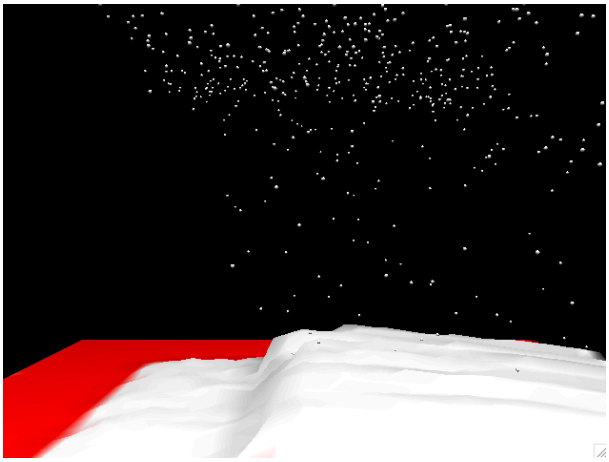

figure 8 – Long term accumulation in the presence of wind on a flat surface

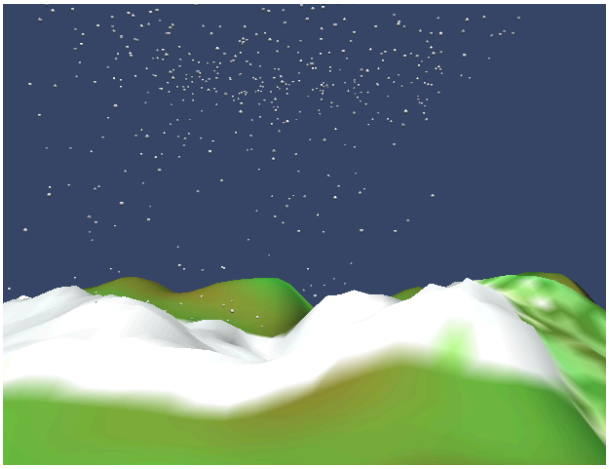

figure 9 – Short term accumulation in the presence of wind on a terrain generated by Perlin noise (wind moving left)

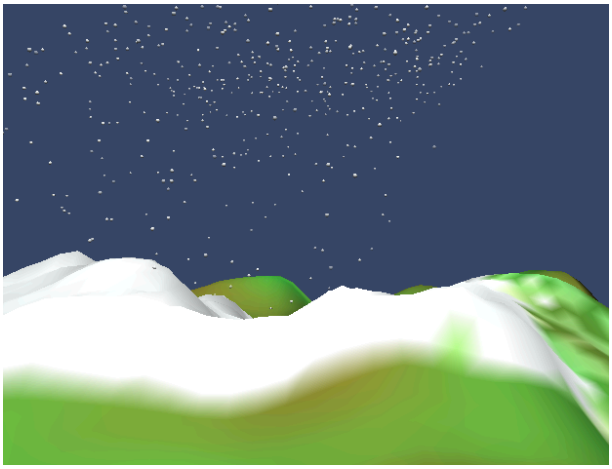

figure 10 – Long term accumulation with wind on the same terrain

8 | References

[1] https://www.math.ucla.edu/~jteran/papers/SSCTS13.pdf
[2] http://graphics.berkeley.edu/papers/Feldman-MAW-2002-07/Feldman-MAW-2002-07.pdf
[3] http://www.create.aau.dk/Snow/snow.pdf
[4] https://graphics.stanford.edu/courses/cs448-01-spring/papers/fearing.pdf