A real-time cloud modeling, rendering, and animation system

Article · January 2003

CITATIONS
65

READS
167

4 authors, including:

David Ebert
Purdue University
161 PUBLICATIONS **3,511** CITATIONS

SEE PROFILE

Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

A Real-Time Cloud Modeling, Rendering, and Animation System

Joshua Schpok,[1†] Joseph Simons,[1] David S. Ebert,[1] and Charles Hansen[2‡]

[1] Purdue Rendering and Perceptualization Lab, Purdue University
[2] Scientific Computing and Imaging Institute, School of Computing, University of Utah

Abstract

Modeling and animating complex volumetric natural phenomena, such as clouds, is a difficult task. Most systems are difficult to use, require adjustment of numerous, complex parameters, and are non-interactive. Therefore, we have developed an intuitive, interactive system to artistically model, animate, and render visually convincing volumetric clouds using modern consumer graphics hardware. Our natural, high-level interface models volumetric clouds through the use of qualitative cloud attributes. The animation of the implicit skeletal structures and independent transformation of octaves of noise emulate various environmental conditions. The resulting interactive design, rendering, and animation system produces perceptually convincing volumetric cloud models that can be used in interactive systems or exported for higher quality offline rendering.

Keywords: cloud modeling, cloud animation, volume rendering, procedural animation

1. Introduction

Clouds, like other amorphous phenomena, elude traditional modeling techniques with their peculiar patterns of intricate, ever-changing volume-filling microstructures. To address this challenge, we have created an interactive system that allows artists to easily design and interactively animate visually convincing clouds. Accelerating their rendering to real-time extends their applications from static data visualization and movies to interactive exploration and video games. In addition, providing a responsive, interactive system aids comprehension of synthetic environments and increases the productivity of artists.

We have created a multi-level, interactive, volumetric, cloud modeling and animation system using intuitive, qualitative controls. Our approach scales from entire cloudscapes to detailed wisps appropriate for flythroughs. The cloud models and key-framed animation parameters may be exported to commercial animation packages for higher quality offline rendering.

Our system is composed of four main components: a high-level modeling and animation system, the low-level detail modeling and animation system, the renderer, and the user interface. We have designed an intuitive, multi-level interface for the system that makes it easy to use for both novice and expert users. To create cloud animations, the user interactively outlines the general shape of the clouds using implicit ellipsoids and animates them using traditional key-framing and particle system dynamics. The implicits are evaluated and shadowed over a grid in software, which is sent as triangle vertices to the graphics card. The user describes the cloud details and type (low-level modeling) from a collection of preset noise filters and animates them by specifying the windy environment. This detailed modeling and animation is actually performed using the graphics processor through the use of volumetric textures and texture transformations.

We begin with a brief survey of current cloud modeling techniques along with their implementations in commercial applications, then introduce our procedural approach for cloud formation and animation. Next, we describe our system implementation and user interface. Finally, we conclude by describing our planned future work.

† e-mail:schpokj;simonsj;ebertd@purdue.edu
‡ e-mail:hansen@cs.utah.edu

2. Previous Work

Approaches to cloud modeling may be classified as simulation-based or procedural [1]. Simulation methods produce realistic images by approximating the physical processes within a cloud. Computational fluid simulations produce some of the most realistic images and movement of gaseous phenomena. However, despite the recent breakthroughs in real-time fluid simulation [2], large scale high-quality simulation still exhausts commodity computational resources. Therefore, using simulation approaches makes cloud modeling a slow *offline process*, where artists manipulate low-complexity avatars as placeholders for the high-quality simulation results. With this method, fine tuning the results becomes a very slow, iterative process, where tweaking physical parameters may have no observable consequence or give rise to undesirable side-effects, including loss of precision and numeric instability [3]. Unpredictable results may be introduced from approximations in low resolution calculations during trial renders. These physics-based interfaces can also be very cumbersome and non-intuitive for artists to express their intention and can limit the animation by the laws of physics [4].

In contrast, procedural methods rely on mathematical primitives, such as volumetric implicit functions [5], fractals [6, 5], Fourier synthesis [7], and noise [8] to create the basic structure of the clouds. This approach can easily approximate phenomenological modeling techniques by generalizing high-level formation and evolution characteristics. These procedural modeling systems often produce more controllable results than true simulation in much less time. First, these models formulate the high-level cloud structure, such as fractal motion [9] or cellular automata [10]. Then, the model is reduced to renderable terms, such as fractals [5], implicits [11, 5], or particles [12, 9]. Many approaches use volumetric ellipsoids to roughly shape the clouds, then add detail procedurally [5, 13, 14]. We build atop this approach, leveraging additional control over the perturbation with noise to produce realistic turbulent animation.

Correct atmospheric illumination and shading is another difficult problem that must be addressed to produce convincing cloud images and animations. Early work discussed accurately modeling light diffusion through clouds, accounting for reflection, absorption, and scattering [15]. Further research has explored accurate volumetric light scattering and interreflection for clouds [16, 17, 12, 5, 18]. While full self-shadowing, translucent volume rendering is approaching interactive rates [14], we elect to implement a visually convincing approximate volumetric lighting model utilizing graphics acceleration to achieve true interactive performance.

Commercial software packages present cloud modeling systems as diverse as their underlying implementations. Many terrain generation programs, such as Corel Bryce, permit tweaking of single or multiple noisy fractal layers. In these cloudscapes, users interactively edit basic noise and raster filtering parameters, e.g., contrast, persistence, bias, and cutoff.

In commercial modelers such as Alias|Wavefront Maya, basic clouds may be simulated with particles. These particles serve as placeholders for offline rendering of high-quality cloud media. This cloud media may sometimes be substituted with high-resolution imposters, which typically use regions of transparent volumetric noise. Developing the interrelation between particle and renderable regions within the modeler involves building a dependency graph of attributes and filters (known as a *shader graph*) and assigning numerous variables and bindings to the tree's nodes. Shading systems of this complexity are a ramification of generality and customization. In fact, a non-realtime model of our rendering scheme may be implemented using such a system.

Our system addresses three shortcomings to these approaches. First, developing complex shader networks is a difficult, iterative task. Besides developing the required shader schematic, many hours are usually spent tweaking non-intuitive, technical parameters with indirect consequences. Second, the consequences of manual adjustment is often only apparent after offline, high-quality rendering. Finally, millions of particles may be needed for large-scale cloudscapes or fly-throughs. In contrast, with our interactive interface, a variable's effect becomes apparent with experimental adjustment, and this immediate response promotes understanding and exploration of the shader's numerous parameters.

3. A Phenomenological Approach to Modeling and Animation

The evolving structure of a cloud represents numerous atmospheric conditions [19, 20], which we model using the two-level approach proposed by Ebert [5]. At the high-level, we use volumetric implicits for general shaping control. For low-level cloud detail, we use volumetric procedures based on noise and turbulence simulations [8, 21].

3.1. High-Level Modeling & Animation

Visible clouds form based on the condensate interface between warm and cool fronts. Some models use an isosurface between temperature gradients or use surface-based ellipsoids [7, 13]. However, this approach does not capture the volumetric detail of the cloud. Volume data is important, not only for close-ups and fly-throughs, but for proper illumination as well. We, therefore, use volumetric implicits to model the clouds. These implicit functions have the beneficial attributes of smooth blending, simple computation, and malleability.

For rendering these models, we use Wyvill's cubic blending function to calculate the potential at a given point [22] and evaluate implicits on the vertices of tessellated planes slicing

the volume to perform volumetric rendering. The implicit field value defines the density, transparency and shadowing of the cloud, as described in Section 4. We elect to calculate these values in software and vertex programs to balance utilization with the fragment-level texture look-ups.

Artists begin shaping their cloud by positioning and sizing implicit ellipsoid primitives to define the cloud volume. Though optional indicators may assist the artist, the cloud forms within these primitives in real-time, so users immediately see the actual result.

Implicit ellipsoids may be animated with a variety of effects. The implicits form the basic element, or particle, that can be used with particle system dynamics. Simple particle system techniques can be used to control the macroscale cloud shape and dynamics, with procedural animation controlling the finer detail and cloud evolution. For example, gradual translation along user defined paths emulate prevailing wind effects. Combining various particle motions while increasing the implicit's radii simulates the evolution of naturally occurring cloud structures, which can be interactively key-framed. Key-framing enables more specific animation to be defined, such as clumping and rising primitives resembling stormy environments, and slowly expanding or shrinking implicits simulating growth or dissipation.

3.2. Low-Level Detail Modeling & Animation

Noise has historically provided a means to mimic natural phenomena. Perlin noise [8, 23] bears ubiquity in procedural modeling for its continuity and uniform randomness. Filtering this noise exposes new features. Our implementation uses a variety of noise filters to produce various types of clouds. Furthermore, users can adjust these filters with intuitive transformation widgets and high-level attribute parameters.

As a preprocess, the system computes and loads a volume texture of periodic noise to the graphics hardware. During run-time, the graphics hardware vertex program calculates each octave as a texture coordinate, and the hardware *fragment program* (commonly referred as a pixel shader [24]) composites the look-ups together, similar to Green [25].

This final value of noise modulates the opacity interpolated between vertices during rasterization. In essence, noise volumetrically "subtracts" away the volume, creating the desired detailed features.

Animating the cloud media simulates various atmospheric conditions. While high-level animation controls the general direction of cloud structure, noise animation depicts *why* the action occurs. If the finer octave's motion proportionally decreases, the cloud appears to be moving against some force; wisps strip away from the cloud edges and disappear. Conversely, accelerating coarser octaves conveys propulsion of the cloud with an auxiliary jet, blowing off tufts in its heading. Combining this animation with the decay of the implicit power depicts the cloud blowing apart.

We also allow the user to create atmospheric tiers with different noise characteristics and animation. The different tiers simulate different atmospheric layers and allow the clouds to move and evolve differently as their elevation varies.

Hardware	Operation
CPU	Generate slicing geometry Sample implicit functions Optional coarse noise evaluation Shadow accumulation
GPU: Vertex Program	Interpolate colors Calculate texture coordinates
GPU: Fragment Program	Transparency cut-off Composite noise octaves

Table 1: *Distribution of Operations*

4. Rendering

To visualize our primitives, we use a modified slice-based volume rendering scheme [26]. To render our scene quickly, we balance processing between the CPU, the vertex, and the fragment processing units on advanced hardware accelerators by sampling lower frequency functions on vertices. Table 1 outlines the processing distribution, and Figure 1 summarizes the rendering procedure.

4.1. CPU Operations

We begin by creating planes slicing through our volume. The planes are oriented parallel or orthogonal to the light vector in the orientation, minimizing the difference between the plane normal and the eye. By insisting on these orientations, vertices remain colinear along parallel rays of the light source.

The slicing planes are uniformly subdivided and the CPU calculates the implicit functions at each vertex. This magnitude is mapped to vertex opacity:

$$opacity_i = plane\ opacity \times \sum implicits \qquad (1)$$

Iterating across vertices colinear to a light ray, a fraction of this magnitude accumulates into the shadow buffer, which is mapped temporarily as the vertex color:

$$color_i = color_{i-1} + (shadow\ magnitude \times opacity_i) \quad (2)$$

Since our shadows may broadly change with larger, low frequency noise octaves not considered yet, we may sample the

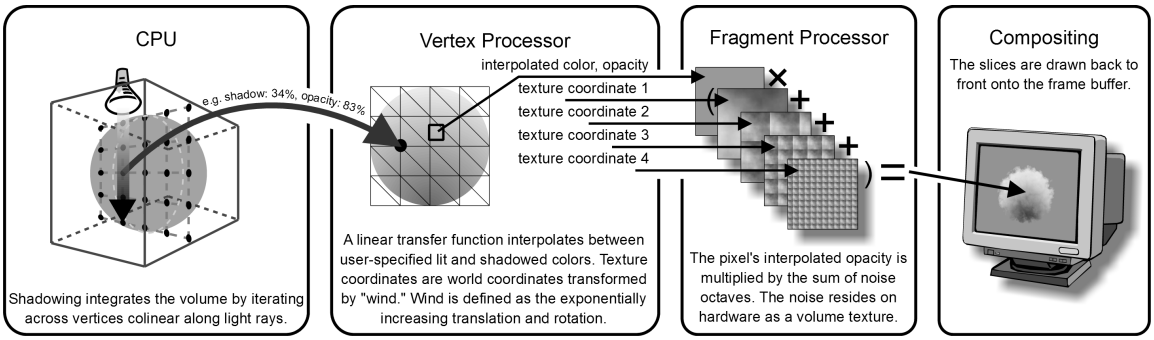

Figure 1: *Schematic diagram of operations*

first octave in system memory along with the implicit functions. Our rendering system was designed to produce visually convincing results at interactive rates. A resulting limitation with our vertex-shadowing scheme incorrectly darkens the cloud when higher octaves of noise later subtracts portions of the volume. Furthermore, shadowing resolution is limited to tessellation density, a trade-off between accuracy and performance. However, using the deterministic variables establishing the cloud scene, we may easily export it to other offline renderers for more accurate results.

The CPU also generates transformation matrices that later determine the texture coordinates. These matrices represent the transformations necessary to produce each octave of noise in each atmospheric tier. As described above, animated noise transforms higher octaves exponentially faster:

$$transform_{octave} = scale(f) \times rotate(f) \times translate(f)$$
$$f = octave^{lacunarity} \quad (3)$$

Each vertex has a world space coordinate, color (shadow depth), and opacity. In implementation, we initially build the tessellated planes and store them as static geometry on the graphics card. During run-time, only a single 32-bit color/opacity value per vertex needs refreshing. Employing graphics hardware's programmable stream model [27], we minimize data transmission to these dynamic values.

4.2. Vertex Operations

For every iteration, the CPU sends the necessary vertex information above to the vertex processor, along with "lit" and "shadowed" colors, tier altitudes, and noise transformation matrices. A linear transfer function evaluates the vertex's color. The vertex processor linearly interpolates from the lit to shadowed color varying along shadow depth. The vertex processor selects the appropriate set of noise transformations by comparing the vertex's world position against the specified altitudes. Texture coordinates are subsequently produced by multiplying the world position by these chosen matrices.

The vertex program produces new vertices with their final color and a set of texture coordinates. The hardware rasterizer interpolates these values.

4.3. Fragment Operations

From the rasterizer, the fragment processor receives the fragment's screen space coordinate, color, opacity, and texture coordinates. The fragment program uses each of the texture coordinates to index a pre-computed noise volume. These samples are weighted by a default fractal persistence of one half, and summed. This harmonic series is bound from 1 to 0. The fragment processor multiplies the opacity by this value and conditionally blends the fragment into the frame buffer, if it is above the alpha cutoff. To blend, we use a painter's algorithm, drawing planes back to front.

Currently, our implementation uses four octaves of noise, as it produces enough visual detail for fly-throughs. The latest generation of graphics hardware is capable of more octaves, at the cost of computing another transformation matrix, texture coordinate, and volumetric texture look-up. Distorted noise shearing may develop between tiers from discontinuous noise transformations. This aberration may be resolved by separating slicing geometry between tiers and blending over the gap. Additionally, we uniformly transform the lowest octave across all tiers because its low frequency most dramatically sculpts the volume and subsequently produces the most apparent discontinuity.

5. User Interface

We organize variables hierarchically through a tree of GLUI [28] roll-up groups. At the top-most groups, we expose the most common and general controls, and more detailed, specific controls under successive groups. Novice users can design complete clouds using the most basic controls, while advanced users can customize properties deeper in the tree.

Some rendering parameters influence the image in multiple ways. For example, increasing the number of slicing

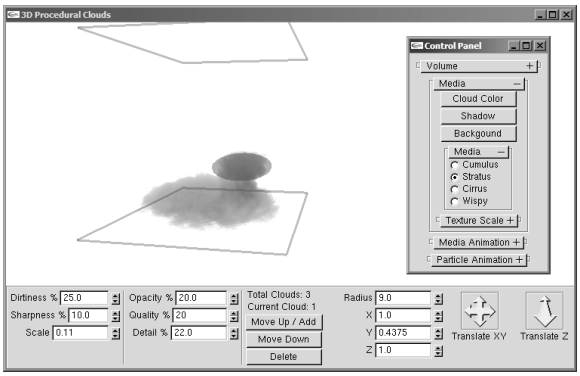

Figure 2: *The interface hierarchically organizes the controls to expose the most common first, and more specific customizations in successive levels.*

Attribute	Function
Opacity	*plane opacity* = **Opacity** ÷ *slicing planes* Adjusts the overall transparency of the cloud.
Quality	*slicing planes* = **Quality** Increasing the total slicing planes by resolving a finer image with greater continuity, but at the expense of performance.
Detail	*lacunarity* The fractal scaling of texture coordinates.
Dirtiness	*shadow magnitude* = **Dirtiness** ÷ *plane opacity* Adjusts how fast the cloud darkens.
Sharpness	$alpha\ cutoff$ = **Sharpness** × *plane opacity* Adjusts the alpha cutoff (cloud fuzziness, or blurriness), while compensating for opacity. Without compensation, adjusting opacity can shrink or expand the cloud.
Cloud	Noise range [0,1]
Media	**cumulus** = $\lvert noise \rvert$ **stratus** = *noise* **wispy** = $1 - \lvert 2 \times noise \rvert$ **cumulus** = $(1 + \lvert 10 \times noise \rvert)^{-1}$ Presents noise filters in a user-friendly qualitative manner.
Wind	*texture translation* A direction and magnitude widget to express "wind" direction.
Torque	*texture rotation* A direction and magnitude widget to express fractal texture rotation.

Table 2: *Cloud Attributes*

planes integrates the volume in smaller steps, increasing the visual opacity of the cloud. We have developed a system of equations exposing *qualitatively independent* parameters, summarized in Table 2. These attributes adjust the image along a single visual dimension without side-effects in another dimension.

We group controls into four general groups: global rendering parameters, cloud media shaping controls, cloud media animation, and high-level particle animation tools. Basic position, shaping, and media sculpting controls are located on the bottom of the render window, as shown in Figure 2. For simple scenes, clouds may be designed without using the more detailed control panels.

5.1. Rendering Controls

We express the number of slicing planes in terms of *quality*, where increasing this value resolves a finer image with greater continuity, but at the expense of performance.

The accumulated opacity increases with the addition of slicing planes. Therefore, we scale the transparency of the slicing planes by the user-defined *opacity* divided by the number of slicing planes.

Advanced settings permit finer adjustments to plane tessellation and selective octave rendering to balance image quality and performance. All variables can be saved and restored system-wide to revisit work.

5.2. Media Controls

Cloud media are sculpted with a set of controls modeling and rendering noise. We have developed a set of filters useful for various cloud textures. *Sharpness* adjusts the transparency cutoff (qualitatively, the blurriness toward cloud edges) multiplied by the plane opacity. By accounting for plane opacity, the clouds do not shrink and grow when it is adjusted. *Dirtiness* adjusts how the cloud darkens with depth. As slicing planes increase, the accumulated shadow grows, so this term is divided by the number of slicing planes. The actual cloud and shadow colors, along with the background, may be specified to simulate special lighting conditions such as sunset.

Cloud size is conveyed by the *scale* of the first octave. The fractal step in noise, lacunarity, is modified as the *detail* control. The visual effect of adjusting detail modifies the size of the smallest characteristics, and is useful for creating smooth or rough cloud detail.

5.3. Media Animation Controls

As mentioned previously, we smoothly evolve the noise volume by fractally transforming sequential octaves. For global cloud movement (translation), the user controls a directional and magnitude widget for ambient *wind*. We provide a similar interface to control *torque*, the fractal rotation of texture space.

These transformation controls influence the selected tier of noise. In this way, users can model unique flow conditions at different altitudes. A special tier, *Base*, applies the transformation over all tiers' first-octave noise, which conveys large-scale cloud evolution and cohesion between tiers.

5.4. Particle Animation Controls

As low-frequency noise effectively conceals the primitives' shape, our particle tools visualize this geometry for early design. We have implemented a set of traditional particle animation mechanisms to evolve the cloud in a variety of ways. Users can project particles along a uniform wind field, useful for slowly scrolling cloudscapes. Finer control is achievable through individually key-framing particles, and interpolating their position and shape properties over time.

6. Results

As seen in Figures 3 through 7, our system can create and animate various cloud types and cloudscapes. For a Pentium IV processor with a NVIDIA GeForce4 Ti4600, performance varies with the quantity of geometry and projected size on screen, but typically runs between 5 and 30 frames per second. We provide a balance between performance and rendered complexity with a "Quality" attribute that adjusts total slicing planes. In design, we begin cloudscape rendering at lower slicing resolutions to temporarily increase the frame rate, and later increase it for fine-tuning and proofing.

Compositing eight-bit values can result in quantization artifacts, particularly with many slicing planes with high transparency. More recent hardware supports 32-bit floating-point textures, capable of resolving this limitation.

In Figure 5, several basic motions govern the evolution of cumulonimbus clouds [29]. Simple ascending implicits model convection, while a combination of rising and rotating noise transformations model the mixing entrainment. If the rising thermal reaches a layer that its thermal buoyancy cannot penetrate, it spreads under the surface forming the familiar anvil head. A flat, growing ellipsoid emulates this expansion over the troposphere. To indicate the spreading motion in our media, we slow the rolling turbulence motion and begin scaling out noise to coincide with the widening plume.

In Figure 6, several scattered implicits emulate sunset light scattering. By setting the shadow color to bright pink, and the tops to a darker grey, we convey a setting sun "under" the cloud layer.

Figure 7 shows specially filtered "cirrus" noise to model the icy media, and scale it along the desired wind direction. Because they sit at or above the tropopause, cirrus clouds don't exhibit the interesting convection motion of cumulus clouds. This simplifies animation to translation across the sky.

7. Conclusion

We have demonstrated an interactive system for artistically modeling, animating and rendering visually convincing clouds using low-cost PC graphics hardware. Using a procedural-based two-level approach for modeling and animation creates a more intuitive system for artists and animators and allows the designers to interact with their full cloud models at interactive rates. The interactive cloudscapes created with this system can be included in interactive applications or can be exported for offline photorealistic high-resolution rendering.

8. Future Work

Our renderer design sought performance before accuracy. The lighting model is a simple shadowing model performed at the vertex level without the contributions of the detailed noise effects. Since clouds are amorphous, this is sufficient to create approximate clouds for interactive applications and to convey to the artist the approximate look of off-line higher quality rendering. We plan to utilize the extensive texture mapping possible in a single pass on the new graphics hardware to enable us to perform high-quality volumetric noise and interactive physics-based atmospheric illumination, shadowing, and translucency per fragment [14]. We also plan to optimize the placement of our slicing geometry based on the location of the implicits and their projected screen area.

We are also exploring the use of our system to visualize atmospheric volume data. Various values of the volume may be interpreted as cloud attributes in the current system. For example, the wind field might be mapped to texture transformation, and moisture to its opacity.

The particle system might serve as a basis for future simulation. Commercial modeling tools use a variety of particle techniques to emulate fluids which, combined with our system, may produce an effective method to integrate clouds into a scene.

9. Acknowledgments

We wish to thank Bret Alferi and Scott Meador for insight into the artistic process. This material is based upon work supported by the National Science Foundation under Grants: NSF ACI-0222675, NSF ACI-0081581, NSF ACI-0121288, NSF IIS-0098443, NSF ACI-9978032, NSF MRI-9977218, NSF ACR-9978099, and the DOE VIEWS program.

References

1. T. Nishita and Y. Dobashi. Modeling and rendering methods of clouds. *Pacific Graphics 99*, 1999.
2. J. Stam. Interacting with smoke and fire in real time. *Communications of the ACM*, 43(7), 2000.

3. A. Lamorlette. *Siggraph Course Notes CD-ROM. 'Shrek': The Story Behind the Screen (Course 19)*. ACM-Press, 2001.
4. A. Lamorlette and N. Foster. Structural modeling of natural flames. *ACM Transactions on Graphics*, 21(3), July 2002.
5. D. Ebert, F. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 3 edition, 2002.
6. R. Voss. Random fractal forgeries. In R. A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics*. Springer-Verlag, 1985.
7. G. Gardner. Visual simulation of clouds. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, volume 19, July 1985.
8. K. Perlin. An image synthesizer. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, volume 19, July 1985.
9. F. Neyret. Qualitative simulation of convective cloud formation and evolution. In *Eurographics Workshop on Animation and Simulation '97*, September 1997.
10. Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000.
11. Y. Dobashi, T. Nishita, H. Yamashita, and T. Okita. Modeling of clouds from satellite images using metaballs. *Pacific Graphics 98*, 1998.
12. M. Harris and A. Lastra. Real-time cloud rendering. In *EG 2001 Proceedings*, volume 20(3). Blackwell Publishing, 2001.
13. P. Elinas and W. Stürzlinger. Real-time rendering of 3d clouds. *Journal of Graphics Tools*, 5(4), 2000.
14. J. Kniss, S. Premoze, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling. In *Proceedings of the conference on Visualization '02*. IEEE Press, 2002.
15. J. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, 1982.
16. J. Kajiya and B. von Herzen. Ray tracing volume densities. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984.
17. N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2), 1995.
18. J. Stam. Multiple scattering as a diffusion process. *Eurographics Rendering Workshop 1995*, June 1995.
19. J. Day. *The Book of Clouds*. Silver Lining Books, 2002.
20. R. Rogers and M. Yau. *A Short Course in Cloud Physics*. Butterworth-Heinemann, 3 edition, 1996.
21. K. Perlin and F. Neyret. Flow noise. *Siggraph Technical Sketches and Applications*, Aug 2001.
22. B. Wyvill, C. McPheeters, and G. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4), 1986.
23. K.Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press, 2002.
24. *OpenGL ARB_fragment_program Specification*, twenty-fourth edition, January 2003.
25. S. Green. 3D procedural texturing in OpenGL, 2000. NVIDIA NVSDK Repository, located at http://developer.nvidia.com.
26. B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on Volume visualization*. ACM Press, 1994.
27. J. Owens, W. Dally, U. Kapasi, S. Rixner, P. Mattson, and B. Mowery. Polygon rendering on a stream architecture. In *2000 SIGGRAPH / Eurographics Workshop on Graphics Hardware*. ACM SIGGRAPH / Eurographics / ACM Press, August 2000.
28. P. Rademacher. *GLUI, A GLUT-Based User Interface Library*, second edition, June 1999.
29. R. S. Scorer. *Clouds of the World*. Lothian, 1972.

Figure 3: *A cumulostratus layer*

Figure 4: *Detail of cumulostratus layer*

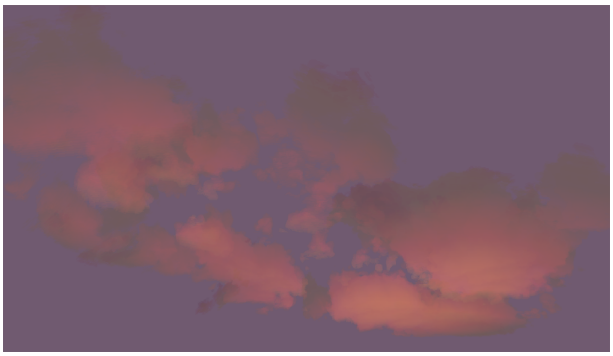

Figure 6: *A stratus cloud at sunset*

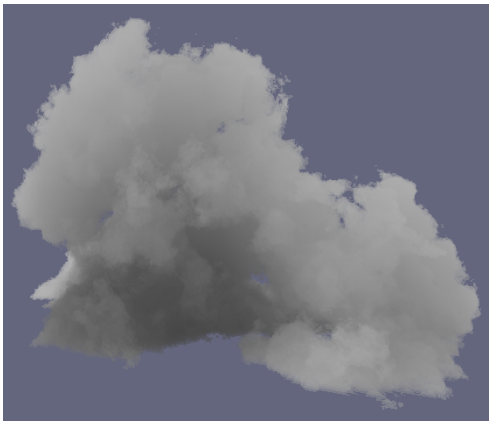

Figure 5: *A developing cumulus*

Figure 7: *A cirrus plane*