

Real-Time Snow Falling on Triangle Strips

Introduction

With the continually increasing speeds of computers and graphics hardware, interest in real-time modeling of phenomena has increased. In our case, we present a program that models snow falling on a terrain in real time. The program is extremely customizable. The user can specify the approximate number of snowflakes to appear (between 0 and 30,000), the lifetime of snowflakes (in seconds), the size of snowflakes, the height from which the snowflakes fall, the x, y, and z components of the wind force acting on the snow, and the gravity acting on the particles (from a list of Earth, Moon, and Mars). There are other options, including changing the camera view, taking a snapshot, and taking a movie (100 straight snapshots). The current terrain and option settings can also be saved to a file for later importation into the program.

Snow has been widely studied in the computer graphics literature. For one thing, it presents a prime example of a use for a particle system in modeling a natural phenomenon. Snow also generally stays in one place after it falls (aside from possible drifts), providing an interesting scene to render. Accumulated snow naturally forms relatively smooth curves, which allows the designer to apply techniques for producing smooth surfaces to the design of the accumulation. Snow is also useful in games when a designer wants to indicate that a location is cold or that there is bad weather. In addition, *The Day After Tomorrow*, a soon-to-be-released movie from Twentieth Century Fox about the coming of a new Ice Age, the need for a good snow simulation is obvious.

The research into modeling snow has generally fallen into two camps. On one side, researchers would like to simulate the way snowflakes fall. This includes the various forces acting on snow, including gravity, the force of the wind blowing it, drag, and interactions between snowflakes. For these researchers, the path along which snowflakes fall is important. Many of these researchers are also interested in similar particle systems, such as sand, fire, smoke, and grass. For more background on particle systems, see the classic paper by Reeves. On the other hand, some researchers are interested only in the effects of snowfall on a landscape. To them, the path is not important; only the end result is. This can greatly change the desired model.

For instance, instead of being time-driven, as an animation of snowfall would be, this model could be event-driven: whenever a snowflake falls on the terrain, its accumulation could be added. Fearing's work concentrated on this area. Although his algorithm produces realistic scenes, it does not render them in real-time. Haglund, et al., and Ohlsson arrived at algorithms to do this, although they took shortcuts that are evident in the lower-quality images produced by their respective algorithms.

Despite the work done in this area, major advances await in the snow simulation field. Programs modeling snowfall do not generally represent accumulation with any reasonable accuracy. The goal of our program is to give up some accuracy on both ends with the goal of creating a reasonable combination of the two. Thus, we aim to create a simulation with a reasonable depiction of both falling snow and snow accumulation. Of course, by performing multiple tasks, we would inevitably detract from both. But we felt that the end product would be advanced enough to provide users with a reasonable approximation in both areas. Another goal was to provide the user with control over the simulation both through an interactive graphical user interface and through an easy-to-use file format. Also, this program was written to simultaneously work on two platforms – Win32 and Mac OS X, and it should be easily portable to any platform that supports GLUT.

Methodology

Our program consists of a number of different components. The most important components, of course, are the snowflakes and the surface on which they fall. However, there are many other components that are necessary in order to produce an interesting simulation. These include the controls in the graphical user interface, the modeling of external forces, and the file formats used for the simulation and surface.

The snow is modeled as a particle system. To ease computation, we decided to simply display the snow as standard OpenGL points. The user has the option of turning on blending and/or point smoothing. While these make the snowflakes look nicer, abandoning them could boost performance on slower machines. We tried using `GL_EXT_POINT_PARAMETERS` and `GL_ARB_POINT_PARAMETERS`, which make the point size vary with distance from the view plane; however, after we worked out the headaches of making this compile and run for multiple graphics cards and on multiple platforms, the extension only seemed to work on one of the

machines. Even on the working machine, the distance effect did not actually make the snowflakes look more realistic. We abandoned this approach due to these problems.

We also considered more complicated ways of rendering snowflakes. We considered using quads and coloring those using textures. To get the desired effect, we would use billboarding, which would orient each quad so that it faced the camera. While this sounded appealing, the added computational overhead would have slowed things down considerably, especially with thousands of snowflakes. In addition, the quads would generally be so small that the look of the texture would not be significant.

The surface took a lot of planning. On the one hand, we wanted a surface that would be relatively smooth. We also wanted one that would be easy to specify. It should not be too hard to render, as it would have to be rendered alongside thousands of snowflakes. In addition, since we wanted some way to represent accumulation based on where the snow hit, we needed to develop a surface that could be changed locally in both height and color.

We decided to use a collection of overlapping 4X4 Bezier patches to represent the surface. This provided easy C^0 continuity by reusing control points at the edges. We felt that achieving C^1 continuity would limit the amount of local control. There are some obvious benefits to using Bezier patches. For one thing, it is easy to specify such a surface. In fact, the file format we use to represent such a surface contains a few initial parameters combined with a matrix representing the heights at each point in the curve. Thus, it is easy to visualize what the surface would look like simply by looking at the file.

Bezier patches allow us to easily grow the surface in a semi-local manner and also to color patches based the amount of snow hitting them. When snow hits a location, we perform an approximate intersection based on the surrounding control points (the actual intersection would take too much time for the real-time simulation of thousands of snowflakes). We then slightly increase the height of the affected control point. The Bezier patches are regenerated at every redrawing of the surface, so this is seen immediately. We map a texture onto the patches to represent the colors of the surface. We originally planned to store a matrix of color values, with one for each control point, but using textures proved to be more convenient. When snow hits near a control point, we simply interpolate the current texture color with white. Thus, the surface slowly turns white based on how often it gets hit by snowflakes.

Other formats we considered lacked some of the features of Bezier patches. At first, we considered using a subdivision surface to achieve continuity. This was appealing, as we already had code to model this surface. However, there were several problems with this approach. For one thing, we would only be able to subdivide a little bit, since we wanted to allow the user to rotate the scene in real-time; this would be impossible with a complex surface unless we added detail elision, which takes away from the feeling of realism. The presence of thousands of snowflakes would further slow things down. In addition, it would be hard to figure out where the snowflakes intersect the surface in an efficient manner. Also, as snowflakes accumulate, the extra continuity achieved by using this representation would be lost, as spikes would appear.

We built a wind model into our program. For wind, the program allows the user to specify the x, y, and z components of the wind force acting on the snowflakes. The units are sort of arbitrary, but, since the change is immediate, the user can adjust the values until the desired wind direction and force is reached. We considered allowing the user to specify wind using a vector field or a path drawn on the screen, but we abandoned this due to time and computational constraints.

Our simulation allows the user to choose whether the acceleration of the falling snowflakes should be based on the gravity of the Earth, the Moon, or Mars. The relative acceleration values are then used in snowflake calculations. Since all snowflakes at a given time are of the same size (again, due to graphical display constraints), we decided not to factor snowflake size into this calculation.

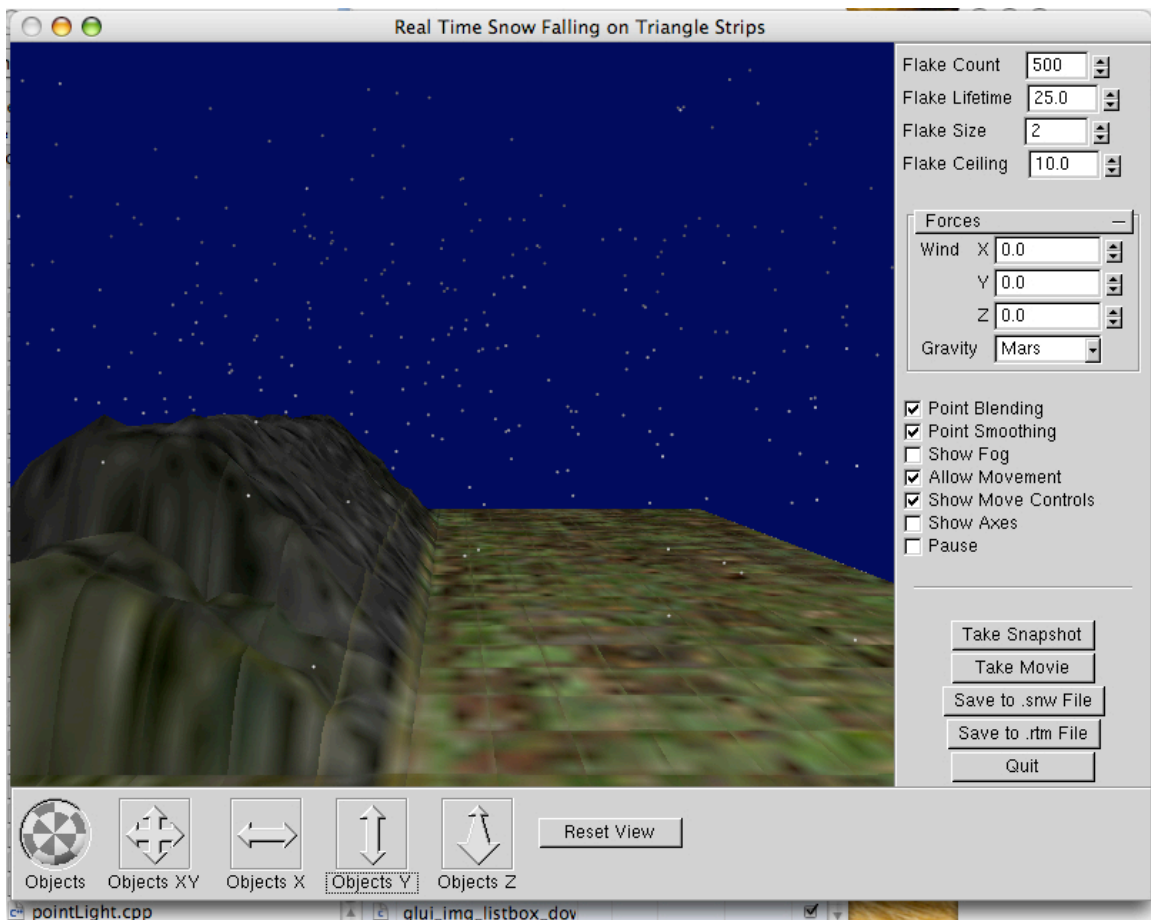
Another major decision was the graphical environment we would use. We decided early on to use OpenGL, since we wanted the program to work on multiple operating systems. For user controls, we decided to use Paul Rademacher's GLUI extension to GLUT. It provided us with text boxes, spinners, buttons, checkboxes, and camera controls. These allowed us to easily make our program interactive on any system that supports GLUT. While Win32 controls might have been nicer and more efficient, they would have limited the program to only working on the Windows platform.

We also had to consider what type of file format to use and what features to allow the user to specify. An XML-based file format was appealing; however, writing our own parser would have been too tedious, and Apache's Xerces parser requires too much overhead. Therefore, we decided to create a file format adapted from the .ray format. We were even able to

reuse some commands. For a list of supported commands, see Appendix A. In addition, we also have a separate file format for specifying the surface. There are a number of constraints on the terrain size, however. This file is also described in Appendix A.

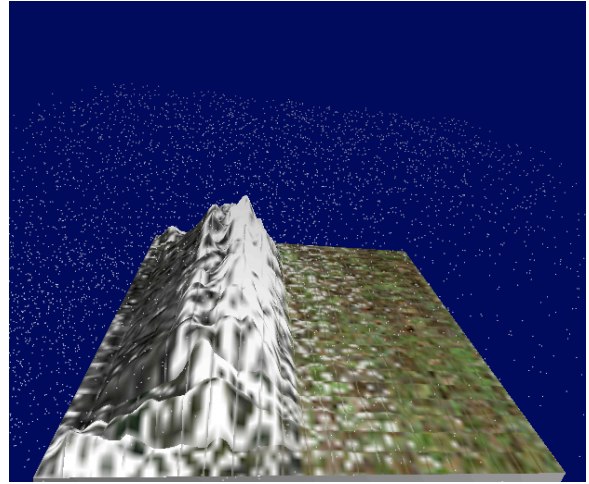
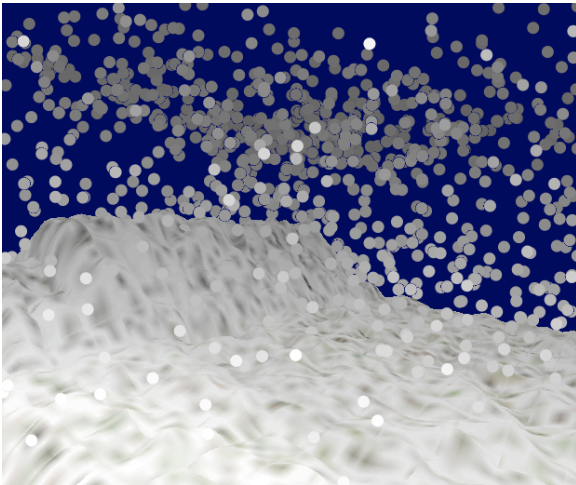
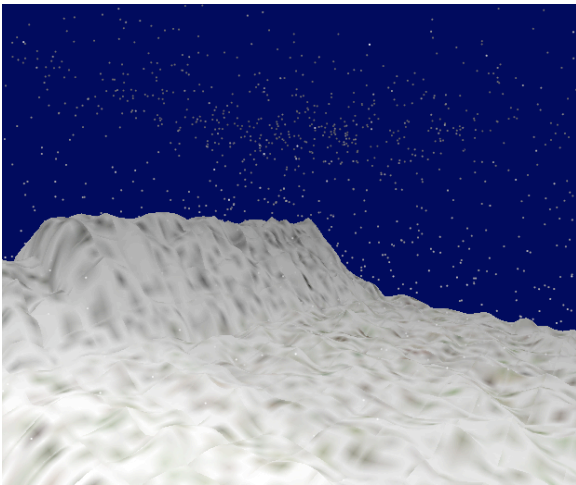
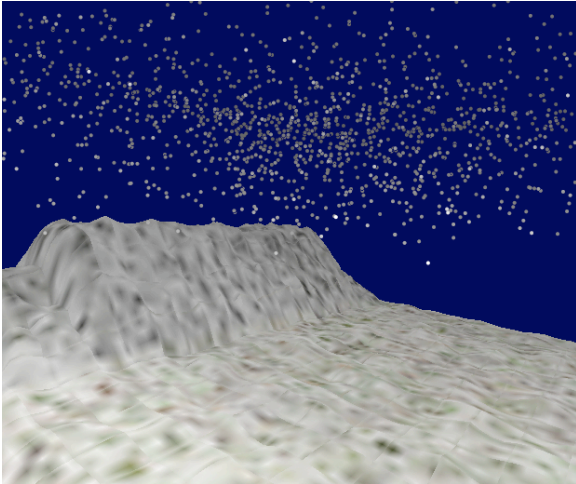
Results

Here are some results from our program.

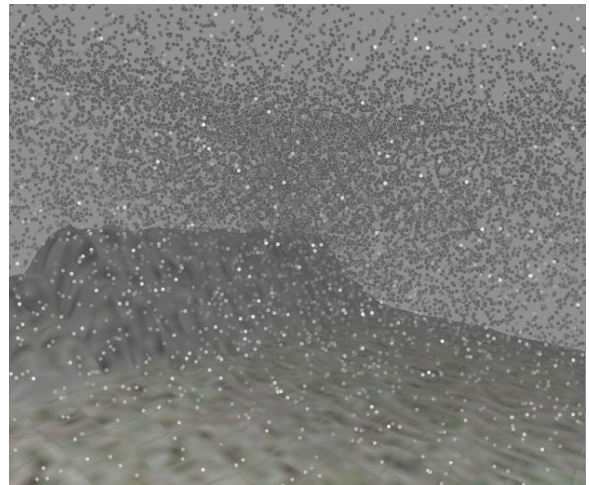


*This image shows the interface in action,
with all of the settings on the right set to the values entered in the input file.*

*In the sequence of images below,
the snow coverage increases over time.*



*This image shows the
partial accumulation effects of wind.*



This image shows fog effects.

Discussion

Overall, we are impressed with the results of our program. Although better models exist for modeling snowfall and accumulation separately, we feel that our program does a good job of trading off between the two. The snowfall and accumulation each seems fairly realistic.

There are many ways the current program could be improved. A more realistic model for snowfall could improve the simulation; this could include implementing a drag force, allowing snowflakes of multiple sizes to fall at the same time, simulating collisions among snowflakes, allowing the wind to blow snow that has already accumulated, and creating a better model for intersections. However, each of these would require more work and more computation; we chose to implement those features that we thought would make the biggest difference in the speed / realism tradeoff.

We encountered a number of unexpected difficulties. OpenGL extensions proved more difficult to understand and use than we thought. Generating spline surfaces was also more difficult than we imagined it would be. We ended up adapting freely-available code from Neon Helium Productions for this purpose. We still had to spend time generating the patches and lining them up, but it made the process easier. Textures also proved to be more difficult than we thought to adapt. After much tweaking, however, we were able to get it to work. However, we still had problems getting the textures to work at seams. OpenGL either uses a border-color or interpolates from the other side of the patch, whereas we would like to simply use the color interpolated from the corners. Therefore, the final result looks patchy.

Another difficulty involved generating a relatively smooth surface after snow accumulation. Originally, we simply added to the height of the nearest control point. While this provided a simple way to record the change in terrain due to the snowfall, it created spikes in the surface. Snow generally has a smooth appearance; these disturbances in our surface took away from the program's realism. We addressed this issue by also slightly increasing the heights of surrounding points. This treatment is similar to what would happen if a snowball would hit the ground. While the center of impact would be highest, the surrounding areas would also receive more snow.

Conclusion

Real-time simulation is an intriguing area of computer graphics. Faster processors and graphics cards allow these simulations to contain more and more detail. However, the quality of images produced cannot compete with those rendered more completely, such as by a ray tracer. Similarly, if a program focuses entirely on snowfall, it can implement other features that ours lacks, such as billboarding. In essence, our program is a “Jack of all trades but master of none.” At the same time, however, it is a lot of fun to watch.

Acknowledgments

We would like to thank Jason Lawrence for his help in formulating the ideas used in our program. Paul Rademacher’s GLUI extension to GLUT (available at <http://www.cs.unc.edu/~rademach/glui/>) played a major role in our user interface. We used the tutorials from Neon Helium Productions (<http://nehe.gamedev.net>) for help in generating Bezier patches and mapping textures. We adapted some of their functions in our code. We also used SGI’s website for information on OpenGL extensions, although we eventually decided not to use them. In addition, some of the textures were taken from the web. In particular, the texture for the mountains was taken from http://www.g-point.com/xpcity/textures/esptextures/roads/xp_mid_forkd.jpg, and the texture for the grass was taken from http://godevill.hp.infoseek.co.jp/works/texture/ground_grass03.jpg.

Appendix A

Snow file format: commands

`#camera`

Used the same way as in `.ray` files. Only the first `#camera` command is read.

`#light_num n`

Specifies that there will be `n` lights. This must come before lights are specified.

`#light_point, #light_spot, #light_dir`

Specify a point light, a spotlight, and a directional light, respectively. These are defined as in a `.ray` file.

`#terrain_file !file!`

This specifies that the description of the surface is located in `file`.

`#wind_vector x f y f z f`

This specifies the `x`, `y`, and `z` values for the wind force acting on the snow. The 3 values are floats.

`#max_flakes n`

This specifies the desired maximum number of snowflakes. This can range from 0 to 30,000.

`#initial_pos x f y f z f`

This specifies the `x`, `y`, and `z` values for the initial position of the starting snowflakes. The 3 values are floats.

`#initial_pos_randomness x f y f z f`

This specifies the `x`, `y`, and `z` values for the randomness that should be associated with the starting position of the snowflakes. The 3 values are floats.

`#initial_velocity x f y f z f`

This specifies the `x`, `y`, and `z` values for the initial velocity of the snow. The 3 values are floats.

`#initial_velocity_randomness x f y f z f`

This specifies the `x`, `y`, and `z` values for the randomness of the initial velocity of the snow. The 3 values are floats.

`#blend (0 | 1)`

This specifies if blending should be used.

`#smooth (0 | 1)`

This specifies if point smoothing should be used.

`#fog (0 | 1)`

This specifies if fog should be used.

`#snowflake_size n`

This specifies the size of the snowflakes. This value is passed to OpenGL.

```
#gravity (earth | moon | mars)
```

This specifies if the gravity should resemble that of Earth, the Moon, or Mars.

```
#lifetime nf
```

This specifies the lifetime for snowflakes in seconds. Of course, if snowflakes intersect with the surface, they are “killed” prematurely. This value is a float.

```
#clipping_planes nearf farf
```

This specifies the locations of the near and far clipping planes. These values are floats. This specifies if point smoothing should be used.

Terrain file format

The file format for specifying the surface is extremely simple. There are no commands; all values must be typed in the following order, separated by whitespace.

```
m n
```

The number of control points wide (m) and high (n). Note that due to constraints imposed by the texture mapping and the way we plot the Bezier patches, these values must be the same and must be either 16 or 64.

```
multxz
```

This specifies the scaling factor for the layout of the terrain in the X-Z plane. A value greater than 1 implies that the terrain will be expanded; a value less than 1 implies that it will be condensed.

```
multy
```

This specifies the scaling factor for the heights of the terrain.

```
!bitmap!
```

This specifies the .bmp file of the texture corresponding to this file.

```
x11 ... x1n
```

```
...
```

```
xm1 ... xmn
```

The remaining values represent the matrix of heights.

References

- Fearing, Paul. "Computer Modelling [*sic*] of Fallen Snow." SIGGRAPH 2000. Available: <http://doi.acm.org/10.1145/344779.344809>.
- Feldman, Bryan E. and James F. O'Brien. "Modeling the Accumulation of Wind-Driven Snow: Technical Sketch." ACM SIGGRAPH 2002. Available: <http://www.cs.berkeley.edu/~job/Papers/feldman-2002-MAW.pdf>.
- Haglund, Håkan, Mattias Andersson, and Anders Hast. "Snow Accumulation in Real-Time." SIGRAD 2002, pp. 11 – 15. Available: <http://www.ep.liu.se/ecp/007/002/ecp00702.pdf>
- Ohlsson, Per. "Real-time Rendering of Accumulated Snow." Uppsala Master's Theses in Computer Science 267. Available: <http://home.student.uu.se/p/peoh2665/SnowProject/finalPaper.pdf>. 4 January 2004.
- Reeves, W. Particle systems - a technique for modelling a class of fuzzy objects. *Computer Graphics* 17, 3 (July 1983), 359-376.