Melvin Diaz and Timothy Campbell
Advanced Computer Graphics
Final Report: ICU Virtual Window OpenGL Cloud and Snow Simulation

Introduction

This paper describes the implementation of OpenGL effects for the T-CEPS ICU Virtual Window system being developed by the RPI Design Lab as part of the ECSE Senior Capstone Design course. The overall goal of the system is to develop a photo realistic simulated window as opposed to actual window to provide better therapeutic benefits to the patient. Our graphics code is simply the front-end of a much larger project that includes the central control station for all of the ICU rooms, the server that sends different scenes to each of the rooms, and the human-machine interface that allows the doctors to set the scene for each screen.

Related Work

There are other graphics examples of photorealistic generated clouds and weather effects. Arguably the most relevant implementation of clouds has been developed by a company called Real Environment Xtreme. Their cloud and sky graphics have been sold as add-ons in the Microsoft Flight Simulator software series. This paper presented a fairly realistic goal for this project, both because of the stunning realism they were able to achieve, and because of the fact that the system runs in real time.

Figure 1: An example of a 3D graphics scene from Real Environment Xtreme.

In addition, outside research helped inspire our algorithm for rendering clouds. Of the most importance was a paper written by Niniane Wang, who worked for Microsoft and helped develop the clouds that are utilized in Real Environment Xtreme. Her paper describes a method for having clouds generate randomly while minimizing the computation time.

Most of Wang's work involves clouds in a 3D environment. However we will only be using a 2D environment, so some of the work was either omitted or changed. For example, we will not be concerned about cloud rotation as it relates to the camera, nor do we need to worry about how clouds will fade between layers as the camera passes through them. We will, however, still be concerned with creating a general 3D shape, since these clouds will be viewed from the ground, and they cannot look like flat textures in the sky.

Approach and Algorithm

Base System

The entire ICU Virtual Windows project uses QT as a cross-platform development standard for easy development across multiple operating systems. The QT library has several features that make the implementation easier, such as bindings for networking and OpenGL, and a rudimentary image processing library for creating, manipulating and displaying images.

The virtual window itself is created under a single shader, which is able to decide when a texture is being drawn, or the sky gradient is being drawn. The advantage of using a universal shader is the ability to adjust the global color of the scene. For instance, during sunrise/sunset, the scene can be tinted an orange/yellow color to simulate the effect of a sun on the horizon. This shader is also able to intelligently adjust the perspective of the scene as the window size changes. This way the scene always takes up as much of the window as possible.

Clouds

Clouds are currently rendered as static 2D images in the sky. The initial implementation had only single static pictures of clouds that floated slowly across the scene. This is the obvious approach, but is not acceptable for several reasons. There are only so many cloud pictures, and after enough time has passed, one will start to recognize the same cloud passing through the scene more than once. Also, these pictures of clouds look fake; there is no shading going on, so not only do they look strange as a result of incorrect lighting, but they look flat against the sky.

Our approach is, instead of using single pictures for clouds, to use a predefined set of cloud “sprites” to build new clouds. In our case, we use a set of 16 cloud sprites similar to the ones used by Wang (Fig. 2). The one on the top right is used for cloud bottoms. The other top three can be used for lower-altitude stratus clouds, or higher-altitude cirrus clouds. The bottom six are used to create detail in mid-altitude cumulus clouds. The remaining six are general purpose and may be used for almost any cloud type.

To create a cloud, we define an area to use, which may be any shape. The most common clouds are more horizontal then vertical, and as a result this is the most common shape chosen. Sprites are then added at random, starting from the center, with a bias slowly changing from the middle to the outsides of the bounding box. Wang recommends as many as 400 cloud sprites to create a single unique cloud, however this has its own complications like over-saturation, and an overall loss of control when trying to define a shape. As a result, this implementation only uses between 20 and 40 sprites to create a single cloud. The Qt libraries allow us to create and combine images with ease, thus the creation of the cloud images themselves is taken care of.

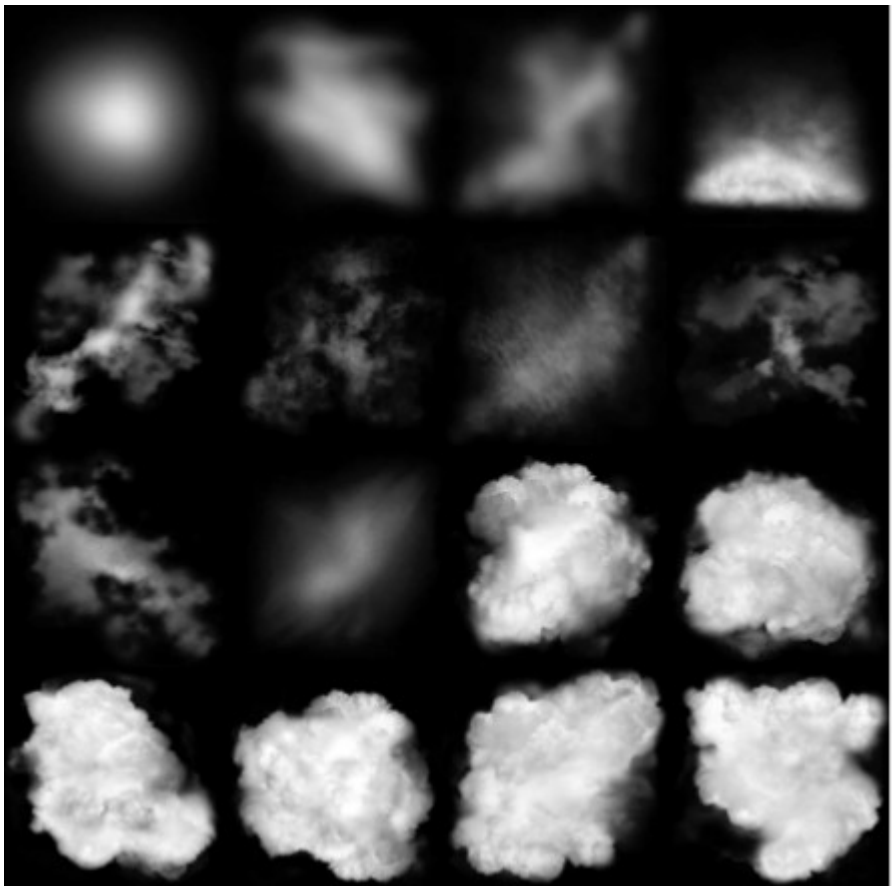

Figure 2: The cloud sprite images that were used for cloud generation.

Snow

Another weather feature being added to the Virtual Window is snow. As far as precipitation goes, snow is probably the the lest difficult to implement in a simulation. The force of gravity can be ignored, since snow always appears to fall at a constant velocity. Collisions are also not necessary for the same reason; a person watching snow will not notice collisions.

The snow is created by using a basic particle system. Particle systems are very commonly used in computer graphics to simulate weather effects such as snow as well as fire and explosions. In our information, a typical snow particle contains various information about its state, such as velocity, position, lifespan, and time alive. Each particle is generated at the top of the window, and is deleted when it reaches the bottom of the window only to be replaced by a new particle at the top. Each snow particle falls at a constant velocity, with varying degrees of horizontal movement. Snow particles are also slightly transparent to give them a softer look; otherwise they would be hard, white dots on the screen.

Determining when a snow particle should be deleted is potentially the most computationally intensive part of this simulation. The naïve approach would be to check a particle's position every timestep to determine whether the particle has passed the bottom, and delete it. However this would take too much time. Our solution is to use a lifespan system. Since a snow particle's downward velocity remains constant for the duration of the simulation, we can use the window's size to determine exactly when the particle will reach the bottom of the screen, and delete it when the time arrives.

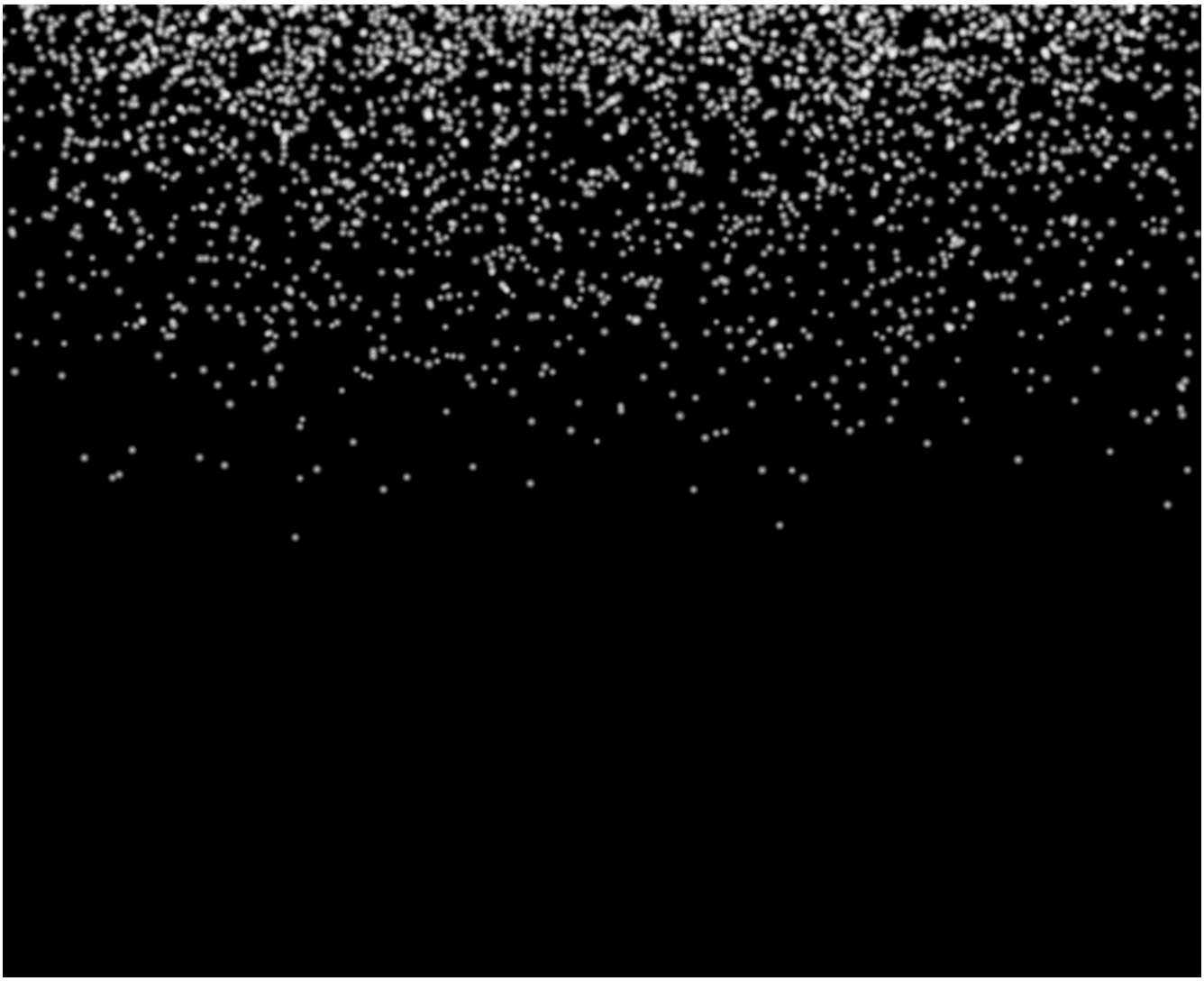

Figure 3: A sample image of snow falling on a black background.

Results

On the whole, we have been pleased with our results. However testing these new features in the Virtual Window itself have been difficult. The control mechanism for the window will be a central control panel on a server, which will be in contact with the individual window units throughout the hospital. This functionality is currently not implemented, and as a result all types of weather is on all the time. The only way to control weather is by commenting/uncommenting parts of the code. However we have been met with some success.

Clouds

For clouds, we were able to get successful generation of a cloud comprised of multiple sprites to appear integrated with the rest of the OpenGL window. There are still a few issues present. For instance, the quality of the clouds generated is not consistent; sometimes it looks too blocky, while other times it will look washed washed out, or the sprites themselves will overstep the bounds of the image, resulting in a cloud that looks cut off. This is a non-trivial problem to solve, since the cloud

generation is done randomly. In Wang's paper, she recommends that clouds be created by hand, since it is difficult to get a computer to consistently create the desired effect.

Another problem the clouds face is overall shading. Normally, clouds are a darker shade of grey on the bottom than on the top because the bottom of the cloud reflects less sunlight. This would have to be done based on the individual sprites, as a universal gradient would look strange.

Snow

For snow, the particle system runs and compiles and snow falls at a steady rate, however we have been unable to integrate the snow with the rest of the virtual window system. In order for it to be able to snow, there must exist a very thick, gray cloud cover and there must be a winter scene present on the Virtual Window screen. Information on the climate and weather conditions are kept in an XML file that contains all of the data for that particular scene, and it has not been changed to allow for the probability of weather events.

Thanks to snow's relatively simple nature, the current implementation of snow looks quite realistic. Snow particles are generated with a random size, which gives the illusion of a 3D space.

Figure 4: An example of the current cloud implementation

Future considerations

While significant progress has been made to this simulation, there is always room for improvement, especially when trying to recreate the world. Here are some relevant areas where we could improve on in the future.

Intelligent cloud generation

The current implementation of clouds is completely random, which can potentially result in a lot of chaos. A strategy for overcoming this problem is by intelligently generating the clouds. This can be done by creating clouds in segments; we can create a top piece, bottom piece, a middle piece, and some amount of small details around the edges. This eliminates some of the random chance, and gives us more control over the shape of the cloud.

Cloud shading

Clouds are not two-dimensional. There is always some amount of lighting working in a cloud. However we are able to cheat by keeping the light source behind us at all times, and as a result the lighting portion of the simulation is relatively simple. For a cloud, the most important aspect is that the bottom remain darker, and the top remain lighter. Using the previously mentioned intelligent cloud generation, we would be able to shade these individual parts, resulting in more believable cloud images.

Cloud layers

Like the clouds on it, the sky is also not two-dimensional. When clouds are generated, they can't all be the same size, and they can't be moving all at the same speed. What we can do is create various degrees of separation from the viewpoint for each cloud. For example, each cloud can carry with it its distance from the camera, and with that its velocity. This would give the illusion of a 3D sky.

Works Cited

Jacobs, B. (2011). *Particle Systems*. Retrieved April 25th, 2011, from OpenGL tutorial: http://www.videotutorialsrock.com/opengl_tutorial/particle_system/text.php

Wang, N. (2003). *Realistic and Fast Cloud Rendering.* Google, Inc.