Real Time Rendering of 3D Clouds

Pantelis Elinas, The University of British Columbia

and

Wolfgang Stuerzlinger, York University

There is a myriad of real-time graphics applications that can greatly increase in realism with the introduction of realistic clouds in the scenery. These include applications in virtual reality, outdoor scene visualization, flight simulation, games etc. In the past, several researchers have proposed algorithms for realistic cloud rendering for non-real-time applications. Of these, the approaches of Gardner 85 and Nishita 96 stand out as the ones that give the most visually pleasing results. In our work, we are developing an implementation of Gardner's 3D cloud rendering algorithm augmented with recent advances such as photon maps with the help of hardware accelerated OpenGL.

Our implementation follows closely, Gardner's approach. We use textured ellipsoids as the building blocks for the clouds. We use Perlin noise textures compared to Gardner's spectral noise textures because we have more flexibility in controlling their appearance. The texture used is an indication of the volume density of the ellipsoid. Rendering the ellipsoids produces ellipses. Because clouds are more transparent at their edges compared to their middle, a result of the fact that the center of a cloud is denser, we need to fuzz out the edges of the ellipses.

Each cloud is made of a number of ellipsoids. The first step is to render each of the ellipsoids individually and evaluate the transparency over the resulting ellipse. The next step is to combine all the ellipses together to render the complete cloud.

Equation (1) is used to compute the transparency as a function of two thresholds one applied to the edge (T_1) and one applied to the center of the ellipse (T_2). The values produced are consistent with OpenGL alpha channel values. We use a normalizing factor D just as Gardner did. We generate the function g(x,z) over the ellipsoid by utilizing projective textures. The latter function is maximum at the center of the ellipse and goes to zero at its edges; a spotlight texture is used to generate this effect. Evaluation is done partly in the alpha channel and partly in the RGB channels of the frame buffer in two steps. First T2*(1-g(x,z)) is computed and stored in the RGB channels. The I_t-T_1 part is computed in the alpha channel. We then perform $\alpha = (R-\alpha)$ by utilizing the color matrix extension. We set the color

$$\alpha = \frac{\left(I_t - T_1 - T_2 * \left(1 - g(x,z)\right)\right)}{D} \quad (1)$$

matrix to copy the red to the alpha channel while setting up subtractive blending and then we perform the operation by copying the frame buffer onto itself. The 1/D scaling is also achieved in the same step by including the 1/D term in the color matrix. The result is stored in the alpha channel while the RGB channels are cleared to accept new data in the next step.

Next we render the ellipsoids using a second texture that determines their color. We use another Perlin noise texture modulated with the color of the ellipsoids calculated using an infinite light source representing the Sun. The latter produces a simple approximation to global illumination of the cloud even though effects such as self-shading of the cloud is not taken into account. We are currently investigating better approaches to this such as using photon maps. The results of this step are stored in the RGB channels.

At this point the back buffer holds each of the ellipses including their transparency. We read these data into a texture of the smallest size that includes all the ellipses. Then, we calculate where on the screen each of the ellipses should appear and we render textured rectangles for each of the ellipsoids. We render each of the rectangles in the order of back to front with respect to the position of the ellipsoids in 3D; the ellipsoids are originally sorted roughly from back to front with respect to the location of their center.

This implementation is compatible with standard OpenGL hardware in its majority with the exception of the color matrix extension that is only available on SGI machines. Our implementation achieves real-time frame rates on low-end workstations such the SGI NT 320 and SGI O2 machines. More complex clouds made of 130+ ellipsoids can be rendered in real-time on more powerful machines such as the SGI Onyx.

References

[1] Blinn F. James, Light Reflection Functions for Simulation of Clouds and Dusty Surfaces, Computer Graphics, Volume 16, Number 3, July 1982.

[2] Gardner Y. Geoffrey, Visual Simulation of Clouds, Siggraph 1985, Volume 19, Number 3, 1985.

[3] Gardner Y. Geoffrey, Simulation of Natural Scenes Using Textured Quadric Surfaces, Computer Graphics Volume 18, Number 3, July 1984.

[4] Jensen H.W., Christensen H.P., Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps, Computer Graphics Proceedings: Siggraph 1998, July 14-18th, 1998.

[5] Nishita Tomoyuki et al, Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light, Siggraph 1996, p. 379.

Poster Outline

Pantelis Elinas, The University of British Columbia and
Wolfgang Stuerzlinger, York University

- **Problem Statement**

We present a real-time implementation of rendering Gardner style clouds using widely available OpenGL accelerating hardware. We chose Gardner's method of cloud rendering because it is suitable to hardware evaluation and it provides visually pleasing results.

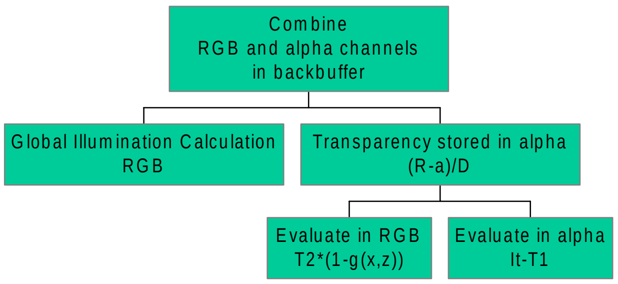
- **Algorithm**

We approximate the structure of a cloud by using a number of textured mapped ellipsoids. We use Perlin noise textures. We calculate the transparency of the cloud for each ellipsoid after projection i.e. on the resulting ellipse.

- **Justification for Transparency Calculation**

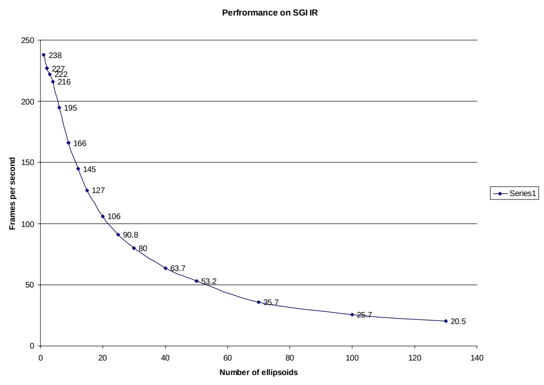
Clouds are more transparent at their edges because they are less dense there compared to their center. Our initial texture does not carry this information but we need to render the ellipsoids with fuzzy edges. We achieve the latter by evaluating a function over each projected ellipsoid i.e. resulting ellipse that makes the ellipse more opaque as we approach its center.

- **OpenGL implementation**



- **Results**

The plot on the bottom left shows the frames per second achieved for clouds made of different complexity. The image on the bottom right is an example of two clouds made of a total of our ellipsoids.


Perfrormance on SGI IR
Frames per second
Number of ellipsoids
Series1