A STUDY OF SIMULATION AND REPRESENTATION METHODS OF DUST CLOUDS FOR REAL-TIME GRAPHICS APPLICATIONS

David Carr[1], Jim Geyer[1]

[1]Charles Sturt University

ABSTRACT: In 3D computer graphics applications, it is often desirable to simulate and visualise the effects of various natural phenomena, such as clouds of dust and smoke. To date, the techniques used in most real-time graphics applications (eg. computer games) for modelling such clouds centre around the use of flat, pre-drawn sprites, layered onto the screen as 'billboards'. While this can produce an adequate effect, it is not a very realistic representation. We develop and compare some alternative approaches to modelling and rendering a dust cloud, including a particle system approach, and volumetric representations based on a voxel grid and an octree space division method. These are compared on grounds of aesthetic results and performance of the algorithm, and a consideration of the breadth of possibilities for further development and enhancement.

INTRODUCTION

As computers and their graphics systems become faster, in many virtual environments and simulation applications, such as computer games, it is often desirable to simulate many natural phenomena in real-time. To date however, clouds of airborne particles, such as dust and smoke, have not been simulated accurately in these applications, and there is a lack of published modelling and simulation techniques for realistic dust behaviour [CFW00].

This paper presents a preliminary investigation of some methods that are or might be used for representing the body of a dust cloud (as generated by a moving vehicle on an unpaved surface) in a computer graphics application, and rendering the results to the screen. Three different approaches were experimented with: a simple particle system where the cloud is rendered utilising *billboarded* polygons; a more complex particle system that corresponds intuitively to real particles in the dust cloud; and volumetric systems that attempted to represent and render the volume of the cloud.

TESTBED

To run the models in a practical setting, a computer program was developed that displayed a vehicle on an empty plain (scale size 1 sq. kilometre). The user interacts with the program via the keyboard to 'drive' the vehicle around, and manipulate the camera to view the 3D scene. The programming language used is C++ for the Win32 platform, using the DirectX 9 API for 3D graphics operations. The programs were tested on two different PCs running Microsoft Windows XP: a Pentium 4 2.0GHz with an NVIDIA GeForce4 Ti4400 graphics card and a Pentium 4 1.3GHz with NVIDIA GeForce3 graphics card (all figures quoted in main text are from the first system). The core program was structured according to an object-oriented paradigm, so that alternate dust models could be coded as classes and dropped-in to the basic framework; in its standard, un-specialised guise, the core program was capable of running at 450+ frames per second on the P4 2.0GHz/GeForce4 system.

BILLBOARDED SYSTEM (SIMPLE PARTICLE SYSTEM)

In 3D computer graphics applications, a *billboard* object is one that is rotated to always face the camera (the user's viewpoint) when rendering. They are typically used to display a 2-dimensional image texture-mapped to a rectangular polygon surface; this polygon is then rotated to always face the viewer. Billboards have long been used in real-time graphics applications and computer games as a fast and simple means of rendering visual effects that would be too complex using other methods – including clouds of dust or smoke. The typical approach is to use a pre-drawn image representing a 'puff' or small cloud of dust for each billboard, making use of transparency and alpha-blending for realism. Multiples of these billboards can then be scaled and arranged into a variety of configurations, to produce clouds of any size and shape that also move and change over time.

Figure 1. View of the basic application display. (This exact screenshot is of the application employing the volumetric octree dust system, hence some specialised on-screen labels; the 'core' program described omits these but is otherwise identical in general appearance.)

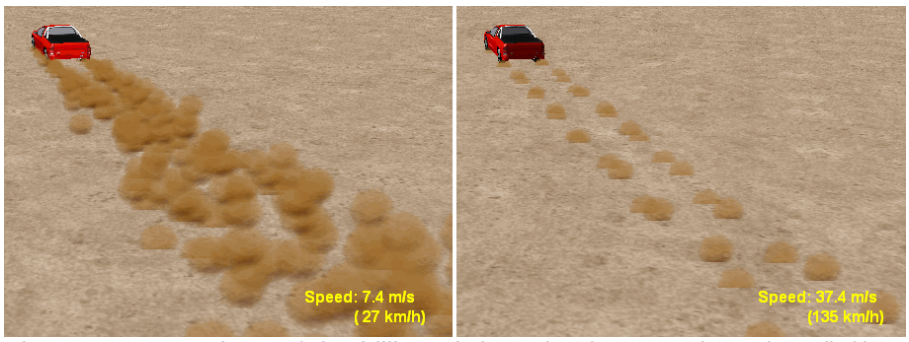

Figure 2. Screenshots of the billboard dust cloud system in action: (left) at 27km/h simulated car speed; (right) at 135km/h simulated car speed.

For our application, we implemented a billboard method of dust rendering using a simple particle system, consisting of 'emitters' (the sources of the particles), and the particles themselves. The particles have attributes including current location, a random velocity, and age; a particle remains active in the system until its age reaches a specified limit. Four emitters follow the locations of the vehicle's tyres, spawning new particles into the system as the car moves; the cloud is rendered simply by drawing a textured billboard at the location of every particle. A quicksort algorithm was used to perform depth-sorting on the billboards to prevent *artefacts* occurring during rendering, and a quick cull test was included to discard those billboards that are behind the camera.

Although this was not a highly sophisticated or polished implementation of the billboard-cloud method, the visual results are consistent with the effects seen in most games; performance-wise the application consistently performed above 200 fps at the maximum particle count. The screenshots in figure 2 demonstrate the visual drawbacks inherent in the billboard cloud system. In the low-speed case, the billboards are clustered together in a way that suggests a single cloud, though it can still be identified as being made up of many instances of the same circular puff texture. However, in the high-speed case they are strung out over a long line, and give a much less convincing appearance. Both renderings also exhibit the occlusion of the billboard polygons intersecting with the ground surface, which further highlights the individual puffs and degrades the visual effect of the cloud.

LITERAL PARTICLE SYSTEM

The previous application used a particle system to represent the positions of billboards. From this we may suppose we could obtain more realistic results by using more particles in a finer-grained system to correspond more closely to the individual particles of real cloud. (Hence the term 'literal' used here to name the model, as the particle system is drawn directly to represent the dust cloud.) For this we implemented a system based very loosely on the model developed by [CFW00]. Again in this model, we use four emitters at the vehicle wheel locations; to ensure a smooth stream of new particles under low frame rates, initial positions are interpolated between the emitters' frame-by-frame locations. Additionally, particles also now have a colour component, used to render each as a point on the screen; as the particle ages, the alpha-blending component of its colour value is modified to make it fade away. Although we avoided the detail and actual physics presented by [CFW00], some simple algorithms were added to (roughly) represent airflow, by modifying the velocities of particles in close proximity behind the moving car in the direction of travel, relative to the car's present speed.

Particles were spawned from the emitters at a rate scaled by the car's speed; at maximum capacity it produces a cloud made up of approximately 36,000 particles. Performance at idle was a reduced 390fps; at two-thirds of capacity (about 24,000 particles) it ran just below 160fps, and at the maximum it ran at around 120fps. A limitation of rendering the particle cloud as points on the screen, is that the cloud does not appear the same when viewed from different distances: with the viewpoint close or within the particle cloud, the volume of the cloud occupies a large area of the screen and the tiny particles are drawn widely spaced out; conversely, as the viewpoint is moved away, the particles are drawn into a smaller area of the screen, making the cloud appear denser. An attempt was made to implement a culling algorithm to scale the number of particles rendered relative to the view distance; however, the calculation to obtain distance of each particle and decide whether to keep it in the drawable set proved too costly – almost exactly halving performance – and so was not used.

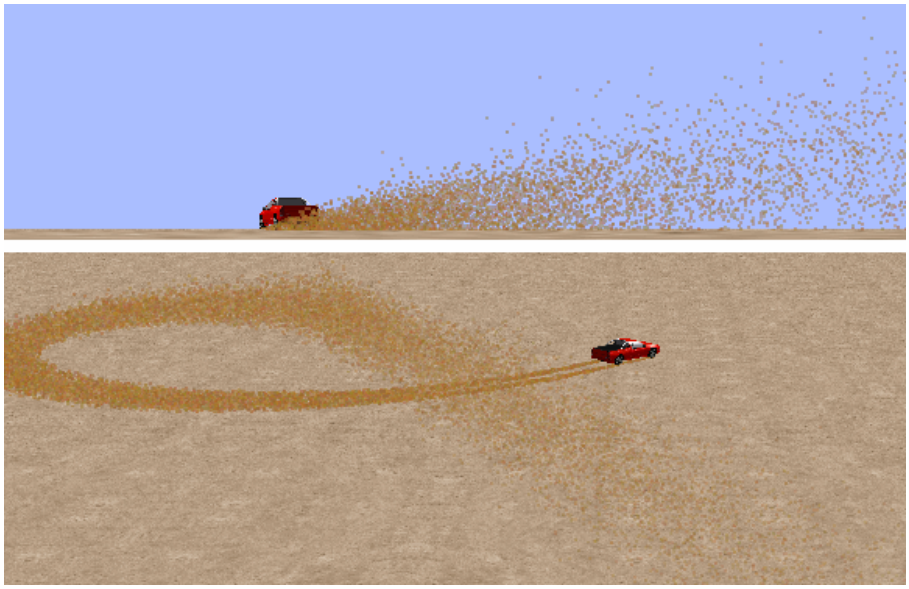

Figure 3. Screenshots of the particle dust system in action: (top) cruising at 29km/h simulated car speed; (bottom) drifting particles have their velocities perturbed as the vehicle crosses back through the cloud while accelerating.

VOXEL-BASED VOLUMETRIC SYSTEM

The attempt here was to start a new line of development by modelling the dust cloud as a volume, rather than base the whole representation on a particle system. A three-dimensional grid of cubic volume elements (voxels) is created over the terrain, each voxel representing the density of airborne dust at that location. For simplicity at the preliminary stage, the individual voxel cells were rendered as coloured and alpha-blended (untextured) square billboards; depth-sorting of the billboards was never implemented. The first problem to be encountered was the exceptionally vast amount of memory required for even a simple voxel grid; for this sample application, we restricted to voxels of size 0.5 metres, covering only a relatively small area – 200m square – at the centre of the terrain.

To make a cloud spread throughout the voxel grid, a greatly simplified method of having the value at one voxel forming the inputs of its neighbours, inspired by [MDCN03], was employed. As the vehicle moves around the world, the voxels corresponding to the locations of its wheels have their density value increased relative to its speed; subsequently, each active voxel (those with a density value above a certain threshold) would 'bleed' density into its neighbours so that the density 'cloud' will grow and disperse over time. This simple premise led to a protracted juggling act trying to find scaling factors to make the cloud grow into new voxels at a pleasing rate, yet not lose stability; the result was to use a fade rate slightly higher than six times that of the dispersing rate.

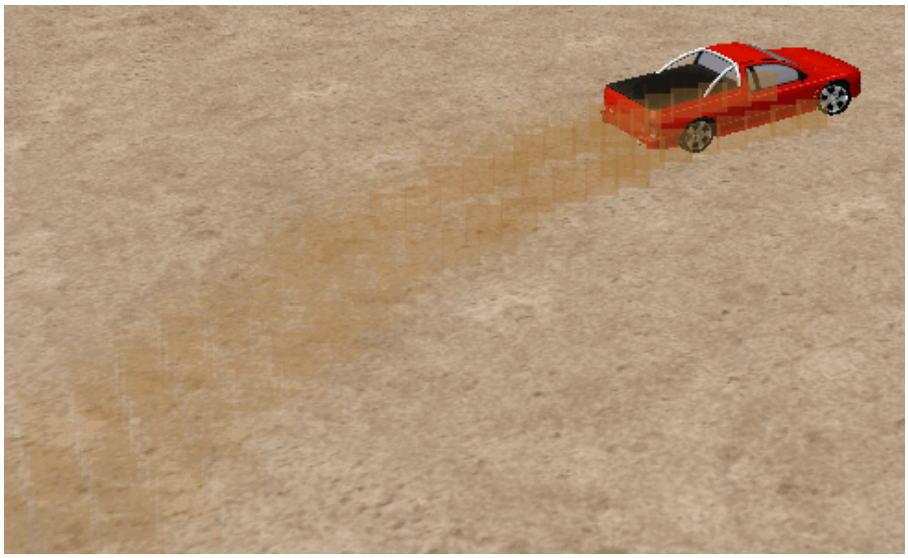

Figure 4. Screenshot of the voxel dust system in action, at 52km/h simulated car speed.

This led to finding a second major drawback of the model: because each active voxel can affect up to six of its neighbours, as the number of active voxels increases, so the amount of computation rises roughly sixfold. From an idle 450+ fps, the performance of the application drops sharply to around 120fps when 500 voxels are being calculated, and 60fps for 1000 voxels; dropping below 25fps when the number of voxels reached a recorded maximum of about 2400.

OCTREE-BASED VOLUMETRIC SYSTEMS

Given the space constraint problems posed by trying to work with a 'dumb' voxel grid, the use of an octree structure was investigated as a 'smarter' means of partitioning the world space. Given a 3-dimensional volume, an octree subdivides along the three axes to produce eight cubes; any or all of those cubes may themselves be subdivided into a further eight cubes, and so on down to the required depth of the tree [Hum01]. Because an octree structure only describes those voxels that make up the shape we are interested in, they are very space efficient when most of our volume space remains empty [Sto03].

Octree System 1 – Recursive Construction

For this application, a particle system was chosen to model the underlying behaviour of the dust cloud; since the octree is used to describe the volume of the cloud, the particle system is relatively low-density but otherwise works in the same way as those used in the previous models. The octree is able to cover the entire 1km square terrain piece, and does so to a depth of 12 levels, meaning the voxels at the bottom level have cubic dimensions of approximately 0.25 metres. Because the dust cloud is not static, the octree must be reconstructed every frame cycle; this is done by a recursive process of testing the location of each particle and assigning them to lists down the levels of the tree. As particles age, this recursion stops at larger-volume cells progressively higher up the tree, representing the expansion of the cloud over time. For simplicity, the octree cells were rendered as textured billboards; the billboard list was built up at the same time the octree is constructed (and implicitly, traversed). The opacity of a cell (or rather that of its attendant billboard) is varied according to the age of the particle(s) it contains to fade away over time.

Dividing the world into this hierarchy of cubes presupposes that we may cunningly use this to perform our depth-ordering of billboards by traversing the octree according to a particular order or algorithm [Ste96]; however, this turned out to be more complicated than initially thought (due to variant viewing angles within the octree space), and was never fully implemented in the application.

Because the main work of this application centres on constructing the octree representing the volume of the dust cloud, the performance of the system varies as to the amount of ground covered by the cloud. At idle, with the vehicle stationary and no particles in the system, the application ran at 450+ fps. With the vehicle travelling at 25km/h, it produced a short cloud behind it of approximately 180 particles and utilising 170 nodes in the octree, and the frame rate was an indicated 260 fps. At 50km/h it produced a cloud twice as long, using approximately 350 particles / 330 nodes for 180fps; at 100km/h it used 500 particles / 600 nodes for 130fps; and at its maximum speed of 180km/h, the cloud several hundred metres in length used approximately 450 particles / 700 nodes, at 120fps.

The resultant effect of the system (albeit without proper depth-buffering) was pleasing. Because the voxels occupy discrete positions in the world space, this can lead to some ugly 'popping' of billboards between cells, rather than the smooth motion described by the particles. However, the octree system creates more cohesive cloud volume than either of the earlier billboard or particle-oriented systems.

Octree System 2 – Search-Principle Construction

It is a feature of the octree structure that the coordinate location of any voxel is implicit from its position in the octree (and vice versa), giving octrees interesting properties for searching [Sto03]. In short, we can find where a given voxel belongs in the tree by simple bitwise operations on its coordinate values. Once this principle was understood, it was used in the application as the basis of a faster octree-construction algorithm with the hope of making a faster overall system. However, this change required the use of an additional traverse loop to build up the billboard list; this tended to cancel out any gain, and overall performance (and effect) were found to be roughly the same between the two versions.

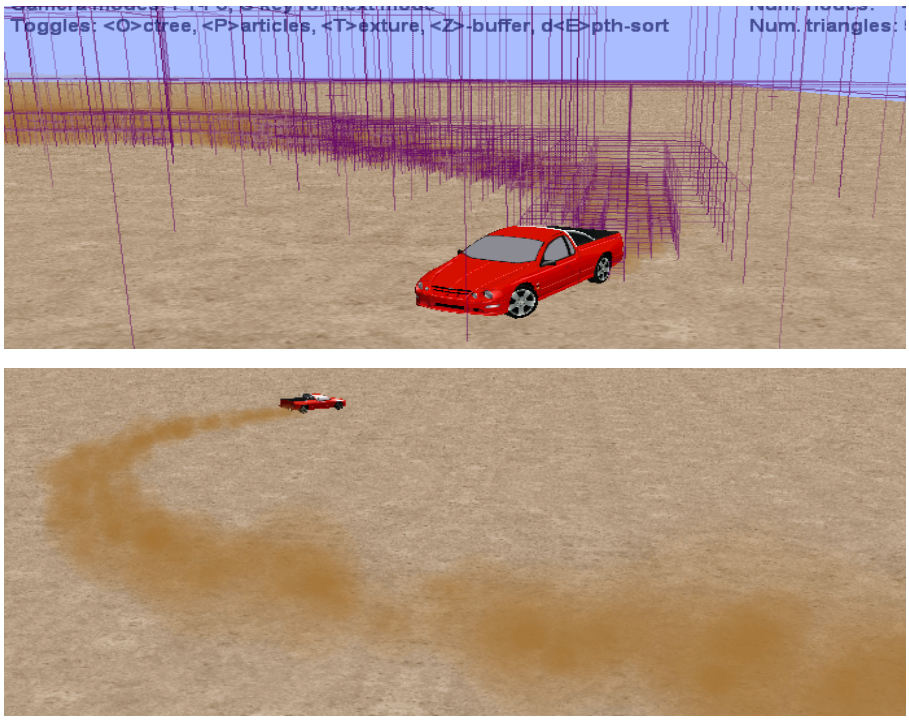

Figure 5. (Top) Screenshot showing a representation of the octree structure describing the dust cloud volume. (Bottom) Screenshot of the octree dust system in action at 88km/h simulated car speed.

CONCLUSION AND FUTURE WORK

The compiled performance data (frames per second) of the applications is presented in Table 1.

Table 1. Comparison of application performance on the two test systems.

Performance (FPS: low–high)	P4 2.0Ghz / GeForce4	P4 1.3Ghz / GeForce3	Est. Complexity of system
Core (Reference)	450+ fps	240 fps	N/A
Billboard system	200 – 400+ fps	100 – 240 fps	O(n) (linear)
Particle system	120 – 400 fps	85 – 240 fps	O(n)
Voxel system	25 – 450 fps	15 – 230 fps	$O(n^6)$ (exp.)
Octree system v1	120 – 450+ fps	100 – 235 fps	O(n)
Octree system v2	100 – 450+ fps	75 – 235 fps	O(n)

Billboards based on simple particle systems have been widely (almost exclusively) used for representing cloud effects, etc. in games because of their simplicity of implementation and fast processing. With tuning and photorealistic textures they can provide a very good effect, though they still have inherent drawbacks that manifest because each billboard acts as (and in reality is) an independent entity from the rest that make up a cloud. Literal particle systems are useful for more correctly simulating a particle cloud, but are usually too computationally-intensive to produce a good effect and still be workable in a real-time environment.

With the volumetric systems described, an attempt was made to sever the reliance on the position and movement of independent particles forming the shape of a dust cloud, instead representing the cloud as a volume. In doing so we investigated the use of an octree structure to store the volume elements in a space-efficient and ultimately time-efficient way. Proposed further work is to refine the octree system to produce a higher-quality result. One step may be to apply some of the physics described by [CFW00] to make the system more behaviourally accurate; another is to apply rendering techniques such as those developed for (rain) clouds to improve the visual appearance (eg. [ND01], [Har02]). Ultimately, the voxel cloud is still being rendered as a collection of billboards, and there could be improvement in finding an alternate way of drawing the voxels.

REFERENCES

[CFW00] Jim X. Chen, Xiaodong Fu, and Edward J. Wegman. Real-Time Simulation of Dust Behaviour Generated by a Fast Travelling Vehicle. *ACM Transactions on Modeling and Computer Simulation,* Vol. 9, No. 2, Pages 81–104. 1999.

[Har02] Mark J. Harris. Real-Time Cloud Rendering For Games. *Proceedings of Game Developers Conference 2002.* 2002.

[Hum01] Ben Humphrey. Octree Tutorial. http://www.gametutorials.com/Tutorials/OpenGL/Octree.htm, www.GameTutorials.com. Accessed 20 August 2004. 2001.

[MDCN03] Ryoichi Mizuno, Yoshinori Dobashi, Bing-Yu Chen, and Tomoyuki Nishita. Physics Motivated Modeling of Volcanic Clouds as a Two Fluids Model. *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG'03)*, IEEE Computer Society. 2003.

[ND01] Tomoyuki Nishita and Yoshinori Dobashi. Modeling and Rendering of Various Natural Phenomena Consisting of Particles. IEEE Computer Society. 2001.

[Ste96] Alexander Stevenson. Voxels and Volumetric Representation. http://www3.telus.net/ah/voxels/voxels.htm. Accessed 20 August 2004. 1996.

[Sto03] Nilo Stolte. Octree – Overview. http://nilo.stolte.free.fr/octree.html. Accessed 20 August 2004. 2003.