Snow Accumulation in Interactive Time

Chris Willmore Scott Fermeglia

Advanced Computer Graphics
Rensselaer Polytechnic Institute

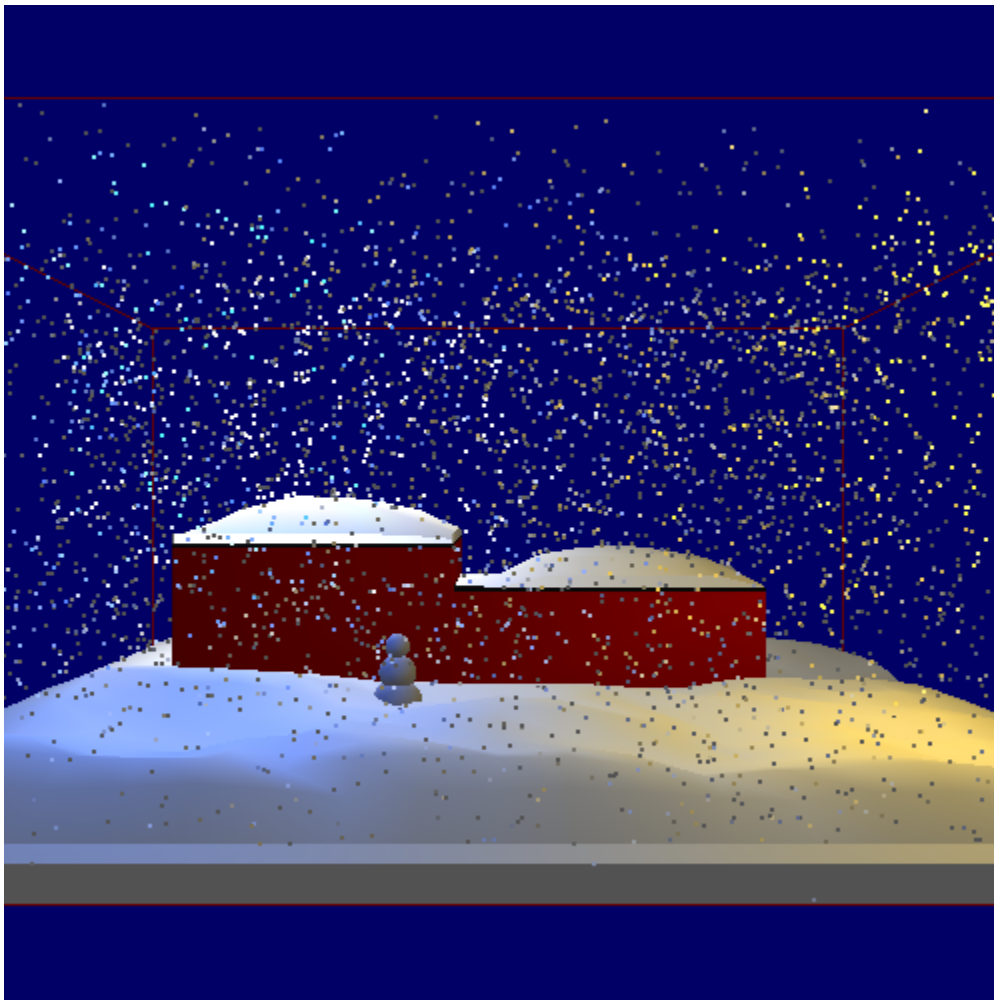

Figure 1 - Test Scene with Wind Blowing Left to Right

Abstract

Falling snow is a major part of many weather patterns, and its accumulation on the ground can greatly change the layout of a given scene. We present an algorithm which simulates snowfall under the influence of wind, as well as the accumulation of snow on the ground. We also simulate the redistribution of snow, both locally and from objects at different heights. The algorithm runs in interactive time.

1. Introduction

Snowy scenes are a common sight in reality, but instances of computer-generated snow are somewhat rarer. Of the instances in which falling snow is generated, many simply use "billboard" effects. These give a somewhat convincing snowfall pattern and avoid rendering snowflake objects in a 3D space. Accumulation of snow is even less well represented; most snowy scenes are simply static artist-created meshes which do not change with time.

Our project has been to develop an algorithm which can render convincing snow in a 3D space, and to simulate snow accumulation on objects in a fairly realistic manner. This is done by assigning physical properties to snowflakes (mass, forces applied, etc.), allowing them to be affected by gravity and wind, and creating a method by which they can impact the ground and other objects and be "converted" from a free-falling flake into snow on the ground. Ground-cover snow is represented by a height field (or, in the case of objects in the scene, multiple height fields).

The ground-cover snow also has physically-based behavior – if any part of the height field becomes too high in relation to its neighbors, it will redistribute its mass to its neighbors. If the height field is on an object, and snow is transferred off the side of said object, the algorithm can redistribute this snow to height fields below it. Overall, we have found that this algorithm is a fairly convincing simulation of snowfall and accumulation, and runs well on our computers.

2. Related Work

Perhaps the most advanced work on physically based wind-driven snowfall and its attendant accumulation has been by Ingar Saltvik in his Master's thesis, "Parallel Methods for Real-Time Visualization of Snow" [1]. His work defines snowflakes in great detail, including an aerodynamic model. Wind is maintained with incompressible Navier-Stokes, while accumulation occurs as a result of the rendered snowflakes. The algorithm does not, however, provide for local redistribution of snow on the ground, which can lead to "spikes" of snow and other discontinuities.

Moeslund, et al's "Modeling Falling and Accumulating Snow" [2] takes a different approach, rendering snowflakes as small randomized agglomerations of triangular polygons, with the polygon count determined by the consistency of the snow (i.e. wet and dry). The shape of the created snowflakes is taken into account in the interaction of these snowflakes with wind; the result is snowflakes that move and tumble quite differently from one another

based on shape. The authors also model accumulation, but have chosen to keep the snowflake and accumulation models separate; this allows for the generation of snow-covered scenes of any depth very rapidly. While the paths snowflakes would take are calculated quickly and this model used to generate the ground cover, rendered snowflakes are not used as the mechanism of snow accumulation.

Haglund et al's "Snow accumulation in real-time" [3], while it does not render snowflakes, focuses on a different but important aspect of accumulated snow – its realistic appearance. The authors took note of patterns in their "randomized" accumulation algorithm and showed a method for correcting this. The paper also devoted much time to rendering physically correct (or at least visually appealing) snow cover "edges" (i.e. the edge of a snowbank on top of an object).

Our algorithm seeks to combine three of what we feel are the most important aspects of a rendered snowfall scene – physically based wind-driven snowfall, accumulation of snow based on the rendered snowflakes and their attendant properties, and the redistribution of snow on the ground to produce a fairly convincing, realistic covering of snow on a landscape. While the other mentioned algorithm have superior implementations of some portion of these, none we have found has implemented all three together, and so we feel a contribution can be made.

3. Theory

A number of the parameters in our algorithm are physically based or inspired, notably the mass of snowflakes, effect of wind and drag, and redistribution of snow on the ground to maintain a smooth surface.

From [4], it can be found that snowflakes can range in diameter from .001m to .0762m, with a common value of .0254m (it should be noted that the .0762m snowflake was the largest ever recorded, and should be statistically ignored). From [5], snowflakes have a density between .005 and .2 grams per meter-squared, based on formation and temperature. The result is a snowflake mass between .000005g and 88.5g, with a common mass of about 1.64g. In our algorithm, we decided to generate snowflake masses between .005g and 5g, a weighting which is well within the established range of masses, and which produces a visually appealing effect.

Once the weight of the snowflake is established, we simulate its motion through solution of the force equation

$$F = mg + k(v - v_{\text{wind}})$$

through simple Euler integration, where *g* is the vector representing gravity and *k* is a drag coefficient (which can be calculated using the Reynolds number of air and the radius and drag of snowflakes, but in practice was tweaked by hand). We considered further randomizing the motion of a snowflake by perturbing its position according to a random walk at each timestep, but we found that the random mass and initial position assigned to each snowflake was sufficient to give the aggregate snowfall a random appearance.

In order to enforce non-spikiness on the height fields, we enforce an upper limit on

the absolute value of the Laplacian of the snow surface, defined by

$$L = \frac{d^2h}{dx^2} + \frac{d^2h}{dy^2}$$

where *h* is the height of the snow at a given point and *x* and *y* are the local coordinates of the height field. The Laplacian in our case is approximated by considering a point on the lattice and its four neighbors, and approximating it as

$$L \approx (4h_{i,j} - h_{i+1,j} - h_{i-1,j} - h_{i,j+1} - h_{i,j-1})/dx^2$$

If the Laplacian is found to exceed the limit, the snow in that neighborhood is redistributed so that the limit is again obeyed, at least locally.

If too much snow accumulates on the edge of a height field (i.e., the height of the height field at that point exceeds a programmed-in limit), snow is emitted into the scene according to how much snow falls off the edge, and the height of the snow at that point is decreased accordingly.

4. Algorithm

The algorithm for simulating snow accumulation may be summarized in three steps which occur in an infinite loop:

- For each snowflake: if it's active, increment its position according to the physical model described above; if it's inactive, make it active with some low probability.
- Deactivate snowflakes which have fallen outside the world's bounds.
- Detect collisions between every possible snowflake-object pair, possibly altering the object or deactivating the snowflake.
- Redistribute/spill the snow in the height fields.

The action of each object upon detecting a flake collision is different:

- If a collision with a height field is detected, the height field is incremented at the point of the collision by an amount determined by the sampling frequency of the height field, the mass of the snowflake, and the density of the snow.
- If the collision with a box is detected, the snowflake is relocated to the nearest point on the outside of the box. This has the effect that snowflakes that run into boxes have the appearance of sliding down the side of the box; this is handy in that it naturally creates snowdrifts at the base of windward sides of boxes on the groun.d
- If the collision with a sphere is detected, the snowflake is simply eliminated; however, there's no reason we couldn't do the same sort of treatment as with the box. However, in our simulation, the only object that used spheres (the snowman) served as a useful gauge of how deep the snow was at that point, and it would have been detrimental if it accumulated snow as well.

The redistribution/spill step is only carried out once for each iteration of the model, but the redistribution step seems to suffer from instability if the amount of snow that falls onto a particular area of the height field exceeds the ability for the Laplacian cap to curb spikiness at that point, and the simulation blows up. We may benefit from running multiple smoothing steps per single time step, but we have not yet tried this.

5. Results

All pictures for this report were generated on an IBM Thinkpad T43p with 2G of RAM, a 2GHz Pentium M processor, and an ATI MOBILITY FireGL V3200 video card. Most simulations were run on a Apple MacBook running Mac OS X, with a 2.16Ghz Intel Core 2 Duo Processor and 2G of RAM, and an Intel GMA 950 graphics card with 64M of VRAM. Code was developed for cross-platform compatibility. Figure 2 shows the test scene we have developed for our algorithm to run on. Note the locations of the three light sources – white above the house, blue closer to the camera from that, and yellow on the right. Figure 3 shows the effect of the redistribution algorithm on the smoothness of the ground-covering snow. Figure 5 is a larger version of Figure 1, and shows the effects of wind. Snowflakes are generated on the sides of the bounding box. Note how snow has accumulated on the roof of the house and garage, but that the accumulation is less on the left side of the garage, as snow is blowing from left to right across the scene. Figure 6 shows the result of a particularly heavy snowfall using only the scene's inherent wind; note the local variations in snow height due to building location and random generation of flakes in the algorithm.

The algorithm runs in interactive time on the laptop computers mentioned above. This has been found to be true for scenes including as many as 50,000 separate snowflakes; the program can probably run at interactive rates for many more snowflakes, though this has not been tested.

6. Discussion

While our algorithm does a good job of simulating snowfall, accumulation, and redistribution, it is not as complete or robust as the algorithms cited above ([1], [2], [3]). The wind simulation we use is a simple global vector which is allowed to change by small random amounts. It is not a Navier-Stokes solution to wind; for example, buildings do not block wind movement, so snowflakes will never move upwards after being carried on wind that has hit a building. Height fields are not affected by wind, and so a strong wind cannot blow snow off of an object.

Currently, snow only accumulates on flat surfaces (the snowman, being made of spheres, never accumulates any extra snow). While we feel it is possible to extend our height field implementation to curved surfaces, it would be somewhat difficult, and require dense fields on these surfaces, leading to a slowdown of the algorithm.

The height fields in our algorithm have an "apron" of quads which keeps the viewer from seeing beneath the height field (A close look at the right image of Figure 3 shows what our algorithm look like without the apron). This is wholly unrealistic, and was implemented more for the visual effect than for any real basis in reality. A

implementation as in [3] would solve this problem.

Our algorithm, while it works well for snowflakes of appropriate size, fails (quite spectacularly, see Figure 4) if the snowflakes are too massive, or if there is a huge concentration of flakes hitting the same area of a height field in quick succession. This is essentially a problem with the redistribution taking place over multiple time steps; it is possible for a field to grow quickly enough that it consumes additional snowflakes, causing a runaway growth. This could be fixed by allowing the redistribution subroutine to be called multiple times per timestep.

Finally, the flakes themselves are represented as points and their masses as point masses. While a decent enough approximation, it is not realistic. With enough time, an implementation as per [2], or better yet, [1] (if combined with a Navier-Stokes wind representation) would yield physically realistic snowflakes, or at least be closer to this ideal.

7. Conclusions

We have presented an algorithm for wind-driven snowfall which accumulates based on the rendered snowflakes, and which allows redistribution of fallen snow to maintain smoothness. Our algorithm runs in interactive time, and produces fairly convincing renderings of snowfall and accumulation. The algorithm also includes lighting and material properties for objects in the scene, giving a framework for potential expansion of the algorithm to larger test scenes. Overall, we are quite pleased with the results, particularly given the time constraints under which the project was completed.

8. Future Work

There are many directions in which this project could be taken. In particular, there are some goals that we had for the project that had to be abandoned due to time constraints, including:

- localized wind, including solution of the Navier-Stokes equations
- wind affecting snow drifts directly (carving out channels, blowing over drifts)
- moving objects
- adding temperature to affect the wetness of snow or melting of snow on differently heated surfaces
- extension of snow accumulation to non-flat objects
- optimizing the algorithm for real-time performance
- snow sticking to surfaces with non-vertical normals (i.e., not facing up)

In particular, there is quite a bit of infrastructure in the code for optimizing object-flake collision, in the form of dividing the simulation environment into several cells which index the space, and calculating in which cells the flakes and objects lie.

We also considered generalizing snow accumulation to arbitrary meshes instead of height fields; the idea is that if a snow particle lands on an arbitrary mesh, the mesh is made to bulge slightly in the direction of the normal at that point on the surface, and over-stretched triangles in the mesh are subdivided as appropriate. This could be extended even further to create a

sort of "snow gun," which could be used to model round-edged objects. However, again, we ran out of time for this idea.

References

1. Ingar Saltvik. “Parallel Methods for Real-Time Visualization of Snow,” 2006. Master’s Thesis, Norwegian University of Science and Technology, Department of Computer and Information Science

2. T.B. Moeslund, C.B. Madsen, M. Aagaard, and D. LercheModelling. “Modeling Falling and Accumulating Snow,” 2005. The Eurographics Association

3. Hakan Haglund, Mattias Andersson, and Anders Hast. “Snow accumulation in real-time.” In *SIGRAD2002, The Annual SIGRAD Conference. Special Theme – Special Effects and Rendering, November 28–29, 2002, Norrkoping, Sweden*, 2002

4.http://hypertextbook.com/facts/BrigidNaughton.shtml

5. Rasmussen, Roy M.; Vivekanandan, Jothiram; Cole, Jeffrey; Myers, Barry; Masters, Charles, "The Estimation of Snowfall Rate Using Visibility", 1999, Journal of Applied Meteorology, vol. 38, Issue 10, pp.1542-1563

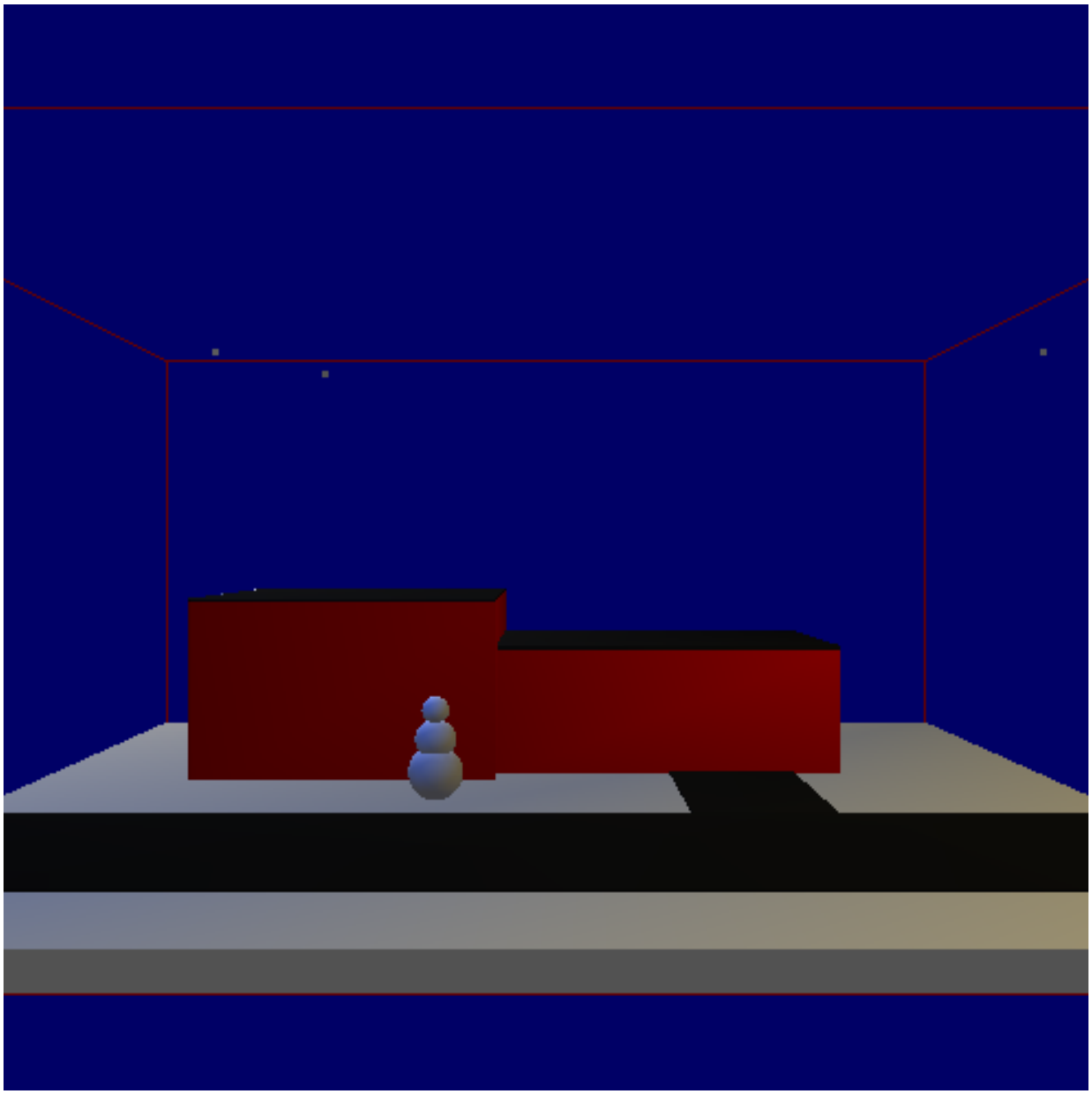

Figure 2 - Our Test Scene

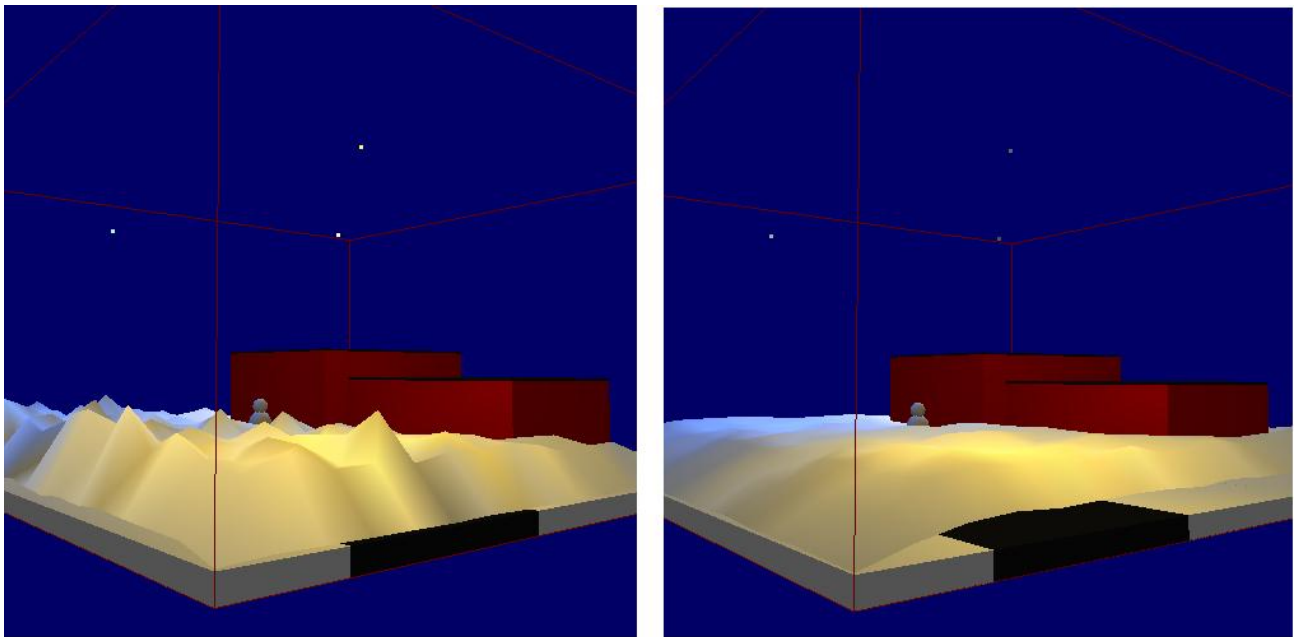

Figure 3 - Our Algorithm Without and With Redistribution

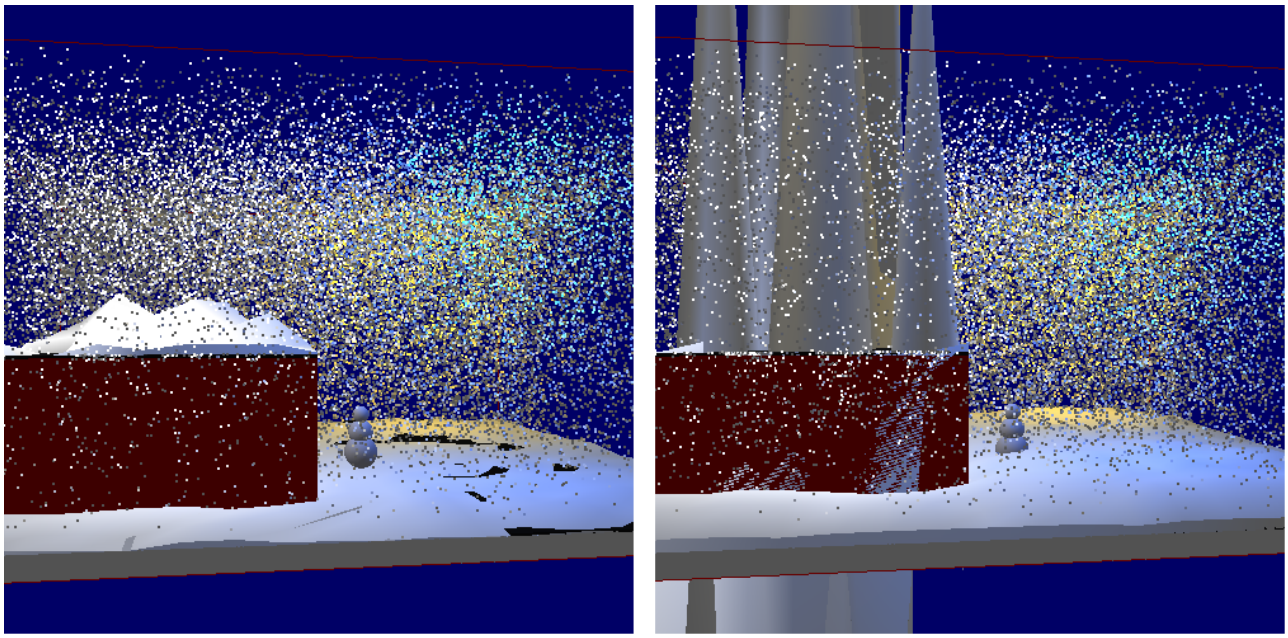

Figure 4 - Increasing Instability for Unrealistically Large Snowflakes

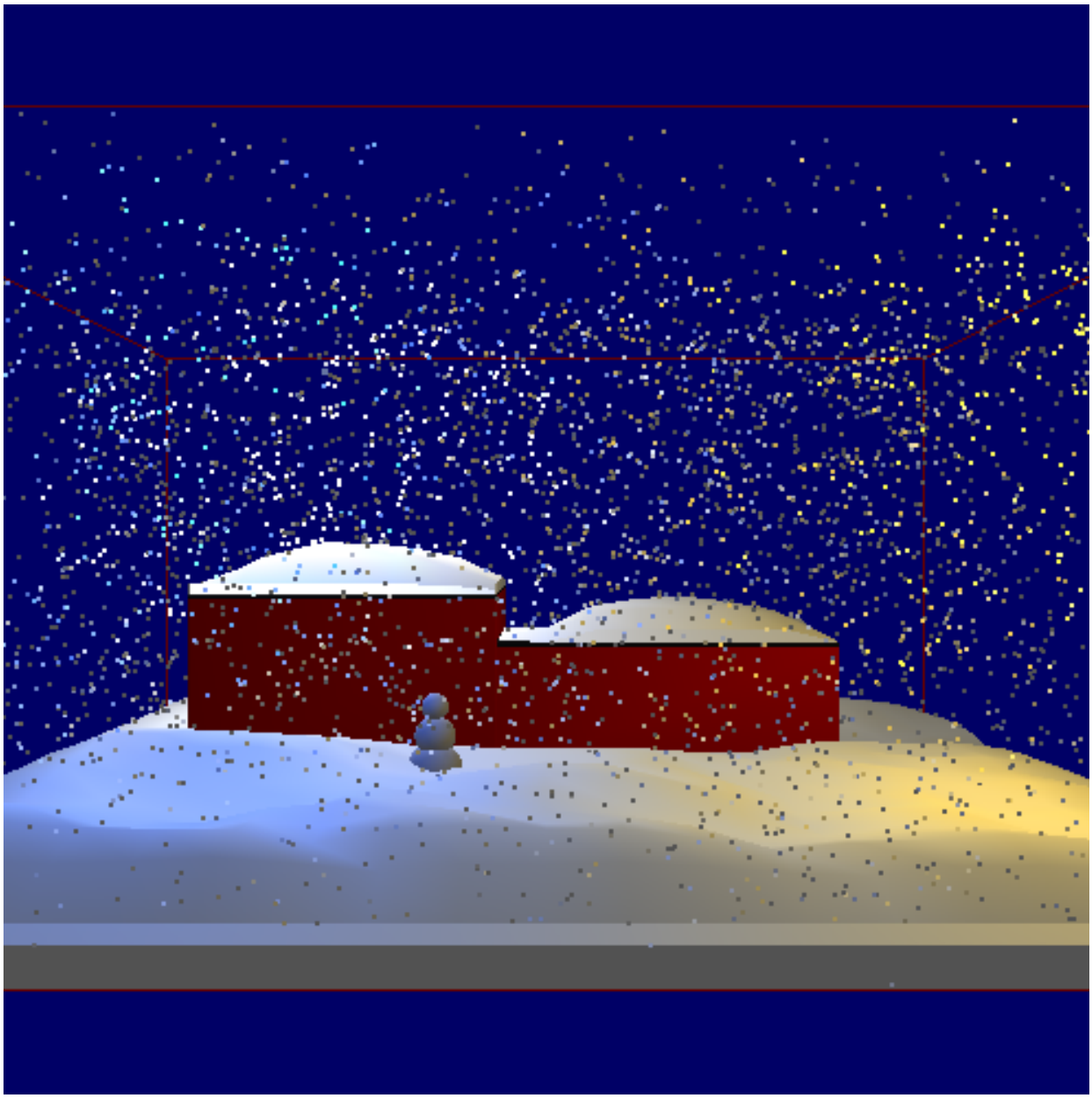

Figure 5 - Larger Version of Figure 1

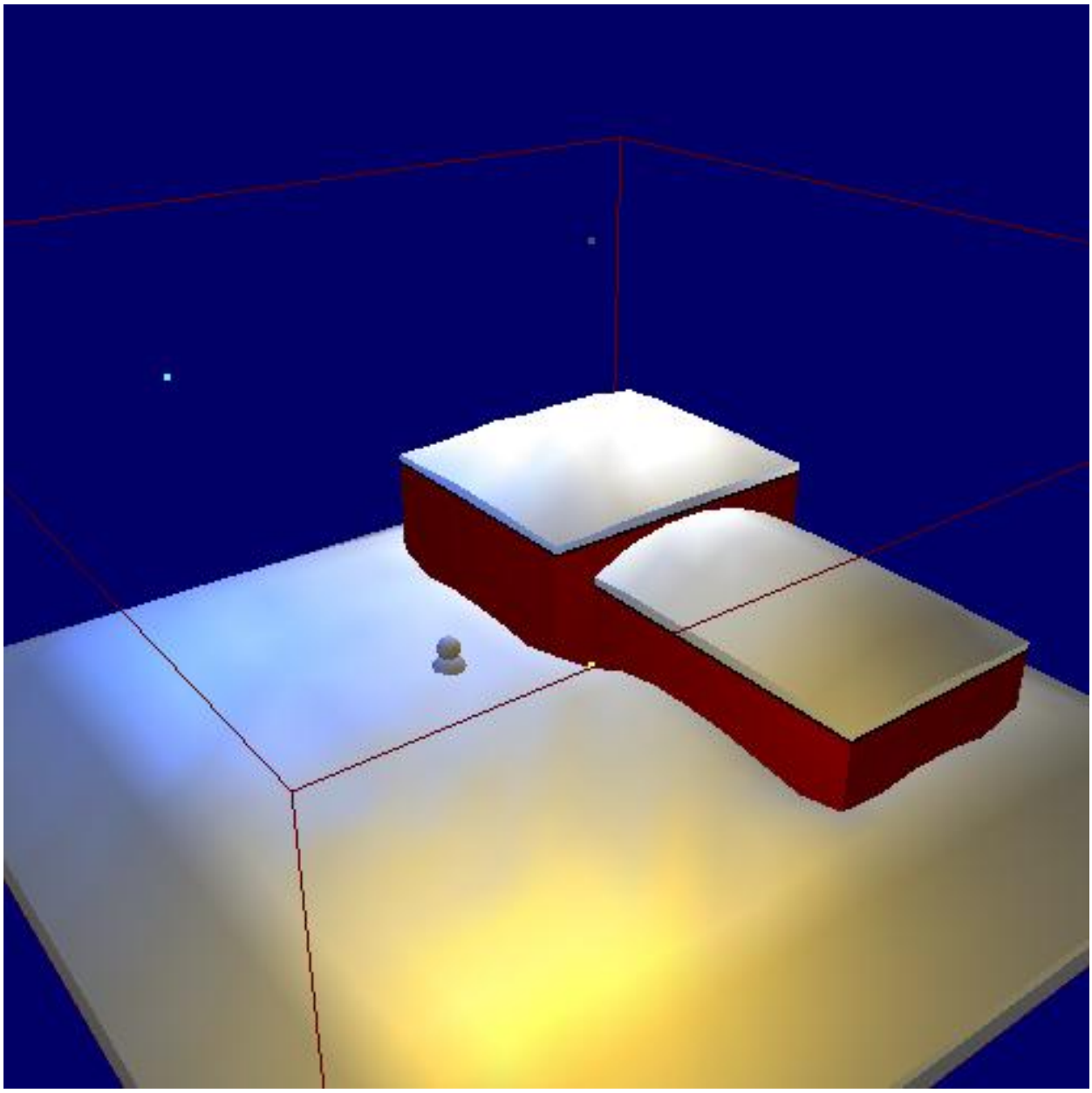

Figure 6 - The Results of a Particularly Heavy Snowfall

Additional Considerations as Part of Advanced Computer Graphics Class (i.e. not part of the actual paper proper)

Work Distribution

Chris Willmore

Randomized Snowfall
Height Field Representation
Snow Redistribution
Subversion Server Know-How
Code Optimization
Debugging

Scott Fermeglia

Wind
Lighting
Material Properties
Creation of Test Scene
This Presentation

Distribution of Time Spent

Meeting 1: Basic Setup…… 4 hours
Meeting 2: Wind, Objects…… 4 hours
Meeting 3: Lighting, Material Properties…… 5 hours
Meeting 4: New Test Scene, Accumulation via Height Fields…… 6 hours
Meeting 5: Height Fields on Objects, Randomized Wind, Code Optimization….. 7 hours
Meeting 6: Writing Presentation, Final Code Tweaks…… 4 hours
Meeting 7: Writing Final Report…… 3 hours

Other Time Spent:…… 2 hours each

Subversion Server at Revision:…… 102

Lines of Code:…… 2850