VOT 74079

THE DEVELOPMENT OF CLOUD MODELLING AND MOTION ANALYSIS FOR VIRTUAL ENVIRONMENT

(PEMBANGUNAN PERMODELAN AWAN DAN ANALISA PERGERAKAN UNTUK PERSEKITARAN MAYA)

MOHD SHAHRIZAL SUNAR
DAUT DAMAN
SARUDIN KARI
NORHAIDA MOHD SUAIB
ABDULLAH BADE

RESEARCH VOTE NO:
74079

JABATAN GRAFIK KOMPUTER DAN MULTIMEDIA
FAKULTI SAINS KOMPUTER DAN SISTEM MAKLUMAT
UNIVERSITI TEKNOLOGI MALAYSIA

2005

UNIVERSITI TEKNOLOGI MALAYSIA

BORANG PENGESAHAN
LAPORAN AKHIR PENYELIDIKAN

TAJUK PROJEK : **THE DEVELOPMENT OF CLOUD MODELLING AND MOTION ANALYSIS FOR VIRTUAL ENVIRONMENT**

Saya **MOHD SHAHRIZAL BIN SUNAR**
(HURUF BESAR)

mengaku membenarkan Laporan Akhir Penyelidikan ini disimpan di Perpustakaan Universiti Teknologi Malaysia dengan syarat-syarat kegunaan seperti berikut:

1. Laporan Akhir Penyelidikan adalah hak milik Universiti Teknologi Malaysia.
2. Perpustakaan Universiti Teknologi Malaysia dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan Laporan Akhir Penyelidikan ini bagi kategori **TIDAK TERHAD**.
4. *Sila tandakan (✓)

☐ SULIT (Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)

☐ TERHAD (Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)

☑ TIDAK TERHAD

_____
(TANDATANGAN KETUA PROJEK)

Nama & Cop Ketua Penyelidik
Tarikh: 15/11/2005

ABSTRACT

Modelling the natural phenomena such as clouds is one of the most challenging problems in computer graphics. The complexity of cloud formation, dynamics and light interaction makes real time cloud modelling a difficult task. The visual portrayal of the sky and cloud is a common requirement when rendering the outdoor scenes in computer graphics. The traditional way to create the sense of cloudy is by using the captured sky images as background. This main output of this project is a cloud modelling editor for designing cloud shapes namely RekAwan. The editor provides integrated environment for modelling volumetric clouds with particle system and surfaced based cloud using texture. This invention uses the newly developed randomized algorithm to fill the cloud volume with particle systems. Randomized method provides an efficient mean in modelling cloud particles data very quickly and filling the cloud volume space with particles randomly. The invention gives on-the-fly control over size of particles and number of particles and radius of particles in the system. This shows its suitability for real time virtual reality applications such as flight simulator and 3D games. The user can pass through the cloud with realistic visual effect. The completed 3D cloud model output from RekAwan can be easily imported into any OpenGL based virtual environment such as simulator, animation and games. RekAwan can also become plug-ins to current commercial 3D modeller software such as AutoCAD, 3D Studio Max, Maya and Rhino3D. The potential industries that highly use RekAwan are weather visualization, film, advertisement, games and flight simulator. This invention will cut the development cost of the industry such as simulator and entertainment. This product can be used effectively by 3D artists, designers and game developer to model cloud shapes easily through interactive user interface. It is simple to use, efficient and gives extensive control over the cloud shapes. RekAwan can also be used to model any other gaseous object such as smoke, haze and fog.

ABSTRAK

Permodelan fenomena semulajadi seperti awan merupakan salah satu cabaran dalam grafik komputer. Ini disebabkan oleh bentuknya yang kompleks, dinamik dan interaksi cahaya diantara setiap partikel yang terdapat di dalamnya. Awan dan langit merupakan perkara yang asas sebagai latar belakang dalam proses menghasilkan suasana persekitaran luaran. Kaedah tradisional dalam menghasilkan suasana berlatarbelakangkan awan dalam persekitaran maya ialah dengan menggunakan imej yang diambil dengan kamera. Output utama dalam projek penyelidikan ini ialah satu editor untuk permodelan awan bagi merekabentuk bentuk awan yang dinamakan RekAwan. Editor ini menggabungkan kaedah permodelan awan secara isipadu dengan penggunaan partikel dan juga kaedah permukaan menggunakan tekstur. Inovasi ini juga menghasilkan algoritma perawakan untuk memenuhi partikel bagi sesuatu bentuk awan. Algoritma ini membolehkan proses permodelan awan dilaksanakan dengan lebih laju dan pengisian isipadu awan dilakukan secara rawak. Pengguna dibenarkan untuk mengawal saiz dan bilangan partikel dalam sistem. Ini menunjukkan keserasiannya dengan aplikasi realiti maya masa nyata seperti simulator penerbangan dan permainan komputer 3D. Pengguna juga boleh menembusi awan dengan kesan yang realistik. Model 3D yang dihasikan oleh RekAwan boleh diimport ke mana-mana persekitaran maya yang berasaskan OpenGL seperti simulator, animasi dan permainan komputer. RekAwan juga boleh dijadikan sebagai plig-ins kepada perisian permodelan 3D komersil yang ada di pasaran kini seperti AutoCAD, 3D Studio Max, Maya dan Rhina3D. Industri yang berpotentsi untuk menggunakan RekAwan termasuklah visualisasi cuaca, filem, pengiklanan, permainan komputer dan simulator penerbangan. Hasil inovasi ini dapat mengurangkan kos pembangunan seperti simulator dan juga hiburan. Hasil produk ini juga boleh digunakan secara efektif oleh artis 3D, perekabentuk dan pembangun permainan komputer untuk memodelkan bentuk awan dengan lebih mudah menerusi antaramuka yang interaktif dan mudah. RekAwan juga boleh digunakan untuk memodel sebarang objek berasaskan gas seperti asap, jerebu dan kabus.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	DESCRIPTION
AGP	Accelerated Graphics Port
API	Application Programming Interface
CA	Cellular Automation
CML	Coupled Map Lattice
CSG	Constructive Solid Geometry
GPU	Graphical Processing Unit
HMD	Head Mounted Display
PDE	Partial differential Equation
VE	Virtual Environment
VR	Virtual Reality

LIST OF APPENDICES

CHAPTER I

INTRODUCTION

1.1 Introduction

If clouds were the mere result of the condensation of vapor in the masses of atmosphere which they occupy, if their variations were produced by the movements of the atmosphere alone, then indeed might the study of them be deemed an useless pursuit of shadows, an attempt to describe forms which, being the sport of winds, must be ever varying, and therefore not to be defined. But the case is not so with clouds.

So began Luke Howard, the “Godfather of the Clouds”, in his ground breaking 1802 essay on the classification of the forms of clouds (Howard, 1804). Howard’s classification system—most noted for its three main classes *cirrus, stratus,* and *cumulus*—is still in use today, and is well-known even among lay people. Howard’s work and its influence on the world exemplify the importance of clouds to humankind. Long before his time, people had looked to the clouds as harbingers of changing weather, but Howard knew that understanding and predicting changes in the weather required a better understanding of clouds. This understanding could not be improved without a concrete yet flexible nomenclature with which clouds could be discussed among scientists. Howard’s contemporaries were immediately taken with his classification, and his fame quickly expanded outside of the circle of amateur scientists to which he presented his work.

Clouds are a frequently observed natural phenomenon. They are estimated to cover between 60 and 70 % of the globe at any given time. At most locations on earth some clouds will occur on every single day. Clouds exist in a great variety of forms and on a large range of both temporal and spatial scales. Individual small cumulus clouds for instance cover a few hundred meters in the horizontal and vertical and normally have a lifetime of less than an hour. In contrast the vast, virtually ubiquitous stratocumulus decks covering the eastern parts of the subtropical oceans have a horizontal extent of several hundred kilometers, while being no more than a few hundred meters thick. The processes involved in the formation and dissipation of clouds span an even larger range of scales from micrometers for the condensation of individual droplets to thousands of kilometers for cloud formation in frontal systems associated with mid-latitude baroclinic systems (Christian, 2000).

An analysis of satellite observations shows that about half of the earth's clouds extend above the freezing level and, therefore, are capable of ice production. The clouds, however, do not glaciate instantly as they are exposed to negative temperatures. Mixed phase clouds are commonly observed at temperatures down to -20°C and below. Nucleation of ice crystals in clouds may occur either by homogeneous freezing of drops or by heterogeneous nucleation on ice nuclei. The former process is believed to be important at temperatures below about -40°C. This process is essential for cirrus formation but its effects may be neglected for most low and middle tropospheric clouds in mid-latitudes (Mikhail, 1997).

Clouds are directly linked to a large variety of weather phenomena. Rain and snow are obviously produced in clouds, as are thunder and lightning. The latent heat release due to condensation processes is known to be one of the most important processes in the spin up and maintenance of tropical storms, which appear in their most violent form as hurricanes and typhoons. It is an everyday experience that clouds influence the radiative fluxes emitted both by the sun and the earth. If clouds form on a sunny day, the maximum temperature near the surface will be lower than without them, a direct consequence of the reflection of sunlight by clouds. Likewise, if low clouds cover the sky at night the near-surface temperature will not drop as low as under clear sky conditions due to the trapping of terrestrial radiation by the clouds. Because of all these reasons it is obvious that it is desirable for any form of weather

forecast to include a prediction of the occurrence and type of clouds and precipitation. Just as importantly, the desire to estimate the future evolution of our planet's climate requires knowledge about clouds. This is due to their strong interaction with the radiative fluxes whose modification through changes in the atmospheric composition is of considerable concern.

Clouds are a ubiquitous feature of our world. They provide a fascinating dynamic backdrop to the outdoors, creating an endless array of formations and patterns. As with stars, observers often attribute fanciful creatures to the shapes they form, but this game is endless, because unlike constellations, cloud shapes change within minutes. Beyond their visual fascination, clouds are also an integral factor in Earth’s weather systems. Clouds are the vessels from which rain pours, and the shade they provide can cause temperature changes below. The vicissitudes of temperature and humidity that create clouds also result in tempestuous winds and storms. Their stunning beauty, physical and visual complexity, and pertinence to weather have made clouds an important area of study for meteorology, physics, art, and computer graphics (Harris, 2003).

Cloud realism is especially important to flight simulation. Nearly all pilots these days spend time training in flight simulators. To John Wojnaroski, a former USAF fighter pilot and an active developer of the open-source FlightGear Flight Simulator project (FlightGear, 2003), realistic clouds are an important part of flight that is missing from current professional simulators.

One sensation that clouds provide is the sense of motion, both in the simulation and in real life. Not only are clouds important, they are absolutely essential to give the sky substance. Like snowflakes, no two clouds are alike and when you talk to folks involved in soaring you realize that clouds are the fingerprints that tell you what the air is doing.

The complexity of cloud formation, dynamics, and light interaction makes cloud simulation and rendering difficult in real time. In an interactive flight simulation, users would like to fly in and around realistic, volumetric clouds, and to see other aircraft convincingly pass within and behind them. Ideally, simulated

clouds would grow and disperse as real clouds do, get blown by the wind, and move in response to forces induced by passing aircraft. Simulated clouds should be realistically illuminated by direct sunlight, internal scattering, and reflections from the sky and the earth below. Previous real-time techniques have not provided users with such experiences.

Clouds have fascinated and vexed computer graphics researchers for many years. The visual appearance of clouds is very complex and extremely varied, yet it is very easy to recognize an "incorrect" cloud model, probably because we see clouds in one form or another every day (Gustav, 1999).

1.2 Motivation

Clouds remain one of the most significant challenges in the area of modeling natural phenomena for computer graphics and it has been a challenge for nearly twenty years. This has made the simulation of various natural phenomena one of the important research fields in computer graphics. Aspects such as sky, clouds, water, fire, trees, smoke, terrains, desert scenes, snow and fog play an important role for creating realistic images of natural scenes. In particular, clouds are indispensable for creating realistic images of natural scenes, outdoor scenes, flight simulators, space flight simulators, visualization of the weather information, creation of realistic clouds from satellite images, simulation of surveys of the earth, earth viewed from outer space, film, art and so on. Some of the motivations factors can be summarized as following:

- Clouds are familiar objects in everyday life, it is desirable to simulate them effectively for the applications, such as entertainment, advertising, and art etc.

- Clouds are a critical element in air-to-air combat and are important in the simulation of intelligent weapon systems, which seek and identify aerial targets in cluttered backgrounds.

- Realistic cloud simulation would also be an effective tool in the field of meteorology.

- Indeed, it is reasonable to say that we want to simulate these beautiful natural features simply because they are there.

1.3 Problem Background

There are two important issues for synthesizing photo-realistic images of all natural phenomena including clouds. These are modeling and rendering. Generally, the modeling process includes the creation of shapes of objects, their dynamics (motion/movement) and their physical properties such as surface reflectance. This is however not an easy task for objects such as clouds, smoke and sand dunes (Nishita and Dobashi, 2001). Rendering is the process of generating images by calculating colors for every pixel. The ray-tracing algorithm is often employed for generating images including the sky, clouds, smoke, desert scenes, and the atmospheric effects. Although the ray-tracing algorithm can create extremely realistic images, the computation time is very long.

There are two main approaches to cloud modeling and rendering. These two approaches classified as procedural techniques and physically based techniques. Procedural techniques try to capture the visual appearance of clouds without simulating the actual physical processes. Voss (1983) and Musgrave (1990) created realistic clouds with fractals. Gardner (1985) has produced realistic images of clouds by using Fourier synthesis. However, this does not create a true three-dimensional geometric model. Ebert and Parent (1990) have used solid texture to generate clouds in three-dimensional spaces. Ebert (1997) has also developed a method combining metaballs and a noise function to model clouds. Sakas (1993) has modeled clouds by using spectral synthesis and Nishita *et. al* (1996) created clouds by generating metaballs using the idea of fractals. Using these methods to create clouds is, however, very difficult since many parameters have to be specified by trial and error.

Stam and Fiume (1991) have developed a simple method for modeling clouds. In their method, a user specifies density values at several points in three-dimensional space. Then the density distribution of the clouds is obtained by interpolating the specified density values. Although this method can create realistic clouds, it is impractical for creating large-scale clouds viewed from space. Methods for modeling a set of clusters of clouds have also been developed. Ebert *et al*. (1998) created realistic images of typhoons by the procedural approach. Nishita *et al*. (1996) have modeled clouds to generate realistic images of the earth viewed from space. In both these methods, however, clouds are simply modeled by applying two-dimensional fractals. The color and shape of clouds change depending on both the viewpoint and the position of the sun. These methods cannot simulate such effects.

The physically based techniques attempt to simulate the meteorological processes that create clouds and the interaction between light and cloudy air (Kajiya and Herzen, 1984; Stam and Fiume, 1991, 1993; Harris *et al*., 2003).Kajiya and Herzen (1984) used a simple method based on Partial Differential Equations (PDE) to generate cloud data sets for their ray-tracing algorithm. Dobashi *et al*. (2000) used a simple cellular automata model of cloud formation to animate clouds offline. Miyazaki *et al*. (2001) extended this to use a coupled map lattice model based on atmospheric fluid dynamics. Overby *et al*. (2002) described another physical model that, like ours, is based on the stable fluid simulation of Stam (1999). Harris *et al*. (2003) presented a method most similar to the work by Kajiya and Herzen (1984) and Overby *et al*. (2002). He implements simulation, dynamics and radiometric entirely on programmable floating-point graphics hardware to get real time simulation. It seems, however, this method is not applicable for large-scale clouds as required by games and flight simulator applications running on desktop machines.

Physical based 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, large-scale high quality simulation still exhausts commodity computational resources (Schpok *et al*., 2003). 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. 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.

Clouds consist of small particles and it is very difficult to define their definite shapes. The simulation of their dynamics (movement) is also a difficult task, since their shape changes continuously with time. Therefore, a lot of modeling methods have been developed to address this problem. Using these methods, however, obtaining realistic-looking shapes and motion is very time consuming. For example, the rendering of clouds and smoke requires the integration of the intensity of light scattered by small particles along the viewing ray. On the other hand, the processing speed of graphics hardware has become faster and faster recently. In addition, high performance graphics hardware is available even on low-end PCs. These facts have encouraged researchers to develop hardware-accelerated methods for rendering realistic images (Ofek and Rappoport, 1998; Heidrich and Seidel, 1999; Stam, 1999; Cabral *et al*., 1999).

After efficiently computing the dynamics and illumination of clouds, there remains the task of generating a cloud image. The translucent nature of clouds means that they cannot be represented as simple geometric “shells”, like the polygonal models commonly used in computer graphics. Instead, a volumetric representation must be used to capture the variations in density within the cloud. Rendering such volumetric models requires much computation at each pixel of the image. This computation can result in excessive rendering times for each frame.

In order to model realistic clouds, there are two possibilities - the first is physical-based simulation and the second is use of procedural techniques. Physical-based simulation provides a straight-forward approach to create realistic clouds by simulating the physical phenomena. This type of simulation creates a three

dimensional density map, which describes the density of the water vapor the object consists of. The density-map is rendered by means of a volume renderer, which takes into account the specific scattering of light-rays, caused by tiny water droplets inside the object. Physical-based simulation, however, demands a high computational cost and is impractical for real time applications. Even on today's fastest processors, rendering times of about a few minutes per image are common.

Procedural modeling techniques provide simple and efficient methods to simulate natural phenomena and give visually convincing results. Such techniques provide an abstraction of model, encode classes of objects, and allow high-level control and specification of the model. The goal of these modeling techniques is to provide a concise, efficient, flexible, and controllable mechanism for specifying and animating models of complex objects and natural phenomena. Code segments or algorithms are used to abstract and encode the details of the model instead of explicitly storing vast numbers of low-level primitives. The use of algorithms provides great flexibility, and allows amplification of efforts through parametric control - a few parameters to the model yield large amounts of geometric details.

Particle system is one of the procedural techniques. Particle systems are most commonly used to represent natural phenomena such as fire, water, clouds, snow, rain, grass, and trees. A particle-system object is represented by a large collection of very simple geometric particles that change stochastically over time. Particle systems do use a large database of geometric primitives to represent natural objects, but the animation, location, birth, and death of the particles representing the object are controlled algorithmically. The procedural aspect and main power of particle systems allow the specification and control of this extremely large cloud of geometric particles with very few parameters. Besides the geometric particles, a particle system has controllable stochastic particle animation procedures that govern the creation, movement, and death of the particles. These animation procedures often include physically based forces to simulate effects such as gravity, vorticity, conservation of momentum, and energy.

Clouds behave like live objects i.e., clouds move from one place to another, clouds disappear while moving, and clouds appear while moving. So clouds are very

easily modeled with the particle systems. Particle systems are a simple and efficient method for representing clouds. Cloud model assumes that a particle represents a roughly spherical volume in which a Gaussian distribution governs the density falloff from the center of the particle. Each particle is made up of a center, radius, density, and color. Good approximations of real clouds can be achieved by filling space with particles of varying size and density. Clouds can very easily be built by filling a volume with particles, or by using an editing application that allows placing particles and building clouds interactively. The randomized method is a good way to get a quick field of clouds. Virtual reality applications such as flight simulators have pre-designed levels and require fine control over all details of the scene. Providing an interactive editor allows producing beautiful clouds tailored to the needs.

1.4 Problem Statement

This research focuses on the development of a technique that can be used for synthesizing cloud images in an interactive way. Particles system is used to model these fuzzy objects.

The following research questions are addressed to solve the problem:

a) How efficient is particle system with randomized method for synthesizing cloud images?

b) How can interactive-ness be employed for cloud modeling?

1.5 Objectives of the Study

The main objective of this research is development of a shape modeling algorithm using particles system and an interactive editing application in the area of

cloud shape modeling problem. This research also aims to achieve the following objectives:

a) To investigate, analyze and formulate an appropriate technique for modeling.

b) To define an appropriate mathematical model for the deformation of the physically based cloud motion.

c) To construct a software library for modeling the cloud with the motion analysis.

1.6 Importance of the Study

This study is conducted particularly to construct a randomized based model in solving cloud shape modeling problem. In general, this study introduces a technique consisting of a randomized based algorithm by making use of particles system for modeling cloud shapes and an interactive editor application – cloud macrostructure editor.

The whole research can be divided into two parts. The first part of the research consists on development of a randomized based algorithm using particles system that can fill cloud space by placing particles at random positions to model microstructure of the clouds. The second part deals with the development of an interactive editor application – **cloud macrostructure editor,** which can be used to model apparent shapes of clouds interactively.

Results of this study can be used for conducting comparative study in the future in order to discover whether the proposed system is suitable for different problems in the area of cloud shape modeling problem.

1.7 Scope of the Study

The scope of the research conducted in this study can be summarized as follows:

a) In a real system, the number of particles keeps on changing with the passage of time. Some particles die from the system and at the same time some new particles are born in the system. For simplicity, the number of particles is considered as constant in this research i.e., no particle is born and no particle dies with the passage of time.

b) In reality, each particle may not resemble the other particles and particles may have different shapes. This research has considered a roughly spherical volume for each particle for simplicity.

c) Particles in a system are free to move and continuously change their position as they move along the atmosphere. In this research, each particle has a static position as this research does not focus on study of cloud dynamics. So particle do not move in any direction along the atmosphere.

1.8 Thesis Contributions

In this thesis, particles systems are used to model clouds and then synthesize cloud images by making use of proposed algorithm based on randomized method that fills the cloud volume with particles randomly. It provides an applicable platform for the efficient and interactive synthesis of cloud images.

In general, the major contribution described in this thesis can be summarized as follows:

a) Development of cloud shape modeling algorithm based on randomized method using particle systems.

b) Development of an interactive cloud editing application – cloud macrostructure editor, to model cloud shapes in an interactive way. This approach provides control over number of particles in a particular cloud area and control over size of the particles.

c) Application of cloud macrostructure editor to test interactive-ness of the randomized algorithm for modeling cloud shapes.

1.9 Thesis Organization

This section presents how this thesis is organized. The main structure of the thesis consists of introduction, literature review, methodology, design, results, discussion and conclusions in chapters described as below.

Chapter I: Introduction. This chapter introduces the research topic consisting of various sections i.e. Introduction, Motivation, Problem Background, Problem Statement, Objectives of the Research, Importance of the Study, Scope of the Study, Thesis Contributions and Thesis Organization.

Chapter II: Cloud Modeling. This chapter presents current studies in the area of cloud modeling problem and evaluates the advantages and disadvantages of the existing solutions.

Chapter III: Particle Systems. This chapter discusses the basics of the particle system.

Chapter IV: Randomized Particle Algorithm. Definition of the problem addressed in this research and model formulation are presented in this chapter. Proposed Algorithm and its testing are also presented in this chapter.

Chapter V: Results and Testing. Results of the experiments are summarized and discussed in this chapter.

Chapter VI: Discussion and Conclusions. This final chapter discusses the strength and weaknesses of the thesis. This chapter also suggests the future work that can be done for the extension of the proposed technique. Finally, this chapter concludes the work of this research.

CHAPTER II

CLOUD MODELING

2.1 Introduction

Throughout the history of computer graphics, advances have been driven by the quest for visual realism. This quest for visual realism encompassed all aspects of image creation from object definition to object color, illumination and shadowing. For example, early models for representing objects used polygonal meshes. These early polygonal models for representing objects were not used to represent abstract artistic shapes; they were model, of actual objects. Spline patch models were later used for object modeling to more accurately represent cured surfaces in the real world. Early rendering systems rendered faceted shaded, flatly illuminated, alias-prone images. Techniques were then developed by Gouraud and Phong to approximate curved surfaces from polygonal models (Rogers and David, 1985). The illumination models for these objects have also greatly improved from simple Larnbertian models to complex illumination models using radiosity, ray tracing and Cook-Torrance illumination models (Ebert, 1996).

Much effort has been made in computer graphics on the synthesis of real-world imagery. The sky is an essential part of realistic outdoor scenery. Because of this, cloud rendering has been an active area of research in computer graphics for the past twenty years.

A survey of previous work on clouds would be incomplete without a description of the variety of methods that have been used in computer graphics to synthesize the images of various natural phenomena including clouds. The rest of this introductory chapter will review advances in creating state-of-the-art images in computer graphics. The use of texture mapping techniques for modeling the surface attributes of objects will be discussed, followed by a discussion on various techniques used for cloud modeling and rendering.

2.2 Texture Mapping Techniques

Texture mapping is a technique for simulating the surface characteristics of an object. This, technique was originally proposed by Catmull in 1974 (Rogers and David, 1985) and is still an important technique for creating realistic images. Texture mapping varies the surface characteristics of an object through the use of mathematical functions, or two-dimensional or three-dimensional tables. This technique has been used to modulate many surface characteristics, including color, roughness, reflection, transparency and even the actual surface geometry (displacement mapping).

Texturing is commonly used in applying a two-dimensional image onto an object to produce a color pattern on the object. Examples of this include creating a label on a wine bottle and the woven fabric color pattern on a sofa.

This technique is also commonly used to simulate bumps, wrinkles, and imperfections on the surface of objects by modifying the normal to the surface of the object. This helps in creating realistic images by reducing the smooth “antiseptic” quality of computer-generated images. Scratches in a table, the winkles on the surface of an orange and the bumps in a stucco wall can all be simulated with two-dimensional texturing of surface normals (bump mapping).

To model the actual surface geometry created by bumps, dents, and scratches, displacement mapping can be used. Displacement mapping differs from bump

mapping in that the actual geometry of the surface is modulated (on displaced) instead of just the normal to the surface. Displacement mapping solves two problems that occur with bump mapping, smooth silhouettes and smooth intersections. In bump mapping, the silhouette of the object will not show the effects of' the bump mapping, since the normal vector is just perturbed. The actual geometry of the silhouette (and the entire object) remains unchanged. The intersection of two bump mapped objects will also not show the effects of the bump mapping for the same reason. However, with displacement mapping, the geometry of the object is actually changed, so these problems will not occur.

Displacement mapping and other types of texture mapping are incorporated as follows into the rendering process. Texture mapping normally occurs just before the illumination calculations are performed to determine the final color of the surface element. Texture mapping of surface color, surface normal vector, surface reflection, and other surface illumination parameters is performed just prior to the illumination calculation to determine the final color of this surface element. Texture mapping of surface transparency normally occurs just prior to the combination of this surface element with the surface element behind it. In most rendering systems, displacement mapping needs to be performed before the hidden surface algorithm and actual changes the geometric representation of the object. In most rendering algorithm, texturing is only applied to the visible points on the surface of the object (after the hidden surface calculations). To avoid perspective distortions in the texture, the object space location of the visible point on the surface of the object is usually used in the texture mapping calculation. Different mapping algorithms will be discussed in the following subsections.

2.2.1 Two-Dimensional Texturing

Texture mapping was originally a two-dimensional technique. In two-dimensional texture mapping, each visible point p on an object is mapped into a two-dimensional apace (normally a two-dimensional table). The two-dimensional texture space is then evaluated to determine the value associated with the particular location

two-space. This value is t hen used for the particular surface characteristic of the point P on the object. Two-dimensional texturing was originally used in modeling the surface color of objects. Blinn *et al*. (1976) extended texture mapping to modulate the specular and reflection characteristics of objects. Blinn (1978) further extended texturing to modulate the normal vector of surfaces as a way of simulating bumps and wrinkles on objects. Finally, Cook (1984) suggested using two-dimensional texturing to manipulate all aspects of the illumination of an object, including surface displacement and shadowing.

There are several techniques for mapping the point into this two-dimensional texture space. For a survey of different mapping techniques, see (Heckbert and Paul, 1986). These techniques can be classified into two different classes of mapping techniques. The first class maps each polygon or patch of the object into the entire two-dimensional space. Therefore, using this class of mapping technique repeats the texture on each polygon or patch of the object. Within this class of mappings, two different mapping techniques are commonly used. The first mapping technique is referred to as inverse bilinear interpolation (Heckbert and Paul, 1986). This technique first, uses the inverse perspective transformation to transform the image space location of the point on the surface of the object into object space. Then, the point in object space is mapped into the texture definition space through the use of inverse bilinear interpolation. The second technique uses simple bilinear interpolation in screen space, where the location of the point in screen space is used in the texture mapping, and not the location of point in object space, as in the previous technique. This technique, however, suffers from perspective distortion. A more detailed description of these techniques can he found in (Heckbert and Paul, 1986).

The second class of mapping techniques maps the entire surface of the object into the texture space. This class of mappings “wraps" the texture around the entire object instead of repeating the texture on each polygon or patch. The simplest technique of this class uses a spherical mapping. With spherical mapping, the polar coordinates of the pint, with respect to the object coordinate system, are used to map the surface of the object into the texture space. Simple linear interpolation in screen space can also be used to map the entire object surface into the texture space. In this

technique, the user assigns texture space coordinates with each vertex of a polygon (or control point of a patch). Then, during scan conversion, these texture space coordinates are simply linearly interpolated to determine the texture space coordinate for each visible point on the surface of the object.

The mapping of each pint on the surface of a three-dimensional object into the two-dimensional texture space creates many problems in the resultant appearance of the texture applied to the object. The two-dimensional texture may suffer distortions when applied to the three-dimensional object. For example, when a simple spherical projection is used to map a two-dimensional texture onto an object, the texture is normally compressed near the poles and stretched near the equator of the object. Another problem caused by the texture mapping is discontinuities at the seams of the texture applied to the object. A seam is where two separate sides of the texture map meet when the texture is applied to the object. Discontinuities can occur at the seams for two reasons. First, the values in the texture map at the two edges that meet may not be the same. Second, the scale can change abruptly at the seams because of the mapping technique. The surface area that each section of the texture map occupies on opposite sides of the seam may be different because of the mapping. Blurring or averaging of values at the seams can be used to solve this problem (Burt *et al*., 1983).

A few techniques have also been proposed to solve the distortion problem. The first technique is referred to as two-part texture mapping (Bier and Kenneth, 1986). In this technique, the texture is first projected onto an intermediate three-dimensional shape. Then the texture is mapped from the intermediate three-dimensional shape onto the final object. The choice of the separate parts of the mapping allows for choosing a combination of techniques that minimize distortion for the particular object.

Another solution can he termed object unfolding (Samek *et al*., 1986). In this technique, the polygons of the object are unfolded onto a flat two-dimensional plane. Then vertices in the object are associated with locations in the texture map. The user interactively assigns a texture space coordinate with each vertex in the unfolded object. The location of the vertices in the texture space, are then used during the

mapping to apply the texture to the object. Problems with this technique include the possibility of introducing discontinuities into the texture applied to the image and loss of texture when applied to the object (parts of the texture map may not be applied to the object). This technique is used to reduce distortion and scale changes when the texture map is applied to the object. It still does not solve the problem with seams.

2.2.2 Three-Dimensional Texturing - Solid Texturing

In 1985, Gardner (1985), Peachey and Darwyn (1985) and Perlin (1985) all independently suggested extending two-dimensional texturing to three-dimensional texturing or solid texturing. Solid texturing differs from two-dimensional texturing in that solid texturing maps each point on the surface of an object to a three-dimensional texture space as opposed to a two-dimensional texture space. The location of the point in this three-dimensional “solid space” is used in calculating the value for modulating the surface characteristic of' the point on the object.

Solid texturing is incorporated into the viewing algorithm in a manner similar to two-dimensional texture mapping. As in two-dimensional texturing, the three-dimensional screen space location of each visible point on the surface of the object is mapped back to object space. The main difference is that then this three-dimensional object space location is used by the solid texturing functions to determine the value of the corresponding screen location, as opposed to the two-dimensional location used in two-dimensional texturing.

Gardner (1985) uses three-dimensional texturing for simulating clouds. This technique is used as a model for clouds by using Fourier synthesis to control the transparency of hollow ellipsoids. Figure 2.1 shows complex clouds modeled by Gardner (1985) by making use of linked ellipsoids.

Figure 2.1: Complex clouds modeled by linked ellipsoids (Gardner, 1985)

Peachey and Darwyn (1985) and Perlin (1985) both proposed solid texturing for controlling the color patterns of objects. They both also used solid texturing as a model for simulating objects carved from solid materials. In solid texturing, the texture is determined from evaluating three-dimensional functions based on the location of each point on the surface of an object in the solid texture Space. Therefore, solid texturing is an extremely powerful technique for simulating objects carved from solid materials such as wood, marble, granite, and other stone materials.

Peachey and Darwyn (1985) use both three-dimensional functions and projected two-dimensional images for defining his three-dimensional textures. Peachey has suggested several functions, which can be used to create interesting three-dimensional textures. By randomly placing spheres of random size throughout a solid space, he simulates materials in which bubbles of one material are captured in the solidification of another material. To simulate wood, Peachey uses concentric cylinders of light and dark colors aligned along an arbitrary axis in three-space. Peachey simulates marble through placing starting locations for veins within the solid space. The direction of the veins is then controlled through the use of sinusoidal functions. The diameter of the cross-section of the veins is also controlled by more sinusoidal functions.

Perlin (1985) also chose three-dimensional functions to define his three-dimensional textures. Perlin makes extensive use of function composition to create interesting three-dimensional textures. The bases of most of these functions are two functions, one of which simulates random noise and the other which provides a “visual” simulation of turbulent flow. Through the use of these functions Perlin is

able to create very realistic images of marble, fire, water, and block glass. Details can be found in (Perlin, 1985).

Solid texturing solves some problems of two-dimensional texturing. Since solid texturing uses an affine mapping from the three-dimensional object space to the three-dimensional texture space (normally scale and translation transformations), there is no distortion of the texture when it is to the object. In two-dimensional texturing where a two-dimensional image is mapped onto a complex three-dimensional manifold, distortion and mapping of the texture applied to the object can occur. Solid texturing avoids these problems.

2.3 Simulating Complex Geometry

There have been many approaches to modeling the complex geometry of objects in our environment. Polygonal and patch models were the first models used. Constructive solid geometry (CSG) models were introduced later. More recent models can be categorized as either particle systems or procedural models.

2.3.1 Polygonal and Patch Modeling

The first geometric models of objects in computer graphics were polygonal meshes. A polygonal mesh is a collection of planner polygons containing vertex, edge, and connectivity information. Polygonal mesh models however, do not accurately model smooth surfaces, which are so commonly found in our environment. To circumvent this problem, Gouraud (Rogers and David, 1985) developed an intensity interpolation method for polygonal mesh objects to simulate the appearance of a curved surface. Phong (Rogers and David, 1985) developed an improved approximation method, which uses interpolation of the normal vector to approximate the illumination from curved surface. Both of those techniques suffer problems since they do not correctly model the geometry of a curved surface. For

example, the silhouette of the object is still polygonal and so is the line of' intersection of two polygonal objects.

Spline surface patches solve some of the problems with polygonal patch models, since they are an actual curved surface model useful for simulating smooth surfaces in our environment. Surface patch models have been extended to formulations that are easier to control and they have advantages over earlier formulations, such as hierarchical Spline models. Forsey and Bartels (1988) have created a hierarchical spline model that allows for uneven spacing of control points to provide an efficient formalization for objects that have areas of varying degrees of surface details.

Both polygonal and surface patch models are usually created by digitizing a real world model, or using an interactive system to create the model.

2.3.2 Constructive Solid Geometry Models

Polygonal and surface patch models are boundary representations of objects. The polygons and patches define the surface of the object. They separate points inside the object from points outside the object. The geometry of the solid interior of the object is not defined (the objects are hollow). Constructive solid geometry (CGS), on the other hand, is a solid model for objects. CSG objects are solid objects, not simply surfaces. The interior geometry of the object is part of the CSG representation of the object. CGS models use Boolean operations on simple shapes to define complex shapes. CSG models have several advantages over boundary representations. First, they are a very compact way of representing complex shapes and can be generated rapidly by solid modeling systems. Second, they contain the full three-dimensional volume geometry of the object as opposed to just the surface information. CSG models, however, do have some disadvantages when compared to boundary representations. First, they either require a volume rendering or ray tracing system to render the full three-dimensional volume of the object or they require the boundary representation to be calculated for use in a surface-based renderer.

Calculating the boundary representation from the CSG models requires extensive computations. Second, some shapes are not easily described as Boolean operations on simple shapes.

2.3.3 Particle Systems

A more recent approach to modeling complex geometric objects is particle systems. Reeves (1983) originated the use of particle systems. Particle systems have mainly been used for modeling natural phenomena, such as smoke, cloud, fire, trees, and water where the intricate detail of the phenomena is represented by a large collection of particles. Particle systems normally involve the use of a large number of small spherical particles. The animation of these particles is controlled through the use of procedures, which simulate the specific natural object, such as fire. The rendering of particle systems normally uses a simple constant shading model and often the color of the particle is determined by its three-dimensional location in space. For example, in Reeves' fire simulation (Reeves, 1983), the color of the particle is determined by its elevation from the base of the fire.

Particle systems have several advantages over patch and polygonal models. Particle systems normally use simple spheres for representing the geometry of the object or phenomenon, instead of complex patch or polygonal models. Therefore, the rendering system only needs to handle simple spherical models. The particle system rendering process also usually uses constant flat shading for the illumination of the particles, which is much simpler and quicker than normal illumination algorithms used for patch and polygonal models.

However, particle systems do have several disadvantages. They require an extremely large geometric database of particles to represent complex objects or phenomena. Shadowing algorithms and illumination algorithms for particle systems, although quicker, are not as realistic as the algorithms normally used for patch and polygonal models. Also, complex shapes can only be approximated by a large collection of spheres. Therefore, sharp edges are hard to simulate.

2.3.4 Procedural Modeling

Many different, modeling techniques can he termed procedural modeling. Fractal synthesis (Mandelbrot, 1982), Fourier synthesis (Gardener, 1985), hypertextures (Perlin and Hoffert, 1989), L-systems (Prusinkiewicz *et al*., 1988), volume density functions (Ebert and Parent, 1990), and inverse particle systems can all be considered to be procedural models. Procedural modeling uses algorithms to represent the geometry of objects. Most, physically based modeling techniques are also procedural modeling techniques, for example Kajiya's cloud modeling technique (Kajiya and Herzen, 1984).

Procedural modeling techniques have many advantages over polygonal and patch modeling techniques. The procedural model is evaluated during the rendering process to determine the geometry of the object / phenomena. Usually, the model is evaluated at the resolution of image rendering. Therefore, the models provide the needed amount of detail without introducing high frequency details, which will result in aliasing artifacts. Secondly, complex shapes can be represented with very little data storage space. For instance, a fractal mountain can be represented with a procedure of less than 50 lines of C language code; whereas, to achieve an equivalently detailed polygona1 model might require 50,000 polygons (Ebert, 1996). Because of this, procedural modeling techniques are often used to represent objects of very high degrees of detail such as natural phenomena.

Procedural models, however, do have disadvantages over particle systems, patch, and polygonal modeling. The main disadvantage is the computation time required to evaluate the procedure. Often the procedure is evaluated during rendering and many floating-point computations are usually performed during each evaluation of the procedural model. The computation time of rendering a procedural model is directly related to the computation time to evaluate the procedure for the model.

2.4 Modeling Natural Phenomena

Natural phenomena are some of the hardest things to model in computer graphics. Modeling phenomena such as mountains, trees, plants, fire, water, clouds, and smoke have inspired much research in computer graphics. Realistic models of these phenomena, however, still elude current computer graphics techniques. The complex intricate geometry and motion of these phenomena make them very difficult to model. Some natural phenomena, such as mountains, trees, and plants have intricate rigid shapes, while others; such as clouds, fire, and water have intricate amorphous shapes. Most models for natural phenomena can he classified as fractals, particle systems, or other types of procedural models.

2.4.1 Modeling Terrain and Vegetation

Most approaches to modeling terrain have used a fractal modeling approach (Miller, 1986). Mandelbort and Musgrave (Musgrave *et al*., 1989) have done much work on the modeling of terrain using fractals. Recent work has included modeling terrain with fractals and then simulating the natural erosion processes that affect terrain to create a more realistic model (Musgrave *et al*., 1989).

There has been a wider range of techniques used to model vegetation. Iterated functions systems (Demko *et al*., 1985), fractals, particle systems (Reeves and Blau, 1985) and L-systems (Prusinkiewicz *et al*., 1988) have all been used to model plants and trees. Very realistic images of plants have recently been produced through the use of L-systems

2.4.2 Modeling Fire and Water

The modeling of water has received as much attention as the modeling of terrain and vegetation. Early models for water used simple cycloidal models for

controlling the height of the surface of the water (Max, 1981). This model had many problems, including its inability to simulate breaking or curling waves since there can only be one height value for each x-y location. 1986 and 1987 saw a flurry of research in modeling ocean waves (T'so and Barsky, 1987; Fournier and Reeves, 1986; Peachey, 1986). Some of these models provided mom physically based models that simulated breaking waves and even spray from breaking waves created by particle systems (Fournier and Reeves, 1986). More recently, Kass and Miller (1990) have developed a model based on wave equations that even allows for the net transport of water volume. The main drawback with their approach is that they again use a height field, so they cannot simulate cresting or breaking waves.

The modeling of fire has received very little attention compared to the modeling of water. The main approach to modeling fire has been the use of particle systems (Reeves, 1983; Sims, 1990).

2.4.3 Modeling and Animating Gaseous Phenomena

The rendering of scenes containing clouds, fog, atmospheric dispersion effects, and other gaseous phenomena has received much attention in the computer graphics literature. Several papers deal mainly with atmospheric dispersion effects (Willis, 1987; Nishita *el al*., 1987; Rushmeier and Torrance, 1987), while many cover the illumination of these gaseous phenomena in detail (Blinn, 1982; Kajiya and Herzen, 1984; Max, 1986; Kass and Miller, 1990). Most authors have used a low albedo reflection model, while a few, B1inn (1982), Kajiya and Herzen (1984), and Rushmeier and Torrance (1987), discuss the implementation of a high albedo model.

A low albedo reflectance model assumes that secondary scattering effects are negligible, while a high albedo illumination model calculates the secondary and higher order scattering effects.

Another issue is modeling the geometry of these gases. Some authors use a constant density medium (Klassen, 1987; Nishita *et al*., 1987), but do allow different

layers of constant densities. This allows for a very limited geometry for the gases. Voss (1983) uses fractals and Max (1986) uses height fields for modeling the geometry of clouds. Kajiya and Herzen (1984) use a physically based model for clouds, which simulates water vapor, heat flow, wing, etc to form a physical model for clouds. However, the resulting images are not very realistic. Gardner (1985) uses hollow ellipsoids to model the geometry of clouds. He controls the transparency of hollow ellipsoids through the use of Fourier synthesis. To form larger cloud formations, he combines many different ellipsoids to form cloud groups. The main problem with his approach is that it is not a true three-dimensional model for the clouds, so accurate cloud shadowing is impossible. Another problem is that once you enter a cloud, you can clearly see that it is a simple hollow object and not a full three-dimensional cloud volume.

Ebert and Parent (1990) use turbulent flow based functions to model the density of a variety of gases. These functions are based on Perlin's visual simulation of turbulent flow (Perlin, 1985) and are similar to the idea of hypertextures (Perlin and Hoffert, 1989). This model is a true three-dimensional model for the geometry of gases and provides realistic results. This technique seems to provide more realistic results than most previous efforts by providing visually realistic renderings and animations of gaseous phenomena and the shadows they cast. These techniques are based on a visual simulation of turbulent flow, so it is a visual simulation of the turbulent processes that determines the geometry of gaseous phenomena. These techniques can also be extended to use a physically based turbulent flow model and can be very efficient when simplifying assumptions are made. This approach will be discussed in more detail later in this dissertation.

2.5 Modeling Fuzzy Objects

As is true for any object or phenomenon, there are multiple ways to model fuzzy objects such as clouds. An explicit representation of every water droplet in a cloud would require far too much computation and storage (Harris, 2003), so most researchers have used much coarser models. In this section I describe five general

methods that have been used to model and render clouds: particle systems, metaballs, voxel volumes, procedural noise, and textured solids. Note that these techniques are not mutually exclusive; multiple techniques have been combined with good results.

2.5.1 Particle Systems

Particle systems model objects as a collection of particles—simple primitives that can be represented by a single 3D position and a small number of attributes such as radius, color, and texture. Reeves (1983) introduced particle systems in as an approach to modeling clouds and other "fuzzy" phenomena, and described approximate methods of shading particle models in (Reeves and Blau, 1985). Particles can be created by hand using a modeling tool, procedurally generated, or created with some combination of the two. Particles can be rendered in a variety of ways. Harris and Lastra (2001) modeled static clouds with particles and rendered each particle as a small texture sprite (or "split" (Westover, 1990)). The details of this technique can be found in (Harris and Lastra, 2001). Figure 2.2 shows clouds modeled by Harria and Lastra (2001).

Figure 2.2: Shading with multiple forward scattering. (Harris and Lastra, 2001)

Particles have the advantage that they usually require only very simple and inexpensive code to maintain and render. Because a particle implicitly represents a spherical volume, a cloud built with particles usually requires much less storage than a similarly detailed cloud represented with other methods. This advantage may diminish as detail increases, because many tiny particles are needed to achieve high detail. In this situation other techniques may be more desirable.

2.5.2 Metaballs

Metaballs (or “blobs”) represent volumes as the superposition of potential fields of a set of sources, each of which is defined by a center, radius, and strength (Blinn, 1982a). These volumes can be rendered in a number of ways, including ray tracing and splatting. Alternatively, isosurfaces can be extracted and rendered, but this might not be appropriate for clouds. Nishita *et al*. (1999) used metaballs to model clouds by first creating a basic cloud shape by hand-placing a few metaballs, and then adding detail via a fractal method of generating new metaballs on the surfaces of existing ones (Nishita *et al*., 1996). Dobashi *et al*. (1999) used metaballs to model clouds extracted from satellite images. In Dobashi *et al*. (2000), clouds simulated on a voxel grid were converted into metaballs for rendering with splatting. The figure 2.3 shows a snapshot of clouds formed around mountains, as modeled by Dobashi *et al*. (2000).

Figure 2.3: Cloud formation around mountains (Dobashi *et al*., 2000)

2.5.3 Voxel Volumes

Voxels are another common representation for clouds. A voxel is the three-dimensional analog of a pixel. It is a single cell of a regular grid subdivision of a rectangular prism. Voxel models provide a uniform sampling of the volume, and can be rendered with both forward and backward methods. There is a large body of existing work on volume rendering that can be drawn upon when rendering clouds

represented as voxel volumes (Levoy, 1988; Westover, 1990; Wilson *et al*., 1994; Cabral *et al*., 1994; Kniss *et al*., 2002). Voxel grids are typically used when physically-based simulation is involved. Kajiya and Herzen (1984) performed a simple physical cloud simulation and stored the results in a voxel volume which they rendered using ray tracing. Figure 2.4 shows clouds rendered by Kajiya and Herzen (1984).

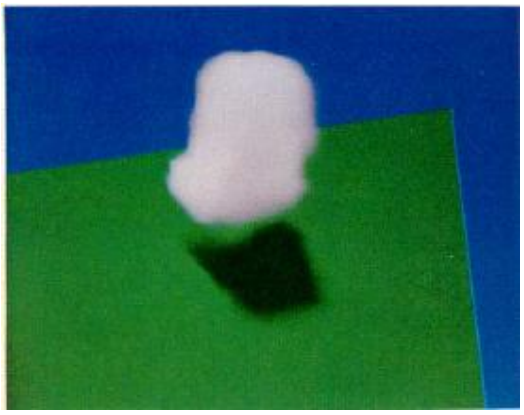

Figure 2.4: Cloud rendered by ray tracing (Kajiya and Herzen, 1984)

Dobashi, *et al*. (1998) simulated clouds on a voxel grid using a cellular automata model similar to Nagel and Raschke (1992), converted the grid to metaballs, and rendered them using splatting (Dobashi *et al*., 2000). Miyazaki *et al*. (2001) also performed cloud simulation on a grid using a method known as a Coupled Map Lattice (CML), and then rendered the resulting clouds in the same way as Dobashi *et al*. Overby *et al*. (2002) solved a set of partial differential equations to generate clouds on a voxel grid and rendered them using SkyWorks rendering engine (Harris and Lastra, 2001).

2.5.4 Procedural Noise

Procedural solid noise techniques are another important technique for generating models of clouds. These methods use noise as a basis, and perform various operations on the noise to generate random but continuous density data to fill

cloud volumes (Lewis, 1989; Perlin, 1985). Ebert has done much work in modeling "solid spaces" using procedural solid noise, including offline computation of realistic images of smoke, steam, and clouds (Ebert and Parent, 1990; Ebert, 1997; Ebert *et al*., 2002). Ebert modeled clouds using a union of implicit functions. He then perturbed the solid space defined by the implicit functions using procedural solid noise, and rendered it using a scan line renderer. Schpok *et al*. (2003) recently extended Ebert's techniques to take advantage of programmable graphics hardware for fast animation and rendering. Figure 2.5 shows volumetric implicit turbulent cloud modeled by Ebert (1997).

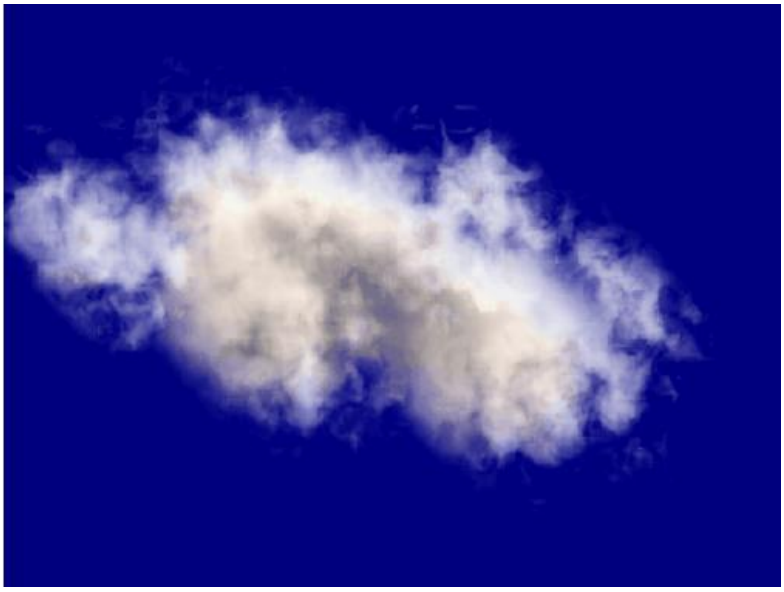

Figure 2.5: Volumetric Implicit Turbulent Cloud (Ebert, 1997)

2.5.5 Textured Solids

Others have chosen surface representations of clouds rather than volume representations. Gardner used fractal texturing on the surface of ellipsoids to simulate the appearance of clouds (Gardner, 1985). By combining multiple textured and shaded ellipsoids, he was able to create convincing cloudy scenes. Lewis also used ellipsoids for clouds, but with procedural solid noise (Lewis, 1989). More recently, Elinas and Sturzlinger used a variation of Gardner's method to interactively

render clouds composed of multiple ellipsoids (Elinas and Sturzlinger, 2001). Figure 2.6 shows clouds modeled by Elinas and Sturzlinger (2001) using 30 ellipsoids.

Figure 2.6: Cloud constructed from 30 ellipsoids (Elinas and Sturzlinger, 2001)

2.6 Cloud Dynamics Simulation

Cloud simulation has been of interest to meteorologists and atmospheric scientists since the advancement in high performance computing, but it has only recently drawn much interest from the computer graphics community. Scientific simulations of clouds and weather are typically very complex, requiring many hours of computation to simulate a relatively short time of cloud development.

The earliest simulations in atmospheric science were simple one-dimensional models. These models represented only vertical motion and computed changes under the influences of condensation and precipitation. Later models extended the simulation to two dimensions, but the extreme computational expense of three dimensions was prohibitive, so researchers tended to resort to slab symmetry or axial symmetry. These symmetries limit simulation to two-dimensions, but they at least provide the ability to simulate horizontal wind shear, which is important to cloud dynamics. One of the earliest such simulations was presented in (Takeda, 1971). Because rotational flow—including vortices with both horizontal and vertical axes of rotation—is common in real clouds, three-dimensional simulation is essential for high accuracy. Steiner presented the first fully three-dimensional model, and in a comparison with a similar two-dimensional model, he showed important differences

in the rotational motion of the clouds (Steiner, 1973). Three-dimensional cloud simulation has progressed since then.

Simulations from atmospheric physics are too expensive for computer graphics applications other than scientific visualization. Because they are used to understand our atmosphere and weather, many of them include a high level of detail that is not visible in nature, including very specific tracking of water state and droplet size distributions, complex microphysics, and detailed fluid dynamics at a variety of scales. If the goal is simply to create realistic images and animations of clouds, much less detailed visual simulations can be used.

Kajiya and Herzen were the first in computer graphics to demonstrate a visual cloud simulation (Kajiya and Herzen, 1984). They solved a very simple set of partial differential equations to generate cloud data sets for their ray tracing algorithm. The Partial differential Equations (PDE) they solved were the Navier-Stokes equations of incompressible fluid flow; a simple thermodynamic equation to account for advection of temperature and latent heat effects; and a simple water continuity equation. The simulation required about 10 seconds per time step (one second of cloud evolution) to update a 10×10×20 grid on a VAX 11/780. Overby *et al*. described a similar but slightly more detailed physical model based on PDEs (Overby *et al*., 2002). They used the stable fluid simulation algorithm of (Stam, 1999) to solve the Navier-Stokes equations. The stability of this method allows much larger time steps, so Overby *et al*. were able to achieve simulation rates of one iteration per second on a 15 × 50 × 15 grid using an 800MHz Pentium III. Harris has implemented a faster and slightly more realistic cloud simulation using programmable floating point graphics hardware (Harris *et al*., 2003; Harris, 2003).

Other researchers have tried simpler, but less realistic rule-based simulation techniques. Neyret used an animated particle system to model cloud behavior, using a set of heuristics to approximate the rolling behavior of convective clouds (Neyret, 1997). (Dobashi *et al*., 2000) used a simple cellular automata (CA) model of cloud formation to animate clouds offline. The model was based on the simple CA introduced by (Nagel and Raschke, 1992). Nagel and Raschke's original CA had rules for the spread of humidity between neighboring cells and for the formation of

clouds in humid cells, but included no mechanism for evaporation. Dobashi *et al*. added a stochastic rule for evaporation so that the clouds would appear to grow and dissipate. Their model achieved a simulation time of about 0.5 seconds on a 256 × 256 × 20 volume using a dual 500 MHz Pentium III.

In similar work, Miyazaki *et al*. used a coupled map lattice rather than a cellular automaton (Miyazaki *et al*., 2001). This model was an extension of an earlier coupled map lattice model from the physics literature. Coupled map lattices (CML) are an extension of CA with continuous state values at the cells, rather than discrete values. Harris *et al*. have done work on performing CML simulations on programmable graphics hardware (Harris *et al*., 2002). The CML of Miyazaki *et al*. used rules based on atmospheric fluid dynamics, including a rule used to approximate incompressibility and rules for advection, vapor and temperature diffusion, buoyancy, and phase changes. They were able to simulate a 3–5 s time step on a 256 × 256 × 40 lattice in about 10 s on a 1 GHz Pentium III.

2.7 Light Scattering and Cloud Radiometry

Some of the earliest work on simulating light scattering for computer graphics was presented in (Blinn, 1982b). Motivated by the need to render the rings of Saturn, Blinn described an approximate method for computing the appearance of cloudy or dusty surfaces via statistical simulation of the light-matter interaction. Blinn (1982b) made a simplifying assumption in his model—that the primary effect of light scattering is due to reflection from a single particle in the medium, and multiple reflections can be considered negligible. This single scattering assumption has become common in computer graphics, but as Blinn (1982b) and others have noted, it is only valid for media with particles of low single scattering albedo. Blinn (1982b) also simplified the problem by limiting application of his model to plane parallel atmospheres, rather than handling scattering in arbitrary domains.

As described by Harris (2003), accurate computation of light scattering in media with high single scattering albedo is expensive, because it requires evaluation

of a double integral equation. In practice, researchers either use simplifying assumptions to reduce the complexity of the problem, or perform long offline computations. There are multiple ways to compute light scattering, and many simplifications that can be applied. The previous work in this area can be grouped into five categories: Spherical Harmonics Methods, Finite Element Methods, Discrete Ordinates, Monte Carlo Integration, and Line Integral Methods.

2.7.1 Spherical Harmonics Methods

The spherical harmonics $Y_l^m(\theta, \phi)$ are the angular portion of the solution of Laplace's equation in spherical coordinates. The spherical harmonics form a complete orthonormal basis. This means that an arbitrary function $f(\theta, \phi)$ can be represented by an infinite series expansion in terms of spherical harmonics:

$$f(\theta,\phi) = \sum_{l=0}^{\infty} \sum_{m=0}^{l} A_l^m Y_l^m(\theta,\phi) \tag{2.1}$$

The method of determining the coefficients, A_l^m, of the series is analogous to determining the coefficients of a Fourier series expansion of a function. If the value of f is known at a number of samples, then a series of linear equations can be formulated and solved for the coefficients. Spherical harmonics methods have been used by (Bhate and Tokuta, 1992; Kajiya and Herzen, 1984; Stam, 1995) to compute multiple scattering.

Kajiya and Herzen presented a ray tracing technique for rendering arbitrary volumes of scattering media. In addition to a simple single scattering model, they also described a solution method for multiple scattering that uses spherical harmonics (Kajiya and Herzen, 1984). Their single scattering simulation method stored the cloud density and illumination data on voxel grids, and their algorithm required two passes. In the first pass, scattering and absorption were integrated along paths from the light source through the cloud to each voxel where the resulting intensities were stored. In the second pass, eye rays were traced through the volume of intensities and

scattering of light to the eye was computed, resulting in a cloud image. For multiple scattering, the authors derived a discrete spherical harmonics approximation to the multiple scattering equations, and solved the resulting matrix of partial differential equations using relaxation. This matrix solution replaces the first integration pass of the single scattering algorithm. As mentioned in (Stam, 1995), this method is known as the PN-method in the transport theory literature, where N is the degree of the highest harmonic in the spherical harmonic expansion.

Following Kajiya and Herzen's lead, two pass algorithms for computing light scattering in volumetric media are now common. Interestingly, (Max, 1994) points out that while Kajiya and Herzen attempted to compute multiple scattering for the case of an isotropic phase function, it is not clear if they succeeded, all of the images in the paper seem to have been computed with the simpler single scattering model.

Stam explained in (Stam, 1995) that while (Kajiya and Herzen, 1984) derived a very general N-term expression for multiple scattering using a spherical harmonics expansion, they truncated the expansion after the first term to produce their results. He showed that this truncation results in a diffusion type equation for the scattered portion of the illumination field. In media where scattering events are very frequent— "optically thick" media—multiple scattering can be approximated as diffusion of the light energy. In other words, at any location in the medium, photons can be found traveling in arbitrary directions. The light is said to be diffuse. Stam presented this diffusion approximation in more detail. Like Kajiya and Herzen, Stam represented the scattering medium on a voxel grid. He described two ways to solve for the intensity. In the first method, he discretized the diffusion approximation on the grid to formulate a system of linear equations that he then solved using the multigrid method. The second method is a finite element method in which he used a series expansion of basis functions, specifically Gaussian kernel functions of distance. This expansion led to a matrix system that he solved using LU decomposition (Harris, 2003).

2.7.2 Finite Element Methods

The finite element method is another technique for solving integral equations that has been applied to light transport. In the finite element method, an unknown function is approximated by dividing the domain of the function into a number of small pieces, or elements, over which the function can be approximated using simple basis functions (often polynomials). As a result, the unknown function can be represented with a finite number of unknowns and solved numerically.

A common application of finite elements in computer graphics is the radiosity method for computing diffuse reflection among surfaces. In the radiosity method, the surfaces of a scene represent the domain of the radiosity function. An integral equation characterizes the intensity, or radiosity, of light reflected from the surfaces. To solve the radiosity equation, the surfaces are first subdivided into a number of small elements on which the radiosity will be represented by a sum of weighted basis functions. This formulation results in a system of linear equations that can be solved for the weights. The coefficients of this system are integrals over parts of the surfaces. Intuitively, light incident on an arbitrary point in the scene can be reflected to any other point; hence the coefficients are integrals over the scene. In the finite element case, these integrals are evaluated for every pair of elements in the scene, and are called form factors.

Rushmeier and Torrance extended the radiosity method to include radiative transfer in volumes of participating media (Rushmeier and Torrance, 1987). This zonal method, like the radiosity method, was originally developed for radiant heat transfer analysis. The zonal method divides the volume of a participating medium into finite elements which are assumed to have constant radiosity. As with the radiosity method, form factors are computed for every pair combination of surface elements in the scene, as well as every pair of volume elements and all surface-volume pairs. This is complicated by the fact that the form factors involve a double integral over points in both elements, as well as along the path between the elements. As in the radiosity method, a system of simultaneous linear radiosity equations is formulated based on these form factors. The solution of this system is the steady-state diffuse radiosity at each element of the environment, including the effects of

scattering and absorption by the participating medium. Rushmeier and Torrance's presentation of the zonal method was limited to isotropic scattering media, with no mention of phase functions.

Nishita *et al*. introduced approximations and a rendering technique for global illumination of clouds, accounting for multiple anisotropic scattering and skylight (Nishita *et al*., 1996). This method can also be considered a finite element method, because the volume is divided into voxels and radiative transfer between voxels is computed. Nishita *et al*. made two simplifying observations that reduced the cost of the computation. The first observation was that the phase function of cloud water droplets is highly anisotropic, favoring forward scattering. The result of this is that not all directions contribute strongly to the illumination of a given volume element. Therefore, Nishita *et al*. computed a "reference pattern" of voxels that contributed significantly to a given point. This pattern is constant at every position in the volume, because the sun can be considered to be infinitely distant. Thus, the same sampling pattern can be used to update the illumination of each voxel. The second observation they made was that only the first few orders of scattering contribute strongly to the illumination of a given voxel. Therefore, Nishita *et al*. only computed up to the third order of scattering.

2.7.3 Discrete Ordinates

The method of discrete ordinates allocates the radiosity exiting each volume element into a collection of M discrete directions. The intensity is assumed to be constant over each direction "bin". This method can be used to account for anisotropic scattering. If an interaction between a pair of elements can be represented by only one direction bin (this is unreasonable for elements that are close), then the number of non-zero elements in the matrix of linear coefficients is MN^2, where $N = n^3$ is the number of elements in the volume (Max, 1994). However, Max points out that this method introduces sampling artifacts because it effectively shoots energy from elements in infinitesimal beams along the discrete directions, missing the regions between them. In work inspired by (Patmore, 1993), Max (1994) improved

on the basic method of discrete ordinates by efficiently spreading the shot radiosity over an entire direction bin, rather than along discrete directions. The method achieves a large speedup by handling a whole plane of source elements simultaneously, which reduces the computation time to $O(MN \log N + M^2N)$ (Harris, 2003).

2.7.4 Monte Carlo Integration

Monte Carlo Integration is a statistical method that uses sequences of random numbers to solve integral equations (Harris, 2003). In complex problems like light transport, where computing all possible light-matter interactions would be impossible, Monte Carlo methods reduce the complexity by randomly sampling the integration domain. With enough samples, chosen intelligently based on importance, an accurate solution can be found with much less computation than a complete model would require. The technique of intelligently choosing samples is called importance sampling, and the specific method depends on the problem being solved. A common application of Monte Carlo methods in computer graphics is Monte Carlo ray tracing. In this technique, whenever a light ray traversing a scene interacts with matter (either a solid surface or a participating medium), statistical methods are used to determine whether the light is absorbed or scattered (for solids, this scattering may be thought of as reflection or refraction). If the light is scattered, the scattered ray direction is also chosen using stochastic methods. Importance sampling is typically used to determine the direction via the evaluation of a probability function.

Blasi, *et al*. (1993) presented a technique for rendering arbitrary volumes of participating media using Monte Carlo ray tracing. They placed no restrictions on the medium, allowing arbitrary distributions of density and phase function, and accounting for multiple scattering. They demonstrated an importance sampling technique that uses the phase function as a probability function to determine the outgoing direction of scattered rays. This way, the in-scattering integral does not have to be evaluated over the entire sphere of incoming directions, and a large amount of computation is saved. Using the phase function for importance sampling

ensures that the most significant contributions of scattering are used to determine the intensity. In this way, the technique is similar to the “reference pattern” technique used by (Nishita *et al*., 1996).

Photon mapping is a variation of pure Monte Carlo ray tracing in which photons (particles of radiant energy) are traced through a scene (Jensen, 1996). Many photons are traced through the scene, starting at the light sources. Whenever a photon lands on a nonspecular surface it is stored in a photon map, a data structure that stores the position, incoming direction, and radiance of each photon hit. The radiance on a surface can be estimated at any point from the photons closest to that point. Photon mapping requires two passes; the first pass builds the photon map, and the second generates an image from the photon map. Image generation is typically performed using ray tracing from the eye. The photon map exhibits the flexibility of Monte Carlo ray tracing methods, but avoids the grainy noise that often plagues them. Jensen and Christensen extended the basic photon map to incorporate the effects of participating media (Jensen and Christensen, 1998). To do so, they introduced a volume photon map to store photons within participating media, and derived a formula for estimating the radiance in the media using this map. Their techniques enable simulation of multiple scattering, volume caustics (focusing of light onto participating media caused by specular reflection or refraction), and color transfer between surfaces and volumes of participating media.

2.7.5 Line Integral Methods

Recently, interest in simulating light scattering has grown among developers of interactive applications. For view-dependent effects and dynamic phenomena, the techniques described in the previous sections are not practical. While those techniques accurately portray the effects of multiple scattering, they require a large amount of computation. For interactive applications, simplifications must be made.

A first step in simplifying the computation is to ignore volumetric scattering altogether. With or without scattering, visualization of the shadowing effects of

absorption by the medium is desirable. This requires at least one pass through the volume (along the direction of light propagation) to integrate the intensity of transmitted light. Because methods that make this simplification perform the intensity integration along lines from the light source through the volume may be called line integral methods. Kajiya and Herzen's original single scattering algorithm is a line integral method. Intuitively, line integral methods are limited to single scattering because they cannot propagate light back to points already traversed.

Dobashi *et al*. (2000) described a simple line integral technique for computing the illumination of clouds using the standard blending operations provided by computer graphics Application Programming Interface (API) such as OpenGL (Segal and Akeley, 2001). Dobashi *et al*. (2000) represented clouds as collections of large "particles" represented by textured billboards. To compute illumination, they rendered the particles in order of increasing distance from the sun into an initially white frame buffer. They configured OpenGL blending operations so that each pixel covered by a particle was darkened by an amount proportional to attenuation by the particle. After rendering a particle, they read the color of the pixel at the center of projection of the particle from the frame buffer. They stored this value as the intensity of incident light that reached the particle through the cloud. Traversal of the particles in order of increasing distance from the light source evaluates the line integral of extinction through each pixel. Because pixels are darkened by every particle that overlaps them, this method computes accurate self-shadowing of the cloud. After this first pass, they rendered particles from back to front with respect to the view point, using the intensities computed in the first pass. They configured blending to integrate absorption and single scattering along lines through each pixel of the image, resulting in a realistic image of the clouds. Dobashi *et al*. (2000) further enhanced this realism by computing the shadowing of the terrain by the clouds and shafts of light between the clouds.

Kniss *et al*. (2002) presented a similar line integral approach for absorption and multiple forward scattering in the context of direct volume rendering. They rendered volumes of translucent media from 3D textures by rendering slices of the volume oriented to face along the halfway vector between the light and view directions. This "half angle slice" technique allowed them to interleave light

transport integration with the display of the volume. The method traverses the volume slices in order of increasing distance from the light source, performing alternate display and illumination passes. Three buffers are maintained: two for the computation of the illumination of the volume (current and next), and one (typically the frame buffer) for display of the volume. During the display pass, the current slice is rendered from the observer's point of view. The slice is textured with the 3D volume texture blended with the current illumination buffer. This results in self-shadowing of the volume as in Dobashi *et al*. (2000), as well as incorporating the scattering computed during the illumination pass as in Harris and Lastra (2001). During the illumination pass, the slice is rendered into the next illumination buffer from the light's point of view, and blended with the current illumination buffer to compute the next step in the line integral of extinction and forward in-scattering. During this blending, the current buffer is sampled multiple times at jittered locations, and the samples are averaged. This accomplishes a blurring of the forward-scattered light, an ad hoc approximation of multiple scattering over a small solid angle around the forward direction. Even though this method is ad hoc, it is physically-based because multiple scattering in media with a high single scattering albedo results in "blurring" of the light intensity (the light is diffused).

2.8 Virtual Environment

Since fiction writers have already been exploring the role of computer in a future world, and have described a synthetic 3D universe that is as believable as the real physical universe. Such Virtual Reality (VR) systems create a 'cyberspace' where it is possible to interact with anything and anyone on a virtual level (Vince, 1995).

The key technologies behind such imaginative writing are real-time computer graphics, color displays and simulation software. Computer graphics provides the basis for creating the synthetic images, while a Head-Mounted Display (MHD) supplies the user's eyes with a stereoscopic view of a computer generated world.

Complex software creates the Virtual Environment (VE), which could be anything from 3D objects to abstract databases.

Virtual reality systems have two important areas: the first concerns with user *immersion*, and the second relates to the degree of *interaction* the user has with the virtual environment.

2.8.1 Immersion

The sensation of being immersed within a virtual environment is greatly influenced by the user's integration with the synthetic images. For example, in the case of flight simulator, the pilot and co-pilot sit inside a replica cockpit and gaze through the window into a 200° panoramic mirror reflecting the computer-generated graphics. This creates a realistic sensation of being in a real plane flying over some 3D landscape. Some virtual reality systems, on the other hand, provide each user with personal view of the virtual environment using an HMD which visually isolates them from the real world and provides the left and right eyes with two separate images that include parallax differences, which, given the right conditions, produce a realistic stereoscopic sensation. The user can acquire a positive sense of being immersed in the virtual environment, which is further enhanced when touch and sound are introduced. The immersion is further enhanced by allowing the user's head movements to control the gaze of direction of the synthetic images.

Virtual reality systems can be divided into three groups: immersive, non-immersive and hybrid. *Immersive* systems, replace the view of real world with computer generated images that react to the position and orientation of the user's head. A *non-immersive* system, on the other hand, leaves the user visually aware of the real world but able to observe the virtual world through some display device such as graphics workstation. The user navigates the virtual world using a device such as a space mouse. A *hybrid* virtual reality system permits the user to view the real world with virtual images superimposed over this view – such systems are also known as 'augmented reality' systems.

2.8.2 Interaction

When visually immersed with a virtual environment, there is a natural inquisitive temptation to reach out and touch virtual objects. Obviously this is impossible, as there is nothing to touch. The user's sense of immersion can be greatly enhanced by including part of a 'virtual body' such as hand in the virtual environment. The user now sees in the HMD a 3D virtual hand as part of the stereoscopic scene. If the user also wears an interactive glove, or a similar device, any movements their hand makes can be tracked and used to control the status of the virtual hand. The user has now been coupled to the VE in a way that allows some high level interaction to occur.

2.9 Summary

Brief review of modeling techniques for various natural phenomena in computer graphics has been discussed. The various methods used by researchers to model clouds have been presented. The methods for cloud radiometry have also been described. The Figure 2.7 a hierarchical diagram of the modeling techniques used for cloud modeling as described by Muhammad Azam Rana *et al*. (2003).

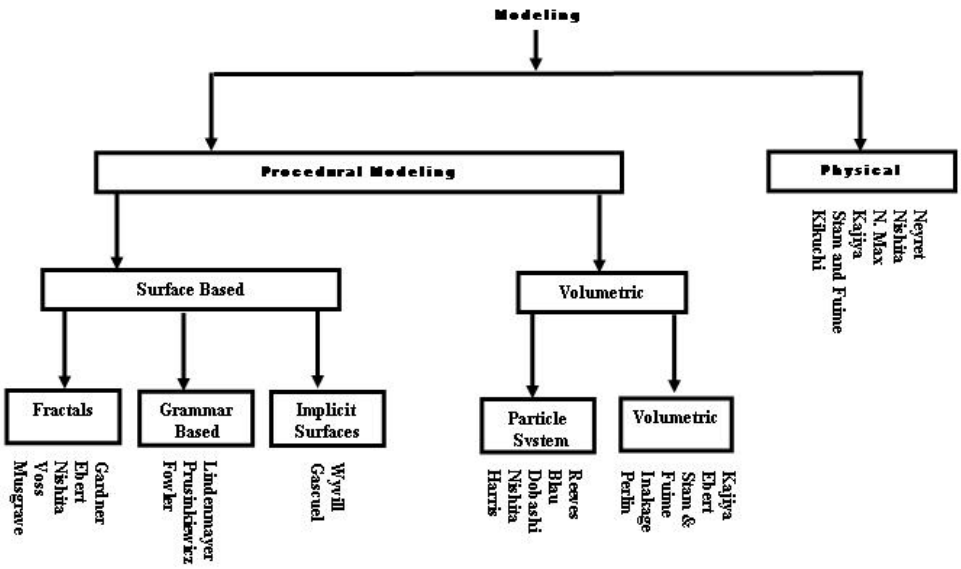

Figure 2.7: Hierarchy for cloud modeling techniques
(Muhammad Azam Rana et al., 2003)

Kajiya and Herzen (1984) modeled cloud by physical model. He made use of numerical simulation of fluid dynamics. Solution of complicated non-linear equations is difficult and resource intensive, requiring special knowledge to set proper boundary conditions. Neyret (1997) tried physical method for modeling clouds by making use of qualitative simulation. The method focuses on simulating growth of a single cluster of cloud and is not suitable for animation. Heinzlreiter *et al*. (2002) uses alpha-blended billboard textures to enhance the rendering process. The use hardware API makes real time rendering possible. The observer position cannot be changed. Cloud model needs improvements. Gardner (1985) uses procedural approach. He uses textured ellipsoids to simulate clouds. This model lacks true 3D geometric model, scattering effects cannot be simulated to calculate color of clouds. Elinas and Sturzlinger (2001) uses fractal/3D textured ellipsoid to model clouds. His approach is similar to Gardner (1985). This model produces good results for clouds at medium distance but may not produce good results for clouds at shorter distance. Ebert (1997) uses metaballs with noise function to model clouds. Metaballs and noise function are used to create animation of cloud formation. Shape of cloud is determined in advance and then visible parts are increased gradually. To include time as parameter, no way has been described. Stam and Fuime (1995) use Stochastic rendering of density fields to model clouds. This method can create realistic clouds, but it is impractical to create large-scale clouds viewed from space.

Trembilski (2002) simulates clouds by isosurfaces generated by Marching Cube algorithm. This method is developed for weather forecast. Lighting model is not true physical; graphics hardware is used for color calculations. Harris (2002) uses procedural approach by making use of particle system and dynamically generated imposters. Hardware accelerated API has made possible to achieve very high frame rates (greater than 50 fps).

CHAPTER III

PARTICLE SYSTEMS

3.1 Introduction

Particle systems differ from the other techniques in that their abstraction is in control of the animation and specification of the object. Particle systems do use a large database of geometric primitives to represent natural objects ("fuzzy objects"), but the animation, location, birth, and death of the particles representing the object are controlled algorithmically. Particle systems are most commonly used to represent natural phenomena such as fire, water, clouds, snow, rain, grass, and trees (Reeves, 1983). A particle- system object is represented by a large collection (cloud) of very simple geometric particles that change stochastically over time. The procedural aspect and main power of particle systems allow the specification and control of this extremely large cloud of geometric particles with very few parameters. Besides the geometric particles, a particle system has controllable stochastic particle animation procedures that govern the creation, movement, and death of the particles. These animation procedures often include physically based forces to simulate effects such as gravity, vorticity, conservation of momentum, and energy. Particle systems pose special rendering problems because of the large number of primitives, but specialized rendering techniques, including probabilistic rendering algorithms, have been developed to render particle systems (Reeves and Blau, 1985).

A particle system is a collection of many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system,

move and change from within the system, and die from the system. To compute each frame in a motion sequence, new particles are generated into the system. Each new particle is assigned its individual attributes. Any particles that have existed within the system past their prescribed lifetime are extinguished. The remaining particles are moved and transformed according to their dynamic attributes. And finally an image of the living particles is rendered in a frame buffer

The particle system can be programmed to execute any set of instructions at each step. Because it is procedural, this approach can incorporate any computational model that describes the appearance or dynamics of the object. For example, the motions and transformations of particles could be tied to the solution of a system of partial differential equations, or particle attributes could be assigned on the basis of statistical mechanics. We can, therefore, take advantage of models, which have been developed in other scientific or engineering disciplines.

Simple stochastic processes can be used as the procedural element of each step in the generation of a frame. To control the shape, appearance, and dynamics of the particles within a particle system, the model designer has access to a set of parameters. Stochastic processes that randomly select each particle's appearance and movement are constrained by these parameters. In general, each parameter specifies a range in which a particle's value must lie. Normally, a range is specified by providing its mean value and maximum variance.

In modeling fuzzy objects, the particle system approach has several important advantages over classical surface-oriented techniques.

- First, a particle (a point in three-dimensional space) is a much simpler primitive than a polygon, the simplest of the surface representations. Therefore, in the same amount of computation time one can process more of the basic primitives and produce a more complex image.

- Second, the model definition is procedural and is controlled by random numbers. Therefore, obtaining a highly detailed model does not necessarily require a great deal of human design time as is often the case with existing surface-based

systems. Because it is procedural, a particle system can adjust its level of detail to suit a specific set of viewing parameters. As with fractal surfaces Fournier *et al*. (1982), zooming in on a particle system can reveal more and more detail.

- Third, particle systems model objects that are "alive," that is, they change form over a period of time. It is difficult to represent complex dynamics of this form with surface-based modeling techniques.

3.2 Usage of Particles System

Dobashi *et al*. (2000) presented a method for simulation of clouds using Cellular Automation. 3D grids represent the simulation space and three state variables are assigned at each grid point. The state of each variable is either 0 or 1. Their status at time t+1 is calculated by the status at time t using transition rules.

The method of Dobashi *et al*. (2000) consists of two processes, simulation and rendering. The simulation space is divided into voxels. The voxels correspond to cells used in the cellular automaton. At each cell, three logical variables, vapor/humidity (hum), clouds (cld), and phase transition (or activation) factors (act) are assigned. The state of each variable is either 0 or 1. Cloud evolution is simulated by applying simple transition rules at each time step. The transition rules represent formation, extinction, and advection by winds. Since the state is either 0 or 1, the rules can be expressed by Boolean operations.

Images are generated in the rendering process by making use of the simulation results. What we can obtained from the simulation is no more than there are clouds (cld = 1) or, there are not-clouds (cld = 0) at each voxel. Therefore, a density at each point is calculated by smoothing the binary distribution. The clouds are then rendered using volume rendering techniques. The rendering process consists of two steps. The first step calculates the intensity of light reaching the center of each voxel. Cloud shadows are also calculated in this step. The shadows are obtained as a texture. Then, in the second step, images are generated. Clouds are rendered by using

a splatting method. To render shafts of light, consider multi spherical shells with their center at the viewpoint are considered.. The shells are then drawn from back to front using the hardware alpha-blending function. Shafts of light are rendered by mapping the shadow texture on the shells.

For growth simulation, the simulation space is aligned parallel to xyz axes and the number of cells is assumed to be $n_x \times n_y \times n_z$. As mentioned before, three logical variables, hum, act, and cld are assigned at each cell. The state of each variable is either 0 or 1. hum=1 means there is enough vapor to form clouds, act=1 means the phase transition from vapor to water (clouds) is ready to occur, and cld=1 means there are clouds. In the following, B I A and B Y A indicate conjunction and disjunctinon between A and B, respectively, and ¬A indicates negation of A. Their transition rules are given as follows.

$$hum(i,j,k,t_{i+1}) = hum(i,j,k,t_i) \text{ I } \neg act(i,j,k,t_i) \tag{3.1}$$

$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \text{ Y } act(i,j,k,t_i) \tag{3.2}$$

$$act(i,j,k,t_{i+1}) = \neg act(i,j,k,t_i) \text{ I } hum(i,j,k,t_i) \text{ I } f_{act}(i,j,k) \tag{3.3}$$

where $f_{act}(i,j,k)$ is a Boolean function and its value is calculated by the status of act around the cell. The following function is used by taking into account the fact that clouds grow upward and horizontally.

$$\begin{aligned} f_{act}(i,j,k) = {} & act(i+1,j,k,t_i) \text{ Y } act(i,j+1,k,t_i) \\ & \text{Y } act(i,j,k+1,t_i) \text{ Y } act(i-1,j,k,t_i) \\ & \text{Y } act(i,j-1,k,t_i) \text{ Y } act(i,j,k-1,t_i) \\ & \text{Y } act(i-2,j,k,t_i) \text{ Y } act(i+2,j,k,t_i) \\ & \text{Y } act(i,j-2,k,t_i) \text{ Y } act(i,j+2,k,t_i) \\ & \text{Y } act(i,j,k-2,t_i) \end{aligned} \tag{3.4}$$

For cloud extinction, firstly the animator specifies cloud extinction probability, p_{ext}. Next, at each cell whose cld is 1, a random number, rnd ($0 \le rnd \le 1$), is generated and cld is changed to 0 if $rnd < p_{ext}$. By changing the probability at

each cell at different times, the animator can specify regions where cloud extinction occurs frequently. Although this realizes the cloud extinction, there remains another problem. Clouds are never generated after the extinction at the cell. To solve this, vapor (hum) and phase transition factors (act) are supplied at specified time intervals. Similar to extinction, vapor probability, p_{hum}, and phase transition probability, pact, are used to set them randomly. That is, hum is changed to 1 if rnd < p_{hum} and act is changed to 1 if rnd < p_{act}. Cloud motion can be controlled by controlling the probabilities, p_{hum}, p_{act}, and p_{ext} at each cell at each time step. The methods described in this section are summarized by the following three transition rules.

$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \text{ I } IS(rnd > p_{ext}(i,j,k,t_i)) \quad (3.5)$$

$$hum(i,j,k,t_{i+1}) = hum(i,j,k,t_i) \text{ Y } IS(rnd < p_{hum}(i,j,k,t_i)) \quad (3.6)$$

$$act(i,j,k,t_{i+1}) = act(i,j,k,t_i) \text{ Y } IS(rnd < p_{act}(i,j,k,t_i)) \quad (3.7)$$

where rnd is a uniform random number, IS(e) is a Boolean function that returns 1 if the expression e is true, otherwise returns 0.

Beginning from initial random status, cloud growth is simulated by updating the state of each variable. First, *hum* is initialized by using uniform random numbers of probability p_{hum}. That is, *hum* is set to 1 if a random number between 0 and 1 is less than p_{hum}, otherwise *hum* is set to 0. Similarly, *act* is set to either 0 or 1 by using the probability p_{act}.

For continuous density distribution, metaballs are used. Metaballs are spheres in which a field function is defined. A metaball has two parameters, that is, density at the center and effective radius. Metaballs are placed at each grid point and the continuous distribution is represented as a weighted sum of the field functions. Continuous distribution is obtained by adjusting densities at their centers and their effective radii, based on the binary distribution.

Clouds can be observed moving in one direction, blown by winds. New transition rules are introduced to include the wind effect. The idea is simply to shift

all the variables toward the wind direction. It is assumed, for simplicity, the wind blows toward the direction of x-axis. Other cases can be handled by rotating the simulation space according to the wind direction. Furthermore, it is well known that the wind velocity is different depending on the height from the ground. The wind velocity, $v(z_k)$, is therefore specified as a function of z-coordinate of each cell (i,j,k). To implement the wind effect in the context of the cellular automaton, the function $v(z_k)$, is assumed to return integer value. The transition rules as follows.

$$hum(i,j,k,t_{i+1}) = \begin{cases} hum(i-v(z_k),j,k,t_i), & i-v(z_k) > 0 \\ 0, & otherwise \end{cases} \quad (3.8)$$

$$cld(i,j,k,t_{i+1}) = \begin{cases} cld(i-v(z_k),j,k,t_i), & i-v(z_k) > 0 \\ 0, & otherwise \end{cases} \quad (3.9)$$

$$act(i,j,k,t_{i+1}) = \begin{cases} act(i-v(z_k),j,k,t_i), & i-v(z_k) > 0 \\ 0, & otherwise \end{cases} \quad (3.10)$$

Rendering of clouds is based on the splatting algorithm using billboards. Details of the splatting method are well described in (Westover, 1990; Blythe, 1999; Meuller *et. al* 1999). The basic idea for applying it to cloud display is described here. First, the sum of the scattered light reaching from the sun on the viewing ray is calculated. The attenuated light reaching from behind the clouds is also calculated. The light reaching the viewpoint is the sum of those two. Therefore, the color of a voxel depends on the scattered color of the sun, the transmitted color of the sky, and the attenuation due to cloud particles. Calculation of cloud color using splatting is as follows. First, textures for billboards are pre-calculated. Each element of the texture stores the attenuation ratio and cumulative density of the light passing through the metaball. Since the attenuation is not proportional to it, the texture has to be prepared for all meatballs when their center densities are different. However, this requires a large amount of memory. So, the density is discretized into n_q levels and n_q textures are prepared. The value used for n_q is 64. The texture corresponding to the nearest

density of each metaball is mapped onto the corresponding billboard. An image is calculated in two steps using the texture-mapped billboards. In the first step, the intensity of the light is calculated reaching from the sun at each metaball. The shadows of the clouds are also calculated in this step. In the second step, the image viewed from the viewpoint is generated.

The basic idea is to calculate an image viewed from the sun direction to obtain the intensity of light reaching each metaball. First, the viewpoint is placed at the sun position and the parallel projection is assumed. The frame buffer is initialized as 1.0. Then the billboards are stored as a light map texture to cast shadows on the ground. In the second step, the image is generated by using the color of the metaball obtained in the first step. First, all the objects except clouds are rendered. Next, the billboards are faced perpendicularly to the viewpoint and sorted in descending order based on distances from the viewpoint. Then they are projected onto the image plane in back-to-front order. The color in the frame buffer is blended with that of the billboard texture. For blending process, the colors in the frame buffer are multiplied by the attenuation ratio of the billboard texture and then the colors in the texture are added. The same process is repeated for all metaballs.

Shafts of light are caused by particles in the atmosphere. The sunlight passing through gaps in clouds is scattered by the particles. The scattered light, I_s, reaching the viewpoint is recognized as shafts of light. The scattering/absorption due to the atmospheric particles must therefore be taken into account. The intensity of light reaching the viewpoint is obtained by the following equation.

$$I = I_c \beta(T) + \sum_{k=0}^{n_s} \gamma(k\Delta s) I_s(k\Delta s) \beta(k\Delta s) \Delta s \qquad (3.11)$$

where n_s and Δs are the number of sample and the sampling interval. The attenuation, $\beta(T)$, in the first term is calculated analytically by the positions of the viewpoint and each metaball. The color of each metaball is then attenuated by multiplying it. To calculate the second term, spherical shells are considered. Their centers are placed at viewpoint and their radii are determined so that the intervals of shells coincide to Δs. The shells are approximated by a set of polygons to render

them using OpenGL. Polygons outside the viewing pyramid are discarded. Next, the intensity of the light scattered at each vertex and the attenuation ratio of the path between the viewpoint and the vertex are calculated. Then $I_s(k\Delta s)\ \beta(k\Delta s)\ \Delta s$ is stored as the colors of vertices of all the polygons in the viewing pyramid. Finally, the second term is computed by rendering the shells with OpenGL's additive blending function. To render the shafts of light, the colors of the polygons have to be multiplied by attenuation ratio due to clouds, $\gamma(s)$. This can be easily achieved just by mapping the shadow texture onto the polygons using OpenGL's texture mapping function.

Harris (2002) presented a method for realistic real-time rendering of clouds for flight simulators and games using particle systems. He described a cloud illumination algorithm that approximates multiple forward scattering in a pre-process and first order anisotropic scattering at runtime. Impostors are used to accelerate cloud rendering by exploiting frame-to-frame coherence in an interactive flight simulation. The method allows hundreds of clouds with hundreds of thousands of particles to be rendered at high frame rates, and improves interaction with clouds by reducing artifacts introduced by direct particle rendering. The following is details of his method. Harris used Particle system, as it is simple and efficient method for representing and rendering clouds. He assumed that a particle represents a roughly spherical volume in which a Gaussian distribution governs the density falloff from the center of the particle. Each particle is made up of a center, radius, density, and color. He got good approximations of real clouds by filling space with particles of varying size and density. The researcher rendered particles using splatting technique, by drawing screen-oriented polygons texture-mapped with a Gaussian density function. A particle system was selected for cloud representation, but both shading algorithm and fast rendering system are independent of the cloud representation, and can be used with any model composed of discrete density samples in space.

Scattering illumination models presented by Harris (2002) simulate the emission and absorption of light by a medium as well as scattering through the medium. Single scattering models simulate scattering through the medium in a single direction. This direction is usually the direction leading to the point of view. Multiple scattering models are more physically accurate, but must account for scattering in all

directions (or a sampling of all directions), and therefore are much more complicated and expensive to evaluate. In a multiple scattering simulation that samples N directions on the sphere, each additional order of scattering that is simulated multiplies the number of simulated paths by N. Fortunately, as demonstrated by (Nishita *et. al*, 1996), the contribution of most of these paths is insignificant to cloud rendering. Nishita *et al*. found that scattering illumination is dominated by the first and second orders, and therefore they only simulated up to the 4th order. They reduce the directions sampled in their evaluation of scattering to sub-spaces of high contribution, which are composed mostly of directions near the direction of forward scattering and those directed at the viewer. Harris (2002) made further simplification and approximated multiple scattering only in the light direction – or multiple forward scattering – and anisotropic single scattering in the eye direction. The rendering method used by Harris (2002) is a two-pass algorithm similar to the one presented by Dobashi *et. al* (2000). Harris pre-computes cloud shading in the first pass, and uses this shading to render the clouds in the second pass.

For calculation of multiple forward scattering, the first pass of our shading algorithm computes the amount of light incident on each particle. The simplified equation used is as follows.

$$I_k = \begin{cases} g_{k-1} + T_{k-1} \cdot I_{k-1}, & 2 \le k \le N \\ I_0, & k=1 \end{cases} \qquad (3.12)$$

In addition to multiple forward scattering approximation, Harris (2002) implements single scattering toward the viewer as in Dobashi *et. al* (2000). The equation for this is as follows.

$$E_k = S_k + T_k \cdot E_{k-1}, \qquad 1 \le k \le N \qquad (3.13)$$

The above equation says that the light, E_k, exiting any particle p_k is equal to the light incident on it that it does not absorb, $T_k \cdot E_{k-1}$, plus the light that it scatters, S_k. In the first pass the light I_k incident on each particle from the light source is computed. In the second, the portion of this light that is scattered toward the viewer

is calculated. This can be achieved by replacing S_k with $a_k \cdot \tau_k \cdot p(\omega, -l) \cdot I_k / 4\pi$, where ω is the view direction, and $p(\omega, -l)$ is phase function discussed below. This recurrence approximates single scattering toward the viewer.

The phase function, $p(\omega, \omega')$ is very important to cloud shading. Clouds exhibit anisotropic scattering of light (including the strong forward scattering that we assume in our multiple forward scattering approximation). The phase function determines the distribution of scattering for a given incident light direction.

$$p(\theta, g) = \frac{3(1-g^2)}{2(1+g^2)} \frac{(1+\cos^2\theta)}{(1+g^2-2g\cos\theta)^{3/2}} \tag{3.14}$$

where θ is scattering angle and g is an symmetric function which is determined by the cloud condition and the wavelength, if g=0, this function is equivalent to Rayleigh scattering.

The rendering presented by Harris (2002) is a two pass algorithm which is similar to the one presented by Dobashi *et. al* (2000). A shading phase runs once per scene and a rendering phase runs in real time. The key to the implementation is the use of hardware blending and pixel read back. Blending operates by computing a weighted average of the frame buffer contents (the destination) and an incoming fragment (the source), and storing the result back in the frame buffer. This weighted average can be written as.

$$C_{result} = f_{src} \cdot C_{src} + f_{dest} \cdot C_{dest} \tag{3.15}$$

If we let $C_{result} = I_k$, $f_{src} = 1$, $C_{src} = g_{k-1}$, $f_{dest} = T_{k-1}$, and $C_{dest} = I_{k-1}$, then we see that (3.15) and (3.17) are equivalent if the contents of the frame buffer before blending represent I_0. The runtime phase uses the same algorithm, with particles sorted with respect to the viewpoint, and without reading pixels. The pre-computed illumination of each particle I_k is used in this phase to compute scattering toward the eye.

In both passes, particles are rendered in sorted order as polygons textured with a Gaussian "splat" texture. The polygon color is set to the scattering factor $a_k \cdot \tau_k \cdot p(\omega,l) \cdot I_k / 4\pi$ and the texture is modulated by this color. In the first pass, ω is the light direction, and in the second pass it is the direction of the viewer. The source and destination blending factors are set to one and one minus source alpha, respectively.

While the cloud rendering method described above provides beautiful results and is fast for relatively simple scenes, it suffers under the weight of many complex clouds. The games for which we developed this system dictate that we must render complicated cloud scenes at fast interactive rates. Clouds are only one component of a complex game environment, and therefore can only use a small percentage of a frame time. With direct particle rendering, even a scene with ten or twenty thousand particles is prohibitively slow on current hardware. In order to render many clouds made up of many particles at high frame rates, Harris (2002) used dynamically generated impostors. An impostor replaces an object in the scene with a semi-transparent polygon texture-mapped with an image of the object it replaces. To generate impostors, a view frustum is positioned so that its viewpoint is at the position from which the impostor will be viewed, and it is tightly fit to the bounding volume of the object. Then the object is rendered into an image used to texture the impostor polygon.

3.3 Summary

In this chapter, the basic model of particle systems has been discussed. Various advantages of particle systems making it a good choice for simulation of fuzzy objects as compared with other surface-based techniques has been presented. Methods for controlling some of basic features such as particle generation, particle attributes, particle dynamics, particle extinction, and particle rendering have been discussed. Finally, the use of particle systems by some selected researchers in computer graphics has been summarized.

Dobashi *et. al* (2000) used cellular automation (CA) to model clouds data. The simulation space divided into small voxels and each voxel corresponds to a cell used in the cellular automation. At each cell three logical variables are assigned. Cloud evolution is simulated by applying transition rules to each cell at each time step. The cloud density obtained from simulation has two values, that is, 0 or1, whereas in the real world it is continuous from 0 to 1. So, they require calculation for smoothing the density distribution. The whole method for simulation of cloud evolution involves a lot of calculations and is difficult in use. In a contrast, particles system provides a very simple way to model cloud data. Particles system may require a little more computer space for storing particle data, but it needs fewer calculations for modeling and rendering the particles and getting final images of the cloud shapes.

Harris (2002) used particle systems to model clouds. For rendering of clouds he used the approach of Dobashi *et. al* (2000) and extended their model of single light scattering to multiple light scattering model. In order to enhance the rendering process, he used dynamically generated imposters. However, Harris (2002) generated the data for cloud particles with other means such as 3D Studio Max, which is copyright software. The main work of Harris (2002) is on rendering part of clouds.

Virtual reality applications, such as flight simulators, have built in scene stages that are pre-designed. Providing a randomized method along with interactive development environment can prove more useful to these applications.

CHAPTER IV

RANDOMIZED ALGORITHM AND IMPLEMENTATION

4.1 Introduction

This chapter tends to deal with foundation of the research work. At first, it states the proposed system overview, where it explains the modules that are going to be established. A particle system is a collection of many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system.

The purpose of a model of an entity is to allow people to visualize and understand the structure or behavior of the entity (Foley *et al*., 1990). For clouds and gases the model is often implemented as a density function listed below.

$$\rho(x),\ x\ \varepsilon\ R^3 \qquad (4.1)$$

where ρ represents density of matter,
x represents any real number,
ε means belongs to and
R^3 represents volume space.

such that, for each point in space, evaluates to the amount of cloud matter that exists at that point. Particle systems are simple and efficient method for representing clouds. It is assumed that a particle represents a roughly spherical volume in which a Gaussian distribution governs the density falloff from the centre of the particle. Each

particle is made of a centre, radius, density and color. For a good approximation of real clouds, cloud space is filled with particles of varying sizes and density.

Clouds can be built by filling a volume with particles or by using an editing application that allows placing particles and build clouds interactively. The randomized method is a good way to get quick field of clouds.

4.2 Flow Diagram of Shaping Modeling Algorithm

The following is the flow diagram of the proposed algorithm for cloud shape modeling process.

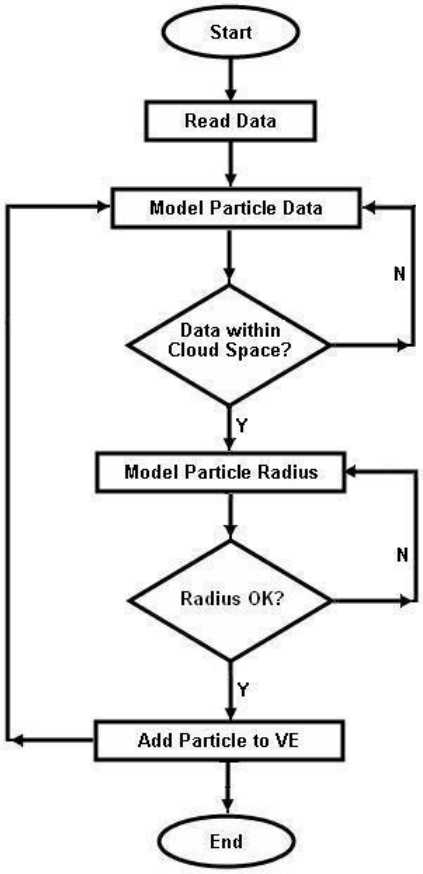

Figure 4.1: Flow diagram of Shape Modeling Algorithm

4.3 Input Data

Data required for the cloud shape modeling algorithm consists of the attributes of the particles making clouds and coordinates of 3-D space in virtual environment that contains particle clouds. The data inputs that are used are described below:

4.3.1 Cloud Centre

Cloud Centre consists of coordinates in 3D space. It consists of X-Coordinate, Y-Coordinate, Z-Coordinate values representing the centre of cloud bounding volume.

4.3.2 Number of Particles

It represents the total number of particles that are randomly distributed in the cloud bounding volume to make shape of a cloud. Each particle is consists of a number of attributes such as centre, radius, density and color.

4.3.3 Radius

It represents the maximum value for radius of cloud particles. In order to get good approximation of cloud, particles of varying sizes are used to make cloud. The radius of each particle is randomly calculated that is les than or equal to this value.

4.3.4 Data File Format

In the following, the organization of the input data files is discussed. This data file serves as input data files for the proposed cloud shape modeling algorithm. Table 4.1 shows the format of a data file. On the first line, it has total number of particles in the system. On the second line, it has total number of cubes in the system. Rest of each line contains the information about each cube such as particles in a cube, cube dimension, cube centre and radius of particles in that cube.

Table 4.1: Format of input data file

Line Number	**Input Data**
1	<Total Particles>
2	<Total cubes>
3	<Particles, Cube(height, width, length), Center(x,y,z), Radius>

The description of each and every piece of data and its type is as described below:

TotalParticles:

It represents the total number of particles in the system. The *integer* data type is used for this data.

TotalCubes:

It represents the total number of cubes used in the system to build the apparent shape of the clouds. The *integer* data type is used for this data.

Particles:

It represents the total number of particles to be distributed in a particular cube. The *integer* data type is used for this data.

Length:

It represents length of a particular cube. The *float* data type is used for this data.

Height:

It represents height of a particular cube. The *float* data type is used for this data.

Width:

It represents width of a particular cube. The *float* data type is used for this data.

X-Coordinate for Centre:

It represents X-Coordinate for the centre of a particular cube. The *float* data type is used for this data.

Y-Coordinate for Centre:

It represents Y-Coordinate for the centre of a particular cube. The *float* data type is used for this data.

Z-Coordinate for Centre:

It represents Z-Coordinate for the centre of a particular cube. The *float* data type is used for this data.

Radius:

It represents maximum value for the radius of particles in a particular cube. The *float* data type is used for this data.

4.4 Data Acquisition

This research has come up with an editing application – namely cloud editor, to allow design of the cloud shapes. The approach used allows the design of cloud macro shapes with the help of cubes. Any number of cubes can be used to design cloud shapes. The dimension and size of cubes can be changes by the use of various control of the cloud editor. After finishing design, the data related to cubes such as cube centre, cube length, cube height and cube width, maximum of value for the radius of particles and the total number of particles to be randomly distributed in the cubes are stored in a data file. This data file serves as input data for the proposed cloud shape modeling algorithm.

Table 4.2 shows a sample data file that has a total of 2000 particles in the system and the number of cubes used to model cloud shape is 3.

Table 4.2: A sample of input data file

Line Number	**Input Data**
1	2000
2	3
3	1000, 8.0, 7.0, 3.0, 10.0, 10.0, 10.0, 2.5
4	600, 5.0, 3.0, 3.0, 15.0, 20.0, -20.0, 2.9
5	400, 6.0, 4.0, 2.0, -10.0, -5.0, -10.0, 3.0

According to the Table 4.2, *Line Number* 1 contains the total number of particles in the system and the value for this data is 2000. *Line Number* 2 contains the total number of cubes used to model the apparent shape of the cloud. According to data file, the number of cubes used in the system is 3. The rest of the data file, from *Line Number* 3 to onward, contains the data about the cubes such as number of particles contained in a cube, cube length, cube height, cube width and maximum value for radius for particles in the cube. *Line Number* 3 shows that the number of particles in the first cube is 1000, its length is 8.0, width is 7.0, and height is 3.0. The value for the centre (x, y, z) of this cube is (10.0, 10.0, 10.0). The maximum length for the particle radius is 2.5. *Line Number* 4 shows that the number of particles in the second cube is 600, its length is 5.0, width is 3.0, and height is 3.0. The value for the centre of the cube is (15.0, 20.0, -20.0). The maximum length for the particle radius is 2.9. Similarly, *Line Number* 5 shows that the number of particles in the third cube is 400, its length is 6.0, width is 4.0, and height is 2.0. The value for the centre of the cube is (-10.0, -5.0, -10.0). The maximum length for the particle radius is 3.0.

4.5 Design of Shape Modeling Algorithm

This section presents the general model formulation of cloud shape modeling algorithm based on randomized method. Part of the reason that particle clouds look any good is that it incorporates randomness in a controlled way. Most of the work involved in achieving this is an approximation contained in noise function.

Each particle is assumed to have a roughly spherical volume in which a Gaussian distribution governs the density falloff from the centre of the particle. Each particle has attributes of a centre, radius, density and color. For a good approximation, particles of varying sizes and densities are used build a cloud space.

4.5.1 Model Formulation

The model formulation for each step of the cloud shape modeling process is presented using the following notation (Gunadi, 2003).

N = Total number of particles.
X_L = Length of cloud bounding volume along x-axis.
Y_L = Length of cloud bounding volume along y-axis.
Z_L = Length of cloud bounding volume along z-axis.
R_{max} = Maximum value for radius of particles.
X_C = X-coordinate for centre of cloud space.
Y_C = Y-coordinate for centre of cloud space.
Z_C = Z-coordinate for centre of cloud space.
R(i) = Radius of particle i.

The model formulation is as follows.

Step 1:

For each particle, evaluate the location of particles in cloud bounding volume. Let X(i), Y(i) and Z(i) represent the X-coordinate, Y-coordinate and Z-coordinate respectively for the location of particle i. Then these values are calculated as follows:

$$X(i) = \text{random}(s)$$
$$Y(i) = \text{random}(s)$$
$$Z(i) = \text{random}(s) \qquad (4.2)$$

where i=0,1,2, …, N-1,
s is seed for random-number generator and sets starting point for generating a series of pseudorandom numbers, and
random(s) is a function takes seed as an argument and generates a random number.

The constraints on the values are

$$\begin{aligned} X(i) &\le X_L, && \text{for all } i \\ Y(i) &\le Y_L, && \text{for all } i \\ Z(i) &\le Z_L, && \text{for all } i \end{aligned} \qquad (4.3)$$

In order to transform X(i), Y(i) and Z(i), for all values of i, to meet above constraints, we compute as;

$$\begin{aligned} X(i) &= X(i) \,\%\, X_L \\ Y(i) &= Y(i) \,\%\, Y_L \\ Z(i) &= Z(i) \,\%\, Z_L \end{aligned} \qquad (4.4)$$

where % is the modulus operator.

To add randomness, we add phase shifts in X(i), Y(i) and Z(i). Let P_x, P_y and P_z be the phase shifts, then their values are calculated according to the following relationships (Gardner, 1985):

$$\begin{aligned} P_x &= \pi/2 \, \mathrm{Sin}(0.5 \, Y(i)) \\ P_y &= \pi/2 \, \mathrm{Sin}(0.5 \, X(i)) \\ P_z &= \pi/2 \, \mathrm{Sin}(0.5 \, Z(i)) \end{aligned} \qquad (4.5)$$

The phase shifts produce a controlled pseudo-random effect by shifting the X component as a function of Y and Y component as a function of X component. To provide three dimensional variations, the phase shifts are augmented by added sine variations with Z component (Gardner, 1985):

$$\begin{aligned} P_x &= P_x + \pi \, \mathrm{Sin}(P_x * Z(i)/2) \\ P_y &= P_y + \pi \, \mathrm{Sin}(P_x * Z(i)/2) \end{aligned} \qquad (4.6)$$

Then the phase shift values calculated above added to X(i), Y(i) and Z(i) to get their new values as listed below:

$$X(i) = X(i) + P_x$$
$$Y(i) = Y(i) + P_y$$
$$Z(i) = Z(i) + P_z \quad (4.7)$$

Finally, the values of coordinates for the location of particle that fits within the cloud bounding volume in virtual environment with respect to cloud centre are calculated by adding the value of cloud centre (X_c, Y_c, Z_c), as shown by the equations listed below:

$$X(i) = X_c + X(i)/2$$
$$Y(i) = Y_c + Z(i)/2$$
$$Z(i) = Z_c + Z(i)/2 \quad (4.8)$$

Step 2:

Calculate the radius of each particle. Each particle has a different value of radius which is less than or equal to a preset value R_m. Let R(i) represents radius of particle i, then the value for radius of the particle is calculated as follows:

$$R(i) = \text{random(seed)} \quad (4.9)$$

where i = 0,1,2, …, N-1.
and random(seed) is a function takes seed as an argument and generates a random number.

The constraints on the values are

$$R(i) \leq R, \quad \text{for all i} \quad (4.10)$$

In order to transform R(i), for all values of i, to meet above constraint, we compute as;

$$R(i) = R(i) \,\%\, R \quad (4.11)$$

where % is the modulus operator.

In order to get randomness in the values of radius, some noise is added to it as listed below:

$$R(i) = R(i) + \pi/2 \text{ Sin } (R(i)) \quad (4.12)$$

Step 3:

Add the particle to cloud and repeat the process of generating data for new particles according to step 1 and step 2 till total number of particles (N) is reached.

4.5.2 Pseudo Code of Shaping Modeling Algorithm

The following is the pseudo code of the proposed algorithm for cloud shape modeling process.

```
get no_of_clouds
for each cloud
        get cloud_center
        get no_of_particles
        get cloud_bounding_volume
        for n=1 to no_of_clouds
                get center(x,y,z) of a particle
                if (center lies in cloud_bounding_volume)
                        add particle to cloud space
                else    get a new particle
                end if
        next
next
```

Figure 4.2: Pseudo code for Shape Modeling Algorithm

4.6 Implementation of Randomized Algorithm in Cloud Editor

The evolving structure of clouds is modeled using a **two-level** approach. For controlling the general **shape** of clouds, cubes have been used. Various cubes can be sized and arranged in order to create a desired shape of clouds. For **low-level** cloud details particles system has been used to ill he cloud volume with particles.

In order to evaluate the shapes modeled by the proposed algorithm, we make use of an interactive editing application – **cloud macrostructure editor** and use it to design the apparent shape of clouds. The snapshot of cloud macrostructure editor is shown in Figure 5.1 and the further details about this editor are presented in Appendix A.

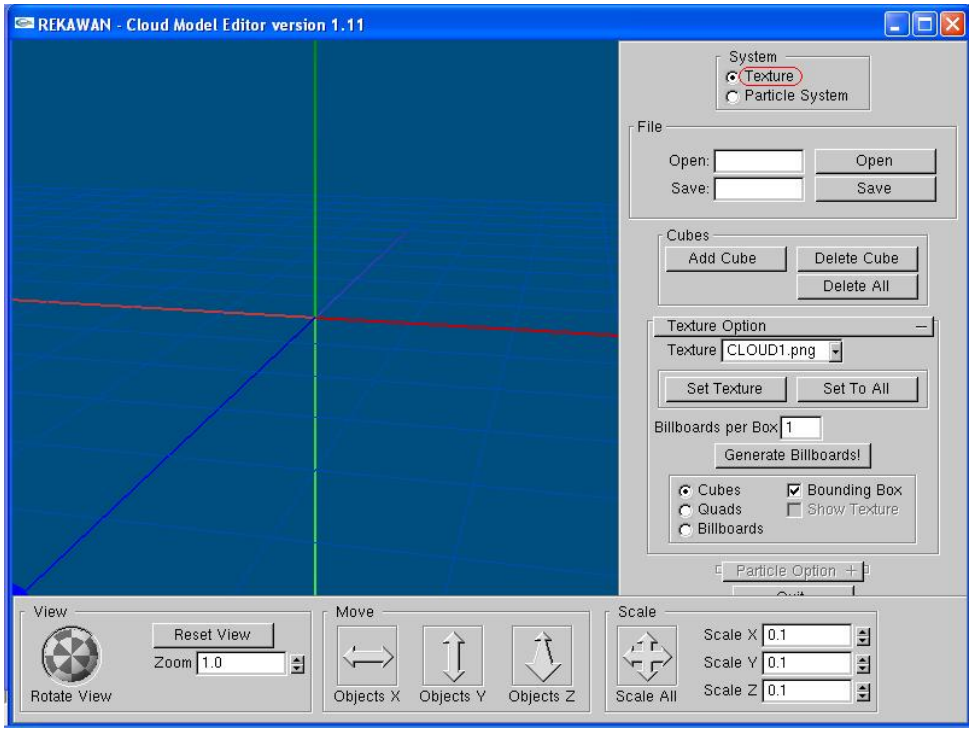

Figure 4.3: Snapshot of Cloud Macrostructure Editor

To design cloud shapes, we can add cubes by pressing '**Add Cube**' button from the control group named as '**Cubes**'. The attached parameters with each cube are its length, width, breadth, centre and number of particles and radius of particles. We can use any number of cubes to design cloud apparent shape.

After finishing the design of the cloud apparent shape, the data related with these cubes is used by the proposed algorithm and it models the microstructure of the cloud by filling the cloud space with particles of varying sizes at random position.

In the following Figure 5.2 shows the snapshot of cloud editor showing a number of cubes placed at different location and having different number of particles and size of particle radius. The cube in red color is active one and its attributes such as size, location, number of particles and size of radius can be changed by using various controls.

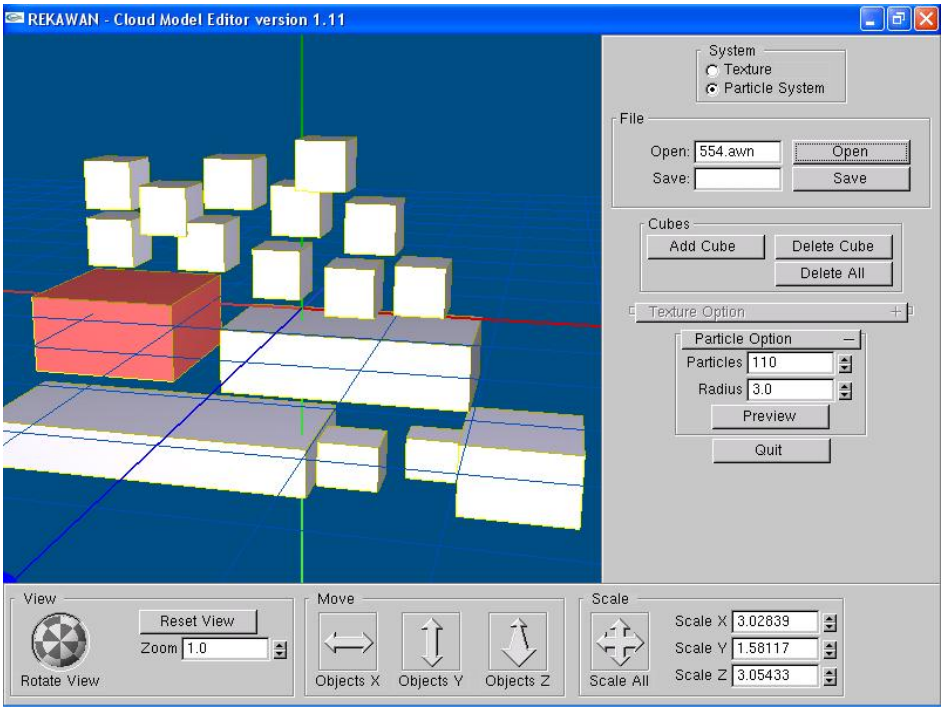

Figure 4.4: Using 17 cubes for clouds apparent shape

In the following, Table 5.1 shows the data file generated by cloud macrostructure editor using 17 cubes as shown in Figure 5.2. The total number of

particles used to model cloud in this system is 554. The details about data file are discussed in section 4.2.4.

Table 4.3: Data file generated by snapshot of Figure 4.4

Line Number	Input Data
1	554
2	17
3	200, 7.39642, 1.07048, 2.89284, -3.4, -3.3, -0.600002, 2.1
4	10, 1.12383, 0.893671, 0.750589, 2.8, -3.1, -0.4, 2.2
5	5, 1.29472, 0.981624, 1, 1.1, -3.2, 0, 2
6	100, 4.99599, 1.30149, 2.33467, 1.1, -0.899999, 0, 3
7	110, 3.02839, 1.58117, 3.05433, -3.8, -0.499999, 0, 3
8	10, 1, 1, 1, 2.5, 0.600001, 0, 2
9	8, 1, 1, 1, -0.399999, 0.800001, 0, 3
10	7, 1, 1, 1, 1.1, 0.500001, 0, 2.2
11	5, 1, 1, 1, 1.5, 1.8, 0, 2.4
12	8, 1, 1, 1, -2, 1.3, 0, 2.123
13	9, 1, 1, 1, 6.12438e-007, 2.1, 0, 2
14	50, 2.36429, 1.50285, 2.29693 4.6, -3, 0, 2.2345
15	8, 1, 1, 1, -4, 2.5, 0, 2
16	0, 1, 1, 1, 0.500001, 3, 0, 2.5
17	7, 1, 1, 1, -4, 1.3, 0, 2
18	7, 1, 1, 1, -2.8, 2, 0, 2
19	10, 1, 1, 1, -1.4, 2.6, 0, 2

In the following, Figure 4.5 shows the snapshot of cloud modeled by 554 particles according to data generated by the cloud macrostructure editor shown in Figure 4.4. The data used by the proposed algorithm to model cloud is shown in Table 4.3 By making use of data of Table 4.3, the cloud shape modeling algorithm modeled cloud data. This cloud modeled data was rendered in the renderer presented by Harris (2002), Figure 4.5 shows this rendered image.

Figure 4.5: Cloud modeled with 554 particles

In the following Figure 4.6 shows another snapshot of cloud editor showing 23 cubes placed at different locations to represent apparent shape of cloud.

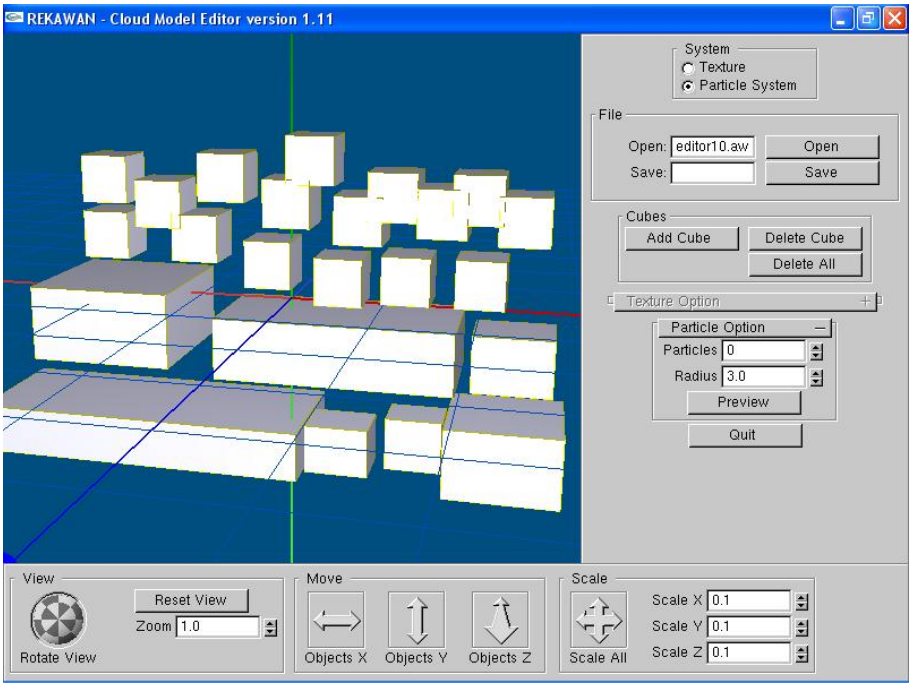

Figure 4.6: Using 23 cubes for clouds apparent shape

In the following, Table 4.4 shows the data file generated by cloud macrostructure editor using 23 cubes as shown in Figure 4.6. The total number of particles used to model cloud in this system is 615.

Table 4.4: Data file generated by snapshot of Figure 4.6

Line Number	Input Data
1	615
2	23
3	200, 7.39642, 1.07048, 2.89284, -3.4, -3.3, -0.600002, 2.1
4	10, 1.21266, 1.14308, 0.963248, 2.7, -3.1, -0.4, 2.2
5	5, 1.36688, 1.05379, 1.07216, 1.1, -3.2, 0, 2
6	100, 4.99599, 1.30149, 2.33467, 1.1, -0.899999, 0, 3
7	110, 3.02839, 1.58117, 3.05433, -3.8, -0.499999, 0, 3
8	10, 1, 1, 1, 2.5, 0.600001, 0, 2
9	8, 1, 1, 1, -0.399999, 0.800001, 0, 3
10	7, 1, 1, 1, 1.1, 0.500001, 0, 2.2
11	5, 1, 1, 1, 1.5, 1.8, 0, 2.4
12	8, 1, 1, 1, -2, 1.3, 0, 2.123
13	9, 1, 1, 1, 6.12438e-007, 2.1, 0, 2
14	50, 2.36429, 1.50285, 2.29693, 4.6, -3, 0, 2.2345
15	8, 1, 1, 1, -4, 2.5, 0, 2
16	10, 1, 1, 1, 0.500001, 3, 0, 2.5
17	7, 1, 1, 1, -4, 1.3, 0, 2
18	7, 1, 1, 1, -2.8, 2, 0, 2
19	10, 1, 1, 1, -1.4, 2.6, 0, 2
20	10, 1, 1, 1, 4.8, 1.9, 0, 2
21	7, 1, 1, 1, 3.2, 2, 0, 2
22	10, 1, 1, 1, 4, 0.600001, 0, 2.1
23	9, 1, 1, 1, 2.2, 2.3, 0, 2
24	5, 1, 1, 1, 3.9, 2.3, 0, 2.123
25	10, 1.62329, 1.19516, 1.31986, 4.7, -0.999999, 0, 3

Similarly, the following Figure 4.7 shows the snapshot of cloud modeled by 615 particles according to data generated by the cloud macrostructure editor shown

in Figure 4.7. The data used by the proposed algorithm to model cloud is also shown in Table 4.4. The image shown in Figure 4.7 was also rendered by the renderer presented by Harris (2002).

Figure 4.7: Cloud modeled with 615 particles

4.7 Summary

A cloud shape modeling algorithm is proposed to address the problem in the area of cloud shape modeling. By proper selection of cube sizes and position, number of particles in cubes, and particles radius, the proposed algorithm is able to model almost all types of cloud shapes. The randomized method is used to model the algorithm.

As the Virtual Reality application have level of details and these details need be designed by the designers. The proposed framework includes an interactive editing application – **cloud macrostructure editor** (Muhammad Azam Rana *et al*., 2004), which allows designers to model the details of cloud scene interactively and

in a very easy manner. Cloud macrostructure editor can easily be used to design the details of each stage and generate data files for each stage. Then the proposed algorithms can be used to model the cloud scenes by using the data for each stage.

However, the proposed algorithm cannot model cloud dynamics. To make the problem simple, only static clouds have been addressed in this research. The target of this research is to model clouds for Virtual Environments such as flight simulators and games etc. and this application do not need much about cloud dynamics. Future research may address this problem and extend the proposed algorithm to incorporate cloud dynamics.

CHAPTER V

TESTING AND RESULTS

5.1 Introduction

As is true for any object or phenomenon, there are multiple ways to model clouds. An explicit representation of every water droplet in a cloud requires far too much computation and storage, so most of the researchers have used much coarser models. In general, the most common methods that have been used to model and render clouds are particle systems, metaballs, voxel volumes, procedural noise, and textured solids.

Particle systems model objects as a collection of particles – simple primitives that can be represented by a single 3D position and a small number of attributes such as radius, color, and texture etc.

Particles can be created by hand using a modeling tool, procedurally generated or created with some contribution of the two. Particles have the advantage that they usually require only very simple and inexpensive code to maintain and render. Because a particle implicitly represents a spherical volume, a cloud built with particles usually requires much less storage than a similarly detailed cloud represented with other methods. This advantage may diminish as detail increases, because many tiny particles are needed to achieve high details.

This chapter discusses the evaluation of the proposed algorithm, test for efficiency of randomized algorithm and summarizes rendering performance test results of the research. A number of interesting issues are presented. The discussion of the proposed algorithm is also presented.

5.2 Efficiency of the Randomized Algorithm

We have performed test for efficiency of the randomized algorithm. For this purpose, process time taken by the algorithm to model the cloud data has been calculated for a number of particles. The machine used for this purpose is Intel Pentium® IV 3.2 GHz having 1 GB RAM. Figure 5.1 through Figure 5.3 show the graphs for these tests. Figure 5.1 shows resulting graph for a Pentium® IV 3.2 GHz Processor having 1 GB RAM. Figure 5.2 shows resulting graph for a Pentium® IV 2.0 GHz Processor having 248 MB RAM. Figure 5.3 shows resulting graph for a Pentium® III 797 MHz Processor having 128 MB RAM. It is obvious that as the number of particles is increased, the process time taken by the algorithm to model cloud data is almost increasing linearly.

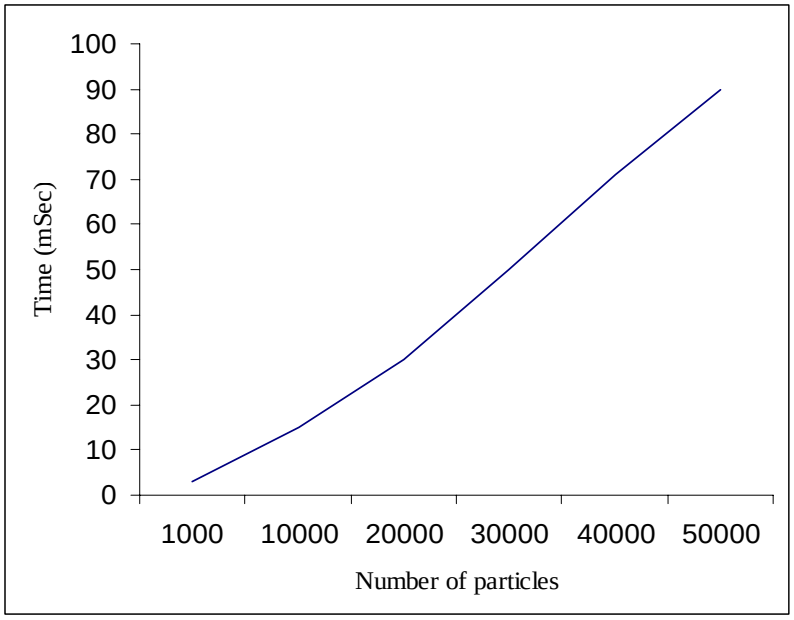

Figure 5.1: Test result for efficiency of the algorithm

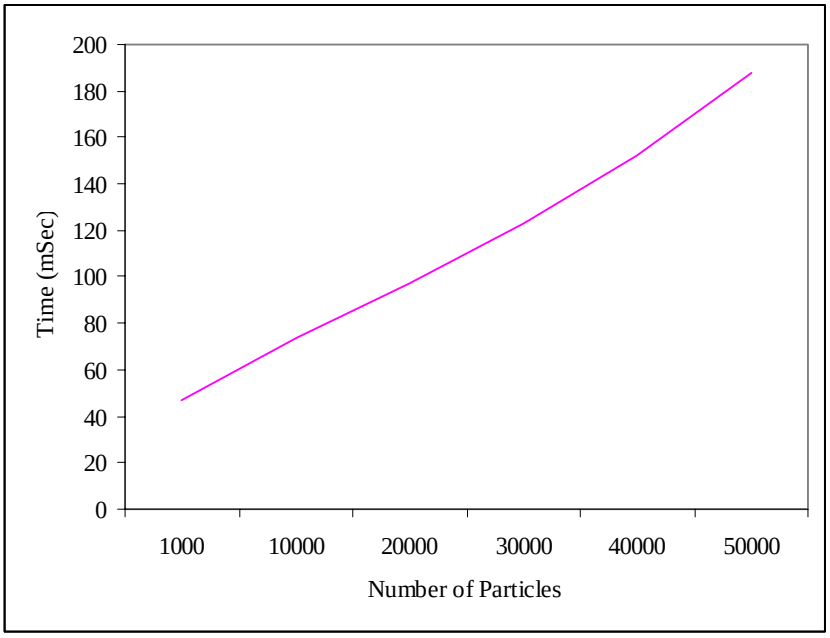

Figure 5.2: Test result for efficiency of the algorithm

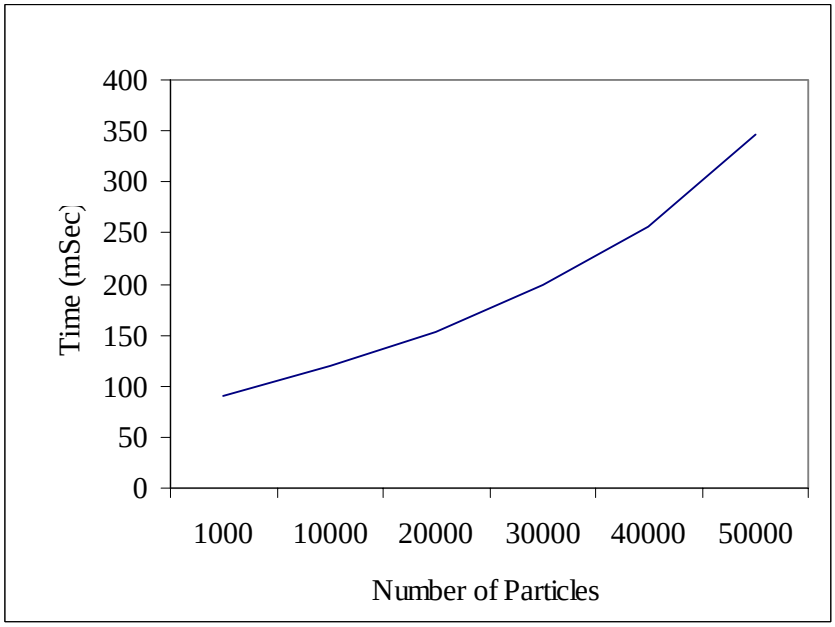

Figure 5.3: Test result for efficiency of the algorithm

5.3 Rendering Performance Tests

We have implemented our cloud shape modeling system using OpenGL. The rendering of the clouds has been done using the method described by Harris and Lastra (2001), Harris (2002) and Harris *et al*. (2003).

We have performed several performance tests for rendered images of clouds modeled by our proposed randomized cloud shape modeling algorithm. Our first test machine was Pentium® IV processor; the details of this system - *System1* are listed in the following Table 5.1.

Table 5.1: Specifications of Test System – System1

Processor	Intel Pentium IV 3.2 GHz
System RAM	1 GB
Graphics Card	nVADIA GeForce FX 5950 Ultra
RAM on GPU	256 MB

In this test, we created the data for cloud particles using our proposed randomized algorithm, distributed the cloud particles randomly in the cloud bounding volume and then rendered the resulting cloud images. Figure 5.2 shows the results of the performance test for a resolution of 800×600, 960×600 and 1024×876. The tests have been performed for a maximum number of 50000 particles. The chart in Figure 5.2 shows the graphs for these test results.

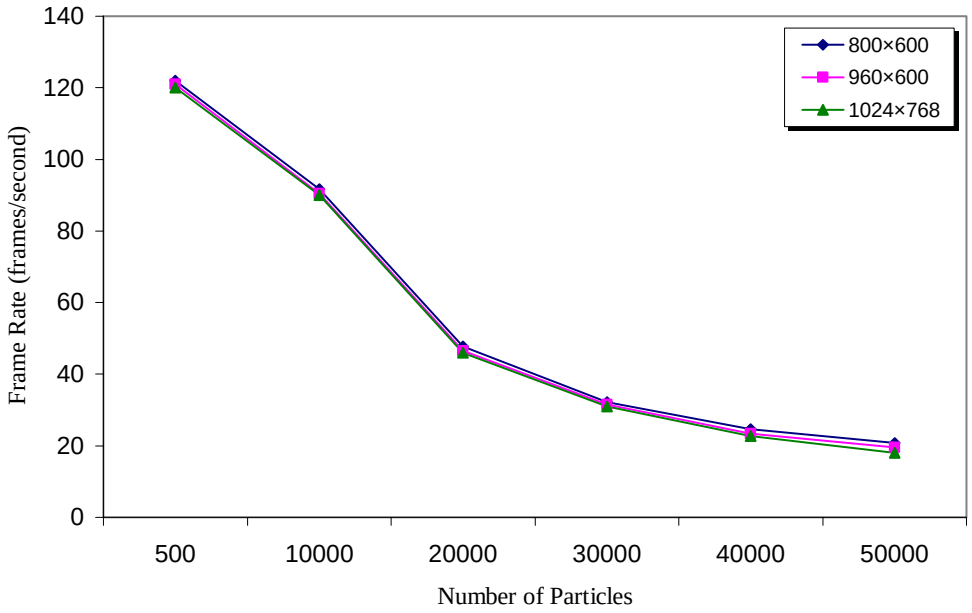

Figure 5.4: Performance Test results for System1

Figure 5.4 shows the test results for the system having Intel Pentium IV 3.2 GHz, 1GB RAM and AGP as nVADIA GeForce FX 5950 Ultra. It is observed that for a less number of particles such as 500 particles, the frame is more than 120 frames per second whereas for a large number of particles we have frames rate around 20 frames per second. It has been observed that a small piece of cloud can be modeled by using a number of particles around 500 to 1000. This shows that even for scenes consisting of several thousands particles we can achieve interactive frame rates and the proposed method is applicable to design cloud shapes interactively.

In order to further test performance of proposed algorithm, a number of performance tests were conducted on other machines having different specifications and with a different variety of AGP card. The specification of these test machines are listed in the following tables from Table 5.2 to Table 5.5. These systems have been labeled as *System2, System3,* System4 and *System5*.

Table 5.2: Specifications of Test System – System2

Processor	Intel Pentium IV 1.5 GHz
System RAM	384 MB
Graphics Card	ATI Radeon 9600 XT
RAM on GPU	128 MB

Table 5.3: Specifications of Test System – System3

Processor	Intel Pentium IV 1.4 GHz
System RAM	256 MB
Graphics Card	nVADIA GeForce FX 5200
RAM on GPU	128 MB

Table 5.4: Specifications of Test System – System4

Processor	Intel Pentium IV 2.8 GHz
System RAM	512 MB
Graphics Card	nVADIA GeForce FX 5200
RAM on GPU	128 MB

Table 5.5: Specifications of Test System – System5

Processor	AMD Athlon™ X 1600 + 1.4 GHz
System RAM	256 MB
Graphics Card	nVADIA GeForce MX
RAM on GPU	64 MB

Figure 5.5 shows the results of the performance test for *System2* for a resolution of 800×600 and 1024×876. The hardware on this system only supported these two mentioned resolution modes and the other resolution modes were not available for testing.

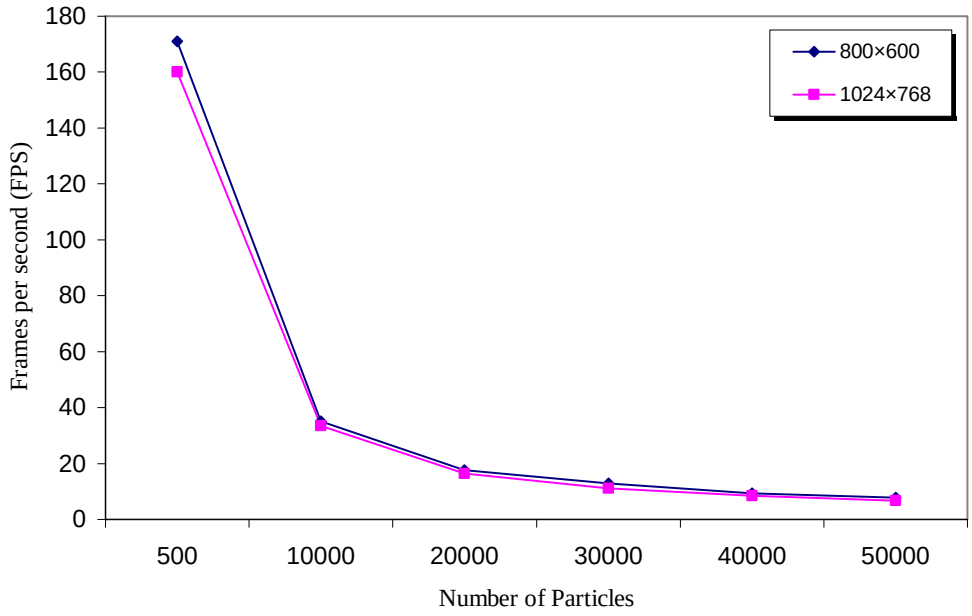

Figure 5.5: Performance Test results for System2

Figure 5.5 shows the test results for a test system having processor as Pentium IV 1.5 GHz, 384 MB RAM and AGP card as ATI Radeon 9600 XT. As shown in the graph, for less number of particles, the frame rate is very good. Between 500 particles and 10000 particles, the frame rate drops quickly and then it drops down linearly. The AGP card used in this system is ATI Radeon 9600 XT. This type of behavior looks due to limitation of computational power of AGP card. For higher number of particles, the frame rate drops below 20 frames per second, but even then this is quite reasonable to design clouds interactively on low power PCs.

Figure 5.6 shows the results of the performance test for *System3* for a resolution of 800×600, 1024×876, 1088x612 and 1152x864.

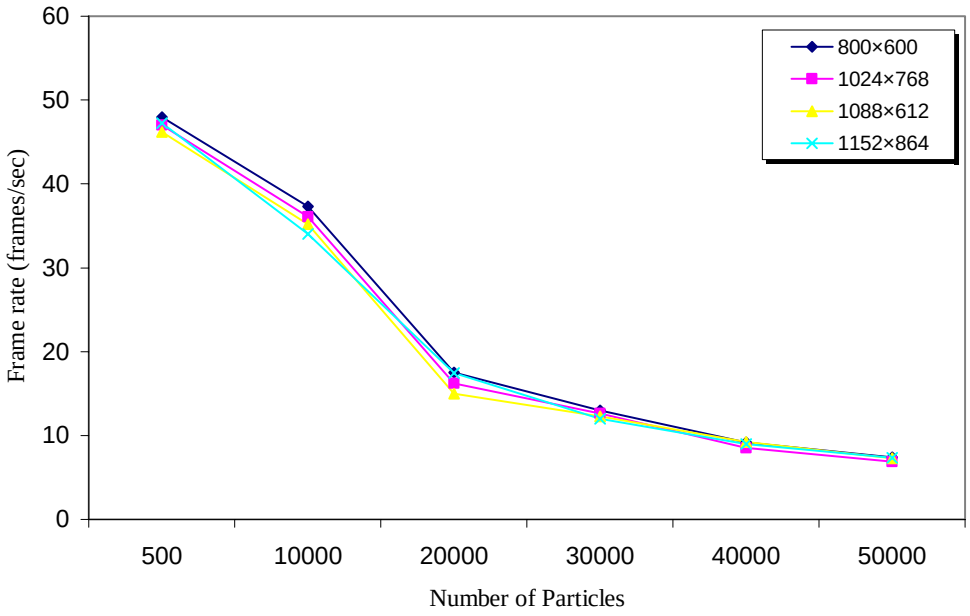

Figure 5.6: Performance Test results for System3

Figure 5.6 shows the test results for a test system having processor as Pentium IV 1.4 GHz, 256 MB RAM and AGP card as nVADIA GeForce FX 5200. As shown in the graph, for less number of particles, the frame rate is closer to 50 frames per second. For higher number of particles, the frame rate drops closer to 10 frames per second. In the series of tests, this is the low power machine and looks reasonable to design cloud images interactively using the proposed interactive environment proposed by this research.

Figure 5.7 shows the results of the performance test for *System4* for a resolution of 800×600 and 1024×876. The hardware on this system only supported these two mentioned resolution modes and the other resolution modes were not available for testing.

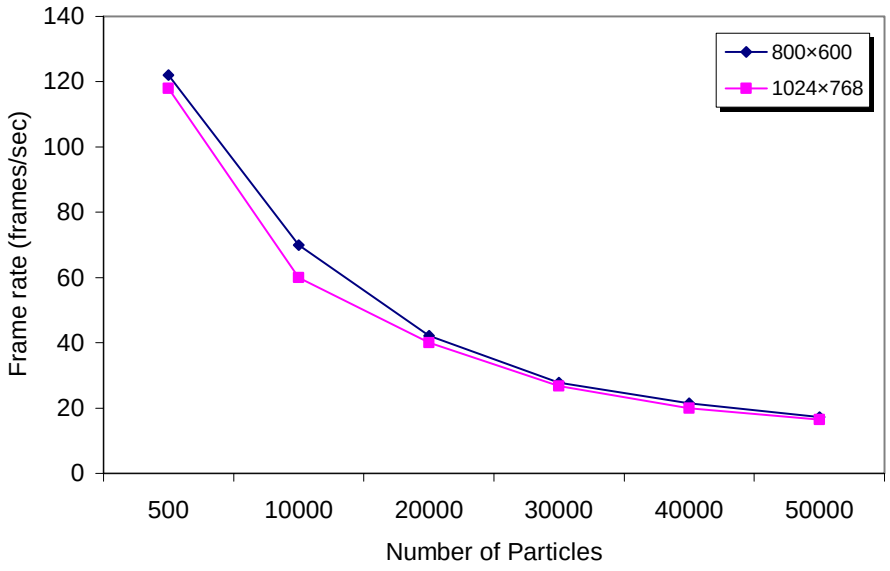

Figure 5.7: Performance Test results for System4

Figure 5.7 shows the test results for a test system having processor as Pentium IV 2.8 GHz, 512 MB RAM and AGP card as nVADIA GeForce FX 5200. As shown in the graph, for less number of particles, the frame rate is closer to 120 frames per second. For higher number of particles, the frame rate drops closer to 20 frames per second. As seen on the graph, the difference in the frame rate for the two mentioned resolution is very close and is insignificant.

Figure 5.8 shows the results of the performance test for *System5* for a resolution of 800×600, 1024×876 and 1152×864.

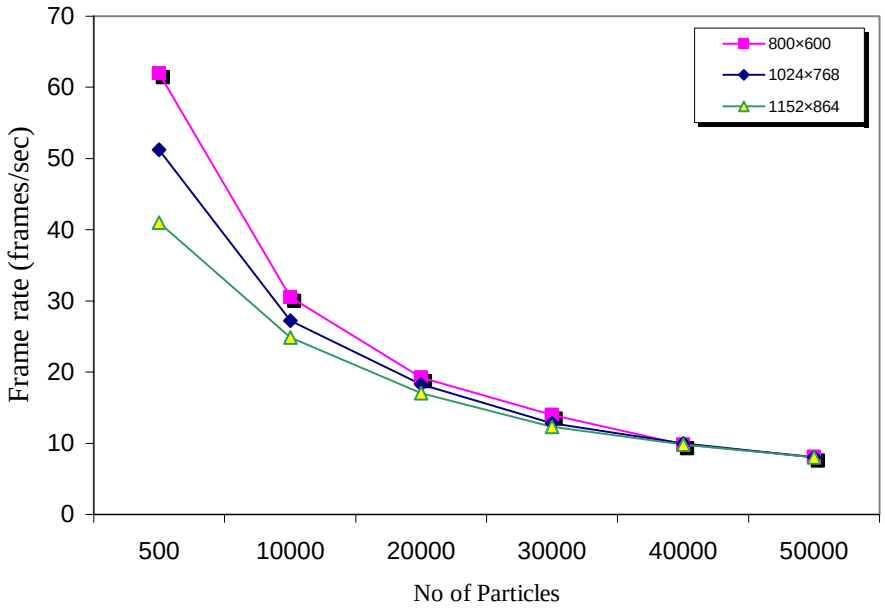

Figure 5.8: Performance Test results for System5

Figure 5.8 shows the test results for a test system having processor as AMD Athlon™ X 1600 + 1.4 GHz, 256 MB RAM and AGP card as nVADIA GeForce MX. As shown in the graph, for less number of particles, the frame rate is closer to 120 frames per second. For higher number of particles, the frame rate drops closer to 10 frames per second. As seen on the graph, the difference in the frame rate for les number of particles is considerable but this difference is insignificant for higher number of particles in the system.

In the following, figures 5.9 through 5.11 show the comparison of rendering performance test for 800×600, 1024×768 and 1152×864 resolutions.

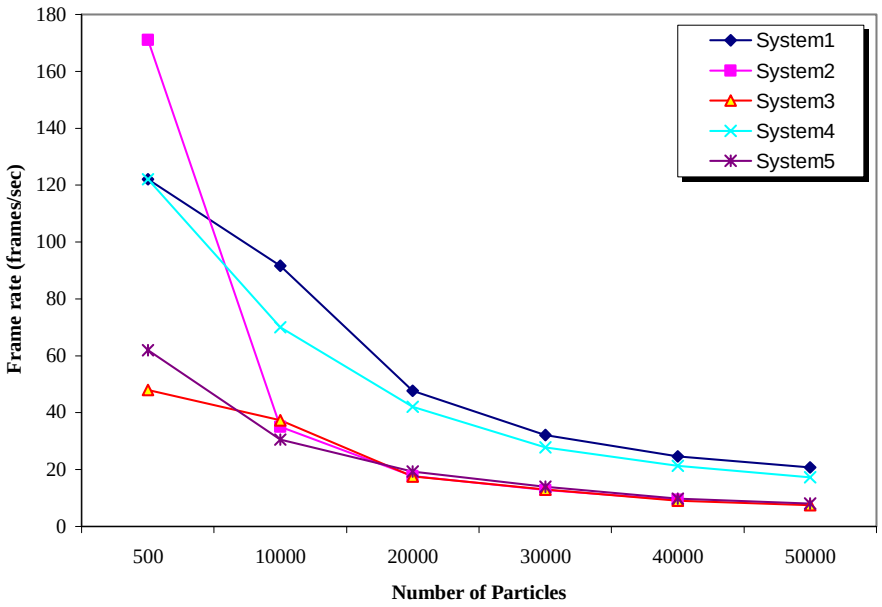

Figure 5.9: Comparison of Performance Test for 800×600 resolution

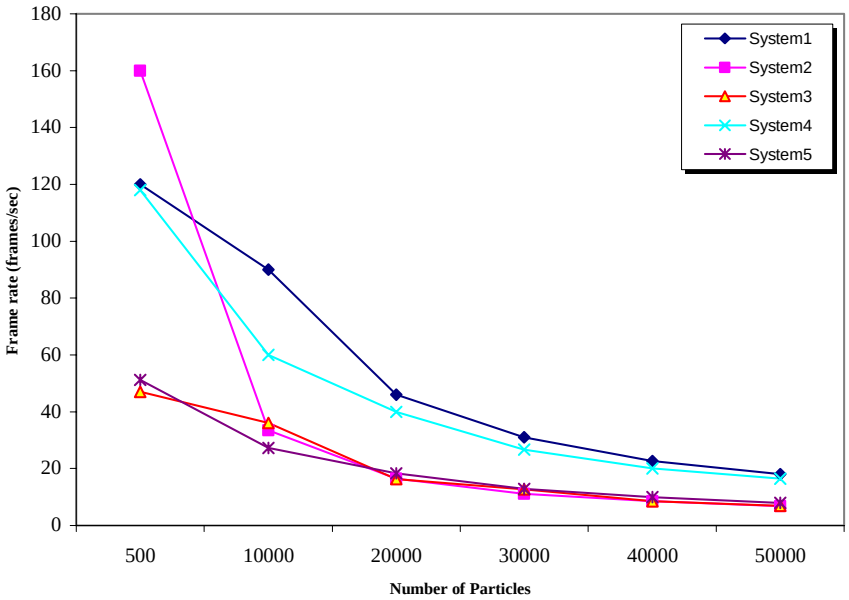

Figure 5.10: Comparison of Performance Test for 1024×768 resolution

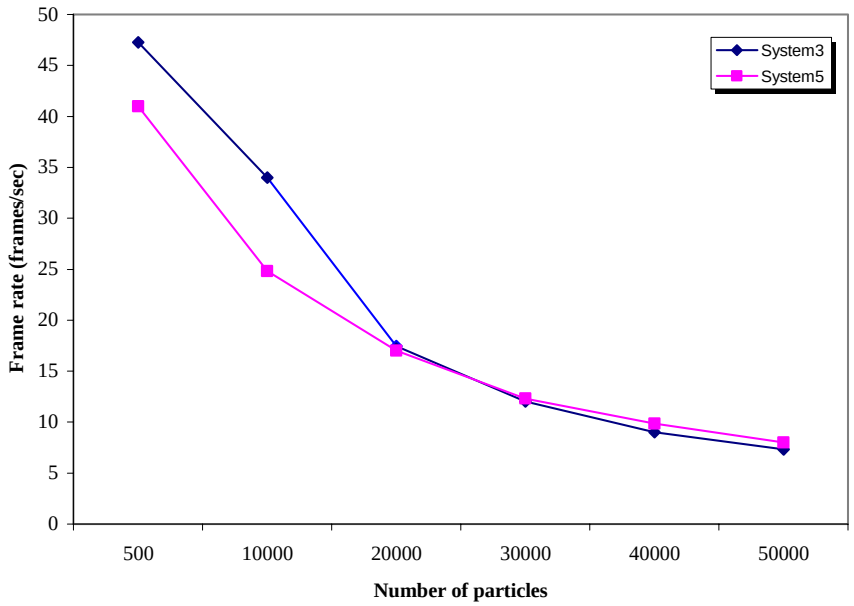

Figure 5.11: Comparison of Performance Test for 1152×864 resolution

Figure 5.9 and figure 5.10 show comparison of frame rate per second for all systems for the resolution of 800×600 and 1024×768 respectively. For the number of particles up to 10000, the frame rate is more than required for real time rendering. For particles more than 20000, the frame rate is close to real time. For the System1 and System4, frame rate is close to real time even for 50,000 particles. System2, System3 and System5 have slightly less frame rate than required for real time as the number of particles increases from 20000; it is due the les power of Graphical Processing Units (GPU). The frame rate greater than real time shows that the modeled clouds can be used in a virtual environment (VE), as clouds are one small part of a VE and should get less time for their processing. Figure 5.11 shows comparison of frame rate at 1152×864 for System3 and System5 only, as the GPUs on other test machines don't support these resolution modes. The frame rate is dropped as the resolution is increased as expected. From above figures, we can deduce that the resolution mode of 600×800 is most sited for the efficient rendering of the clouds.

5.4 Summary

This chapter presents performance test results and discuses these results produced by the research that models the shape of clouds by proposed randomized algorithm. Specification of the test machines and the charts of these performance tests have been presented and discussed.

The efficiency of the randomized particles algorithm has been performed. The time taken for particles up to 1000 particles is about 1millisecond. Even for 50000 particles, the time taken by the algorithm is around 90 milliseconds. For modeling a reasonable good scene of clouds, a number of particles around 10000 to 20000 is enough, which shows that algorithm can model cloud data very quickly from stage to stage as needed by the VE applications, without interrupting the user.

The tests for the performance of the proposed cloud shape modeling algorithm are run on various low cost PCs having a variety of processors such as Intel Pentium IV having CPUs varying from 1.4 GHz to 3.2 GHz and of from different and an AMD Athlon 1.4 GHz processor. These test machines have different ordinary Accelerated Graphics Port (AGP) cards. The complete specifications of these test machines are listed in Chapter V. As shown in Figure 5.2 - 5.9, for less number of particles the achieved frame rates are more than hundred frames per second. This frame rate is more than required for real time application. The reason is as we are testing clouds only, whereas in a VE clouds are just a small part and should demand a less processing time. This shows the suitability of the modeled clouds for VE. Even for larger number of particles, the frame rates are in the range of 20-10 frames per second which looks reasonable for synthesizing cloud shapes interactively. This shows that the cloud shape modeling framework proposed by this research is reasonable good for synthesizing cloud shapes interactively for virtual reality applications on low-cost PCs.

The performance testing also shows that as we increase the resolution of the display, the rending time for the scenes is increased and 600×800 is the most suitable resolution for getting fast rendered cloud images.

CHAPTER VI

DISCUSSION AND CONCLUSIONS

6.1 Introduction

This chapter discusses and concludes the work introduced in this research, and recommends the future work. The topics discussed include Discussion, Conclusion of the introduced work, recommendation for the future work and Summary.

6.2 Discussion

This study describes a system for real-time simulation of cloud shapes suitable for interactive Virtual Environments (VE) applications such as flight simulators and games. These applications require realism, but cannot afford to sacrifice speed to achieve it. Controlling shapes of clouds is a difficult task as they are amorphous objects. Clouds can be built by filling a volume by particles or by using an editing application that allows users to place particles, have control over number of particles, and have control over size that can build clouds interactively.

The randomized method is a good way to get a quick field of clouds. A combination of randomized method and an editing application serves a good choice for designing cloud for Virtual Reality (VR) applications as they have a range of

designed levels and need more control over details of scenes. Providing editing applications for such Virtual reality applications can produce beautiful cloud shapes tailored to the need of the Virtual Environment (VE). The resulting images of clouds shapes, modeled by the proposed algorithm have been demonstrated. It shows these images are promising, easy to design using the proposed framework and almost any type of clouds can be modeled. The proposed framework and algorithm are developed with these requirements in mind.

Thus, this research has demonstrated that realistic cloud shapes can be simulated and rendered in real time using efficient algorithms implemented entirely on programmable graphics processors. To the knowledge, this work is the first in Computer Graphics to model cloud shapes with particle systems based on randomized method.

6.3 Conclusions

This chapter discusses and concludes the study of the research. Contributions made by this study are described. Finally, limitations of this work and suggestions for future work have also been discussed.

The focus of this study is modeling of realistic clouds virtual applications such as games and flight simulators. These applications demand realism regarding presence of clouds and are not mostly concerned with movement of clouds and changes in their shapes under the influence of atmosphere. So, this research focuses on modeling of static clouds only and cloud dynamics are not studied for simplicity.

This research proposes a two-level approach for modeling cloud shapes. In the second level, cloud shape modeling algorithm is developed. This algorithm is based on randomized method and particles system is used to model cloud data. The data generated by cloud macrostructure editor is used by the proposed cloud shape modeling algorithm for modeling cloud shapes. Choosing proper size of cubes,

number of particles and size of particles, interesting shapes of clouds can be synthesized by the proposed framework.

In the second level, an interactive editing application – **cloud macrostructure editor** has been proposed which helps in designing cloud apparent shape by making use of cubes. This interactive environment enables full control over size and location of cubes in three-dimensional space, number of particles in a cube and size of particles. Using cubes of varying sizes, placing them at appropriate places, distributing proper number of particles and adjusting size of particles, the resulting data about cloud apparent shape is then saved in data file.

Virtual reality applications have pre-designed levels of details. The two-level design approach used in this algorithm has proved successful to interactively design cloud shapes for each stage and then save data about these details in separate files for each level.

6.4 Limitations and Future Work

In this section, the major limitations of this work are listed. Most of these limitations are addresses with the ideas of future works and suggest several directions for future explorations including visual improvements for clouds, creative controls, new directions and applications for general-purpose computation on Graphics Processing Units (GPU).

The most obvious direction for the future is to continue to improve the quality and realism of clouds. A number of limitations and problems with this current work provide goals for future work.

a) The most limitation of proposed cloud shape modeling algorithms is the number of different clouds it can support. It is not currently possible to simulate multiple cloud clusters. Future work can address this issue by dividing cloud particles into a cluster of small clouds and then render only

those clouds that come along the view direction. This will increase the performance of rendering process.

b) The most important limitation is the scale and detail that can be supported. Currently, it is not possible to simulate a sky full of clouds due to large processing requirements. For flight simulators, clouds must extend as far as possible as the user desires to fly. The basic reason for scale and detail limitation is that volumetric data requires immense computation and storage resources. Fortunately, Graphics Processing Units are rapidly increasing both of these resources. This will help, but it will not solve the problem of populating the skies with dynamic clouds. More creative techniques will be required.

c) A possible method of creating higher detail at lower cost is to use procedural noise techniques. Much work has been done in the past on generating clouds using noise (Lewis, 1989; Ebert, 1997; Ebert et al., 2002; Schpok et al., 2003). Recently, Perlin and Neyret (2001) made the observation that while noise is a very useful primitive for creating texture detail, it does not work well for describing flowing detail. It lacks “swirling” and advection behavior. To overcome this, they presented a few simple extensions to Perlin (1985) Noise that make the noise appear to flow more realistically (Perlin and Neyret, 2001). Very recently, Neyret (2003) has also presented a method for overcoming problems of basic advection of textures to add detail to flows. An interesting avenue of research would be to combine techniques for advecting procedural noise with physical cloud simulation.

d) Current model assumes that clouds exist alone. In order to represent affect of terrain (such as tall mountains) on the clouds, arbitrary bounding conditions would need to be evaluated. Such boundaries can be implemented as described in (Griebel *et al*., 1998).

e) General-purpose computation on GPUs is an interesting area of research. GPUs will see increasing use in computer games for procedural texturing and physically-based simulation. Also, the low cost, high speed, and parallelism of GPUs makes them ideal in many ways for scientific computing. Imagine giant clusters of PCs with powerful GPUs crunching through massive physical and numerical simulations.

REFERENCES

Bhate, N. and Tokuta, A. (1992). Photorealistic Volume Rendering of Media with Directional Scattering. *In Proceedings of the 3rd Eurographics Workshop on Rendering*. 1992.

Bier, E. A., and Kenneth, R. S. Jr. (1986). Two-Part Texture Mappings. *IEEE Computer Graphics and Applications Vol. 6, No. 9, September 1986:40-53.*

Blasi, P., Le, S. B. and Schlick, C. (1993). A Rendering Algorithm for Discrete Volume Density Objects. *In Proceedings of Eurographics* 1993:201–210.

Blinn, J. F. (1982). Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Proceedings of SIGGRAPH '82. In Computer Graphics*. Volume 16, No. 3. July 1982:21-29.

Blinn, J. F. (1982a). A Generalization of Algebraic Surface Drawing. *In Proceedings of SIGGRAPH*. 1982:273–274.

Blinn, J. F. (1982b). Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *In Proceedings of SIGGRAPH 1982, Computer Graphics*. 1982:21–29.

Blinn, J. F. (1978). Simulation of Wrinkled Surface. *Proceedings of SIGGRAPH'78 (Atlanta, Goergia, August 23-25, 1978). In Computer Graphics*. August 1978:286-292.

Blinn, J. F., Newell and Martin (1976). Texture and Reflection in Computer Generated Images. *Communications of the ACM* 19, 1976:542-547.

Blythe, D. (1999). Advanced Graphics Programming Techniques Using OpenGL. Course Note #29. *SIGGRAPH*.

Burt, P. J., Adelson and Edward, H. (1983).A Multiresolution Spline With Application to Image Mosaics. *ACM Transactions on Graphics Vol. 2, No. 4, October*. 1983:217-236.

Cabral B., Olano, M. and Nemec, P. (1999). Reflection Space Image Based Rendering. *Proc. of SIGGRAPH*. 1999:165-170.

Cabral, B., Cam, N., and Foran, J. (1994). Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. *In Proceedings of the 1994 Symposium on Volume Visualization*. 1994:91–98.

Christian, J (2000). *The Representation of Cloud Cover in Atmospheric General Circulation Models*. Ludwig-Maximilians-University:Munchen: Ph.D. Thesis.

Cook, R. (1984). Shade Trees. *Proceedings of SIGGRAPH'84 (Minneapolis, Minnesota, July 23-27, 1984. In Computer Graphics.* Vol. 18, No. 3, July 1984:223-231.

Demko, S., Hodges, L. and Naylor, B. (1985). Construction of Fractal Objects with Iterated Function Systems. *Proceedings of SIGGRAPH'85, San Francisc, California, July 22-26, 1985. In Computer Graphics*. Vol. 19, No. 3, July 1985:271-278.

Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T. and Nishita, T. (2000). A Simple, Efficient Method for Realistic Animation of Clouds. *In Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. ACM Press*. 2000: 19-28.

Dobashi, Y., Nishita, T., Yamashita, H. and Okita, T. (1999). Using Metaballs to Modeling and Animate Clouds from Satellite Images. *The Visual Computer*. Vol. 15, No. 9: 1999: 471-482.

Dobashi, Y., Nishita, T. and Okita, T. (1998). Animation of Clouds Using Cellular Automaton. *Proceedings of Computer Graphics and Imaging*. 1998:251-256

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K. and Worley S. (2002). *Texturing & Modeling: A Procedural Approach*. Third Edition. Morgan Kaufman.

Ebert, D. S., Musgrave, F.K., Peachey, P., Perlin, K. and Worley S. (1998). *Texturing and Modeling: A Procedural Approach*. Academic Press, Cambridge. 1998. ISBN 0-12-228730-4.

Ebert, D. S. (1997). Volumetric Modeling with Implicit Functions: a Cloud is Born. *Visual Proceedings of SIGGRAPH*. 1997. Los Angeles, California 147.

Ebert, D. S. (1996). *Solid Spaces: A Unified Approach to Describing object Attributes*. The Ohio State University: PhD thesis.

Ebert, D. S. and Parent, R.E. (1990). Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-Buffer Techniques. *Computer Graphics Vol. 24,No. 4*. August 1990:357–366.

Elinas, P. and Sturzlinger, W. (2001). Real-Time Rendering of 3D Clouds. *The Journal of Graphics Tools*. Vol. 5, No. 4, 2001:33–45.

FlightGear (2003). *FlightGear Flight Simulator*. http://www.flightgear.org/.

Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1990). *Computer Graphics. Principles and Practice*. Second edition. Addison

Forsey, D. and Bartels, R. (1988). Hierarchical B-Spline Refinement. *Proceedings of SIGGRAPH'88. Atlanta, Georgia, August 1-5, 1988. In Comuter Graphics*. Vol. 22, No. 3, August 1988:205-202.

Fournier, A. and Reeves, W. (1986). A Simple Model for Ocean Waves. *Proceedings of SIGGRAPH'86, Dallas, Texas, August 18-22, 1986. In Computer Graphics*. Vol. 20, No. 4, August 1986:75-84.

Fournier, A., Fussel, D. and Carpenter, L. (1982). Computer Rendering of Stochastic Models. *Communications of the ACM*. Vol. 25, Issue 6. June 1982:371-384.

Gardener, G.Y. (1985). Visual Simulation of Clouds. *ACM SIGGRAPH Computer Graphics.* Vol.19 No.3. July 1985: 297-303.

Griebel, M., Dornseifer, T., and Neunhoeffer, T. (1998). Numerical Simulation in Fluid Dynamics: A Practical Introduction. SIAM Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics, Philadelphia.

Gunadi, W. N. (2003). *Modified Sweep Algorithm with Fuzzy-Based Parameters for Public Bus Route Selection*. Universiti Teknologi Malaysia: Ph.D. Thesis.

Gustav Taxén (1999). *Cloud Modeling for Computer Graphics*. Royal Institute of Technology, Stockholm, Sweden: Master Thesis.

Haltiner, G. J. (1959). On the Theory of Convective Currents. *Tellus*. Vol. 11.

Harris, M. J. (2003). *Real-Time Cloud Simulation and Rendering*. University of North Carolina, Chapel Hill: Ph.D. Thesis.

Harris, M. J., Willium, V. B. III, Thorsten, S. and Anselmo, L. (2003). Simulation of Cloud Dynamics on Graphics Hardware. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*. 2003.

Harris, M. J. (2002). Real-Time Cloud Rendering for Games. *Proceedings of Game Developers Conference 2002*. March 2002.

Harris, M. J., Coombe, G., Scheuermann, T. and Lastra, A. (2002). Physically-based Visual Simulation on Graphics Hardware. *In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. 2002:109–118.

Harris, M. J. and Lastra, A. (2001). Real-Time Cloud Rendering. *EUROGRAPHICS*. 2001, Volume 20, Number 3, 2001:76-84.

Heckbert and Paul, S. (1986). Survey of Texture Mapping. *IEEE Computer Graphics and Applications* 6, 11 November 986:56-67.

Heidrich, W. and Seidel, H. P. (1999). Realistic, Hardware-Accelerated Shading and Lighting. *Proc. of SIGGRAPH'99*. 1999:171-178.

Heinzlreiter, P., Kurka, G., and Volkert, J. (2002). Real-time Visualization of Clouds. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2002.

Howard, L. (1804). *On the Modifications of Clouds*. J. London:Taylor.

Jensen, H. W. and Christensen P. H. (1998). Ecient Simulation of Light Transport in Scenes with Participating Media Using Photon Maps. *In Proceedings of SIGGRAPH*. 1998:311–320.

Jensen, H. W. (1996). Global Illumination Using Photon Maps. *In Proceedings of the Eurographics Workshop on Rendering*. 1996:21–30.

Kajiya, J. T. and Herzen, B. P. V. (1984). Ray Tracing Volume Densities. *Computer Graphics (Proceedings of SIGGRAPH). ACM Press*. Vol. 18, No. 3. 1984:165–174.

Kass, M. and Miller, G. (1990). Rapid, Stable Fluid Dynamics for Computer Graphics. *Proceedings of SIGGRAPH'90, Dallas, Texas, August 6-10, 1990. In Computer Graphics*. Vol. 24, No. 4, August 1990:49-58.

Klassen, R. V. (1987). Modeling the Effect of the Atmosphere on Light. *ACM Transaction on Graphics*. Vol. 6, No. 3, July 1987:215-237.

Kniss, J., Premoze, S., Hansen, C., and Ebert, D. S. (2002). Interactive Translucent Volume Rendering and Procedural Modeling. *In Proceedings of IEEE Visualization*. 2002: 109–116.

Levoy, M. (1988). Display of Surfaces From Volume Data. *IEEE Computer Graphics & Applications*. Vol. 8, No. 3, 1988:29–37.

Lewis, J. P. (1989). Algorithms for Solid Noise Synthesis. *In Proceedings of SIGGRAPH*. 1989: 263–270.

Magnenat-Thalmann, N. and Thalmann, D. (1985). Computer Animation Theory and Practice. *Springer Verlag*. Tokyo, Japan, 1985.

Mandelbrot, B. (1982). *The Fractal Geometry of Nature*. San Francisco, California: Freeman.

Max, N. (1994). Efficient Light Propagation for Multiple Anisotropic Volume Scattering. *In Proceedings of the 5th Eurographics Workshop on Rendering*. June 1994:87-104.

Max, N. (1986). Light Diffusion Through Clouds and Haze. *Computer Vision, Graphics, and Image Processing*. Vol. 33, 1986:280-292.

Max, N. (1981).Vectorized Procedural Models for Natural Terrain: Waves and Islands in the Sunset. *Proceedings of SIGGRAPH'81. In Computer Graphics*. Vol. 15, No. 3, August 1981:317-324.

Meuller, K., Shareef, N., Huang, J. and Crawfis, R. (1999). Hight-Quality Splatting on Rectilinear Grids with Efficient Culling of Occluded Voxels. *IEEE Trans. on Visualization and Computer Graphics*. Vol. 5, No. 2, 1999:116-134.

Mikhail, O. (1997). *An Investigation of Ice Production Mechanism Using a 3-D Cloud Model With Explicit Microphysics*. University of Oklahoma: Ph.D. Thesis.

Miller, G. (1986). The Definition and Rendering of Terrain Maps. *Proceedings of SIGGRAPH'86. In Computer Graphics*. Vol. 20, No.4, August 1986:39-48.

Miyazaki, R., Yoshida, S., Dobashi, Y. and Nishita, T. (2001). A Method for Modeling Clouds Based on Atmospheric Fluid Dynamics. *Proceedings of Pacific Graphics 2001. IEEE Computer Society Press.* 2001: 363-372.

Muhammad Azam Rana, Mohd Shahrizal Sunar, Mohd Norikhwan Nor Hayat, Sarudin Kari and Abdullah Bade (2004). Framework for Real Time Cloud Rendering. *International conference on Computer Graphics, Imaging and Visualization (CGIV04). In Proceeding of IEEE Computer Society Press*. Penang, Malaysia. 26–29 July, 2004.

Muhammad Azam Rana, Mohd Shahrizal Sunar, Sarudin Kari, Siti Mariyam Shamsuddin (2003). A Survey of Cloud Modeling Techniques. *GMAG03*. London, july 16-18, 2003.

Musgrave, F. K. (1990). A Note on Ray Tracing Mirages. *IEEE Computer Graphics Applications*. 1990:10–12

Musgrave, F. K., Kolb, C. E. and Mace, R. S. (1989).The Synthesis and Rendering of Eroded Fractal Terrains. *Proceedings of SIGGRAPH'89. In Computer Graphics*. Vol. 23, No. 3, July 1989:41-50

Nagel, K. and Raschke, E. (1992). Self-Organizing Criticality in Cloud Formation? *Physica A*. 182:519–531.

Neyret, F. (2003). Advected Textures. *Eurographics/SIGGRAPH Symposium on Computer Animation*.

Neyret, F. (1997). Qualitative Simulation of Convective Cloud Formation and Evolution. International Workshop on Computer Animation and Simulation. Eurographics.

Nishita, T. and Dobashi, Y. (2001). Modeling and Rendering of Various Natural Phenomena Consisting of Particles. *Proceedings of Computer Graphics International*. 2001: 149-156

Nishita, T., Dobashi, Y. and Nakamae, E. (1996). Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light. *Proceedings of SIGGRAPH*. New Orleans, Los Angelus. 1996: 379–386.

Nishita, T., Miyawaki, Y. and Nakamae, E. (1987).A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources. *Proceedings of SIGGRAPH'87. In Computer Graphics*. Vol. 21, No. 4, July 1987:303-310.

Ofek, E. and Rappoport, A. (1998). Interactive Reflections on Curved Objects. *Proc. of SIGGRAPH'98*. 1998:333-342.

Overby, D., Melek, Z. and Keyser, J. (2002). Interactive Physically-Based Cloud Simulation. *Proceedings of Pacific Graphics*. 2002: 469-470.

Patmore, C. (1993). Simulated Multiple Scattering for Cloud Rendering. *In Graphics, Design and Visualization, Proceedings of the IFIP TC5/WG5.2/WG5.10 CSI International Conference on Computer Graphics*. 1993:29–40.

Peachey, D. (1986). Modeling Waves and Surf. *Proceedings of SIGGRAPH'86 Dallas, Texas, August 18-22, 1986. In Computer Graphics*. Vol. 20, No. 4, August 1986:65-74.

Peachey and Darwyn (1985). Solid Texturing of Complex Surfaces. *Proceedings of SIGGRAPH'85*, San Francisco, California, July 22-26, 1985:279-286.

Perlin, K. and Neyret, F. (2001). Flow Noise. *In SIGGRAPH 2001 Technical Sketches and Applications*.

Perlin, K. and Hoffert, E. (1989). Hypertexture. *Proceedings of SIGGRAPH'89, Boston, Massachusetts, July 31- August 4, 1989. In Computer Graphics*. Vol. 23, No. 3, July 1989:253-262.

Perlin, K. (1985). An Image Synthesizer. *Proceedings of SIGGRAPH'85, San Francisco, California, July 22-26, 1985. In Computer Graphics.* Vol. 19, No. 3, July 1985:287-296.

Prusinkiewicz, P., Lindemeyer, A. and Hanan, J. (1988). Developmental Models of Herbaceous Plants for Computer Imagery Purposes. *Proceedings of SIGGRAPH'88, Atlanta, georgia, August 1-5, 1988. In Computer Graphics*. Vol. 22, No. 3, August 1988:141-190.

Reeves, W. and Blau, R. (1985). Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Sytems. *Proceedings of SIGGRAPH'85, San Francisco, California, July 22-26, 1985. In Computer Graphics*. Vol. 19, No. 3, July 1985:313-322.

Reeves, W. (1983). Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *Computer Graphics*. Vol. 17, No. 3. July 1983:359-375.

Reynolds, C., Flocks, H. and Schools (1987). A Distributed Behavioral Model. *Proceedings of SIGGRAPH'97, Anaheim, California, July 27-31. In Computer Graphics*. Vol. 21, No. 4, July 1987:25-34.

Rogers, R. R. and David, F. (1985). *Procedural Elements for Computer Graphics*. New York: McGraw-Hill Book Company.

Rushmeier, H. E. and Torrance, K. E. (1987). The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium. *In Proceedings of SIGGRAPH*. 1987:293–302.

Sakas, G. (1993) Modeling and Animating Turbulent Gaseous Phenomena Using Spectral Synthesis. *Visual Computer*. Vol. 9. 1993:200–212.

Samek, M., Slean, C. and Weghorst, H. (1986). Texture Mapping and Distortion in Digital Graphics. *The Visual Computer*. Vol. 2, No. 9. 1986:313-320.

Schpok, J., Simons, J., Ebert, D. S. and Hansen C. (2003). A Real-Time Cloud Modeling, Rendering, and Animation System. *Eurographics/SIGGRAPH Symposium on Computer Animation. The Eurographics Association*. 2003:160–166.

Segal, M. and Akeley, K. (2001). The OpenGL Graphics System: A Specification (Version 1.3). http://www.opengl.org.

Sims, K. (1990). Particle Animation and Rendering Using Data Parallel Computation. *Proceedings of SIGGRAPH'90, Dallas, Texas, August 6-10, 1990. In Computer Graphics*. Vol. 24, No. 4, August 1990:405-413.

Srivastava, R. C. (1967). A Study of the Effect of Precipitation on Cumulus Dynamics. *Journal of the Atmospheric Sciences*. Vol. 24. 1967:36-45.

Stam, J. (1999). Stable Fluids. *Computer Graphics (Proceedings of SIGGRAPH 1999). ACM Press*. 1999: 121-128.

Stam, J. (1995). Multiple Scattering as a Diffusion Process. *In Proceedings of the 6th Eurographics Workshop on Rendering*. 1995:41–50.

Stam, J. and Fiume, E. (1995a). Dipicting Fire and Other Gaseous Phenomena Using Diffusion Processes, Proc. of SIGGRAPH'95, 1995, 129-136.

Stam, J. and Fiume, E. (1993). Turbulent Wind Fields for Gaseous Phenomena. *Proceedings of SIGGRAPH*. Anaheim, California. 1993:369–376.

Stam, J. and Fiume, E. (1991). A Multiple-Scale Stochastic Modeling Primitive. *Proceedings of Graphics Interface*. Calgary, Alberta. 1991: 24–31.

Steiner, J. T. (1973). A Three-Dimensional Model of Cumulus Cloud Development. *Journal of the Atmospheric Sciences*. Vol. 30. 1973:414-435.

Takeda, T. (1971). Numerical Simulation of a Precipitating Convective Cloud: the Formation of a “Long-Lasting” Cloud. *Journal of Atmospheric Science*. Vol. 28, 1971:350–376.

Trembilski, A. (2002). Surface-Based Efficient Cloud Visualisation For Animation Applications. Proceedings WSCG 2002.

T'so P. and Barsky, B. (1987). Modeling and Rendering Waves: Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping. *ACM Transaction on Graphics*. Vol. 6, No. 3, July 1987:191-214.

Vince, J. (1995). *VirtualRreality Systems*. Singapore:Addison-Wesley.

Voss, R. F. (1983). Fourier Synthesis of Gaussian Fractals: 1/f Noises, Landscapes, and Flakes. *In SIGGRAPH: Tutorial on State of the Art Image Synthesis. ACM, SIGGRAPH*. Vol. 10. 1983.

Westover, L. (1990). Footprint Evaluation for Volume Rendering. *SIGGRAPH*. 1990:367-376.

Willis, P. J. (1987). Visual Simulation of Atmospheric Haze. *Computer Graphics Forum*. Vol. 6, 1987:35-42.

Wilson, O., Van Gelder, A. and Wilhelms, J. (1994). *Direct volume rendering via 3D textures*. Technical Report. UCSC-CRL-94-19. University of California at Santa Cruz.

APPENDIX A

CLOUD MACROSTRUCTURE EDITOR

In order to design cloud shapes easily, an interactive cloud editor – **cloud macrostructure editor** has been developed. **GLUI controls** have been used to develop intuitive and easy to use user interface for this editor. The main window is used for visually placing cubes forming the shape of clouds. The other two accompanying sub-windows hold the various function controls for performing various task related to create cloud shapes.

Followings are the features of cloud editor:

1. GLUI controls have been used to develop this editor.
2. The main window is used for visually placing cubes forming the shape of clouds.
3. The other two accompanying sub-windows hold the various function controls for performing various task related to create cloud shapes.
4. The user interface is very intuitive and easy to use.
5. Any novice user can easily use it to define high-level appearance of clouds.

The controls have been grouped into four general groups;

1. Viewing parameters.
2. Global rendering parameters.
3. Cloud media shape controls.
4. File manipulation controls.

Figure A1 shows the snapshot of cloud editor.

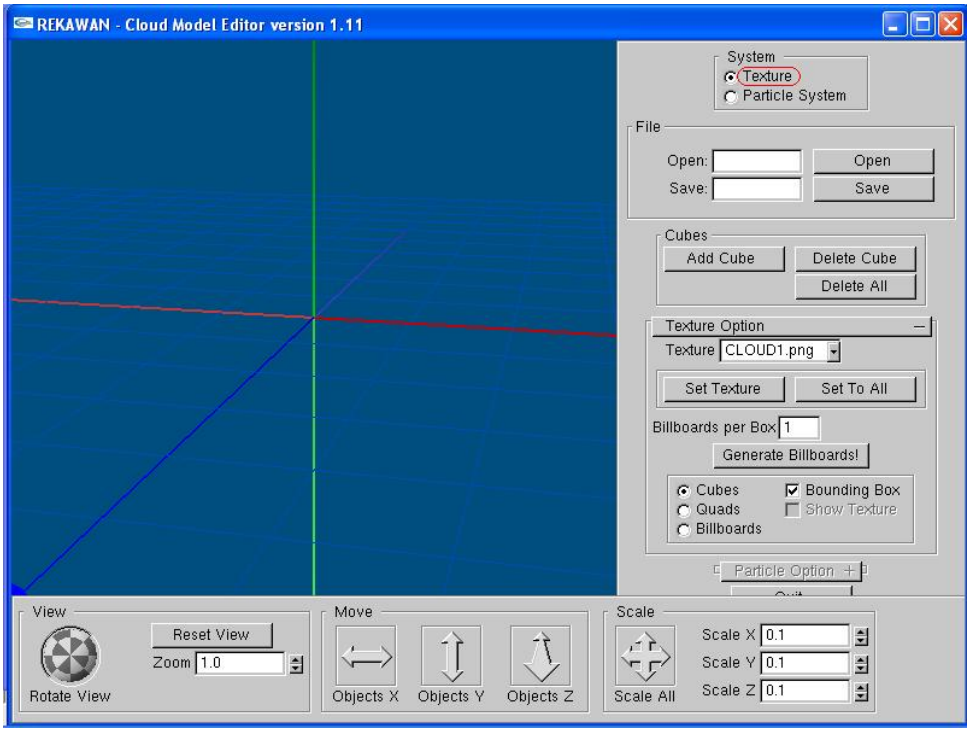

Figure A1: Snapshot of Cloud Editor

As shown in Figure A1, when the modeling system selected in cloud editor is **Texture**, then the following groups of control are provided to work with clouds:

- Files.
- Cubes.
- Texture Option.
- Scale.
- Move.
- View

When the option selected is **Particle System** then the following groups of control are provided to work with clouds (see Figure A2):

- Files.
- Cubes.
- Particle Option.
- Scale.

- Move.
- View

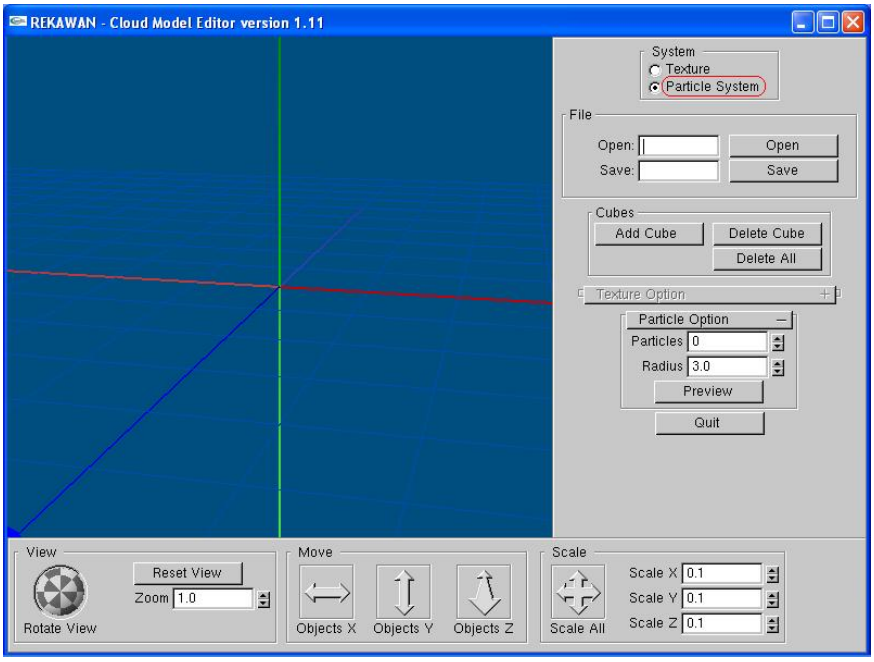

Figure A2: Snapshot of Cloud Editor using Particle System

To design cloud shapes, the designer can add cubes by pressing '**Add Cube**' button from the control group named as '**Cubes**'. He can also delete any specific cube or all cubes at once by using '**Delete'** or '**Delete All**' buttons.

Figure A3 shows the snapshot of cloud editor showing cubes. The cube in red color is active and its attributes such as size and location can be changed by using various controls.

After adding cubes to model cloud shapes, selecting textures with each cube, selecting number billboards for each cube, and etc, the designer can preview the clouds in the same window.

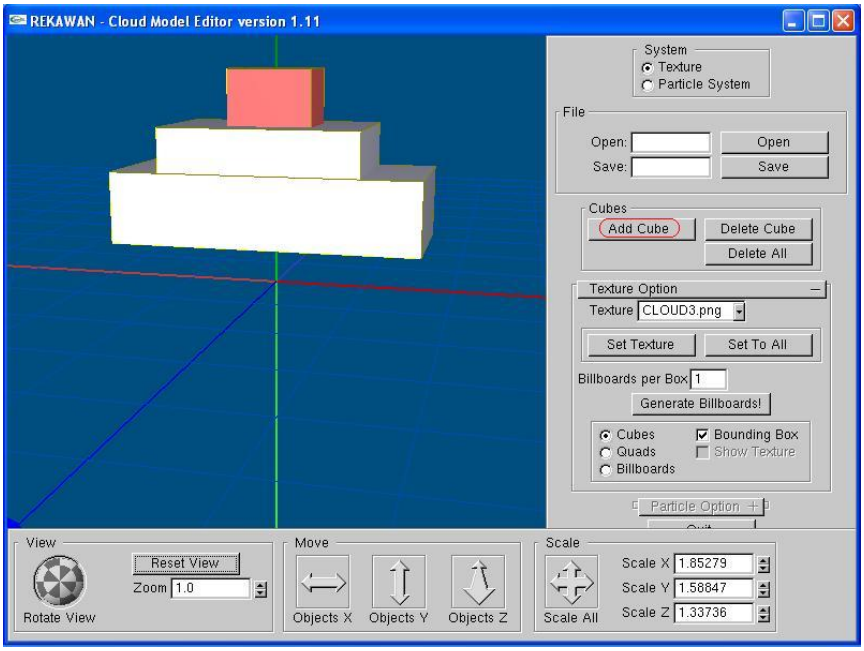

Figure A3: Cloud Editor in action – Adding cubes to shape clouds

In the following, Figure A4 shows the snapshot of cloud editor showing textured clouds by using three cubes as shown in Figure3 above.

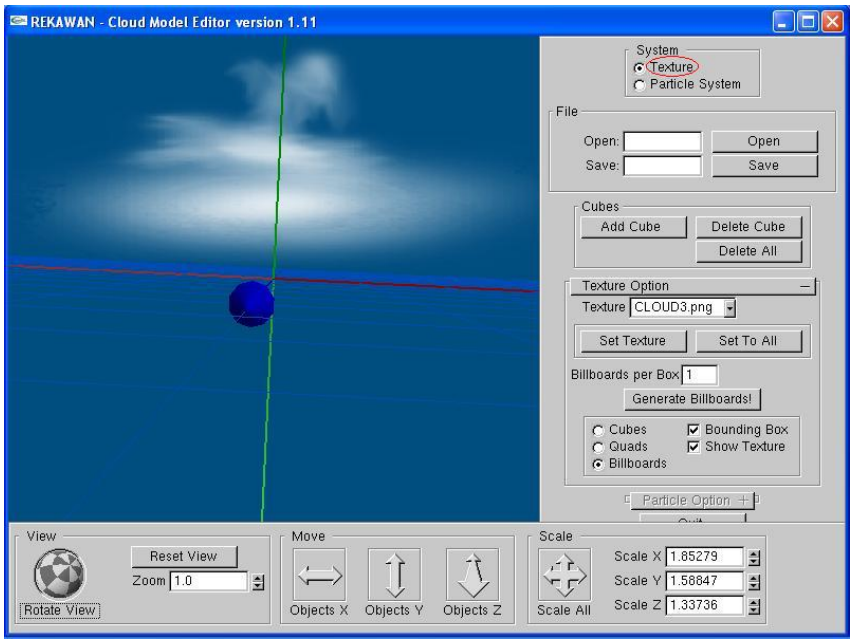

Figure A4: Cloud Editor showing textured clouds

In order to model clouds for particles system, Figure A5 and Figure A6 show arrangement of various cubes having varying sizes and randomly placed at different locations in 3-D space.

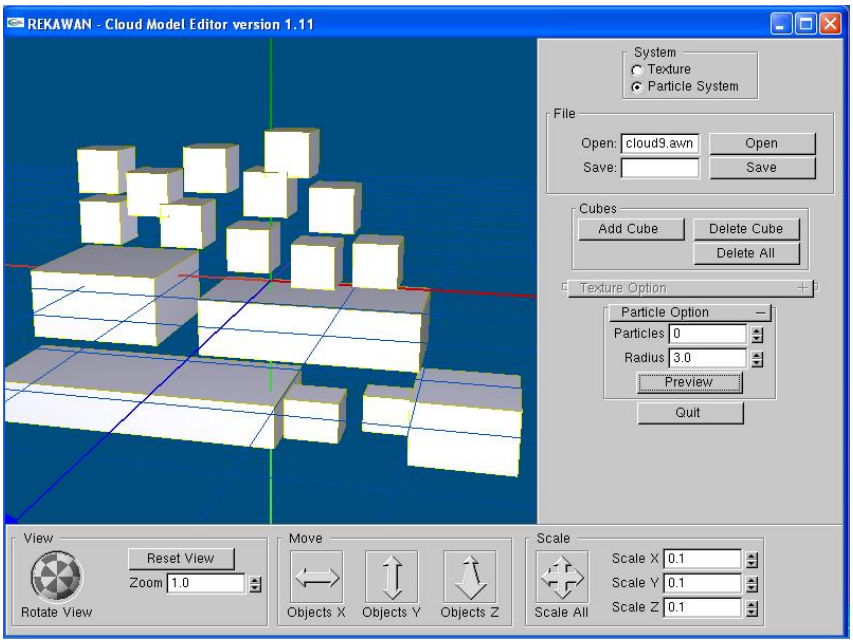

Figure A5: Designing cloud shape for particles system

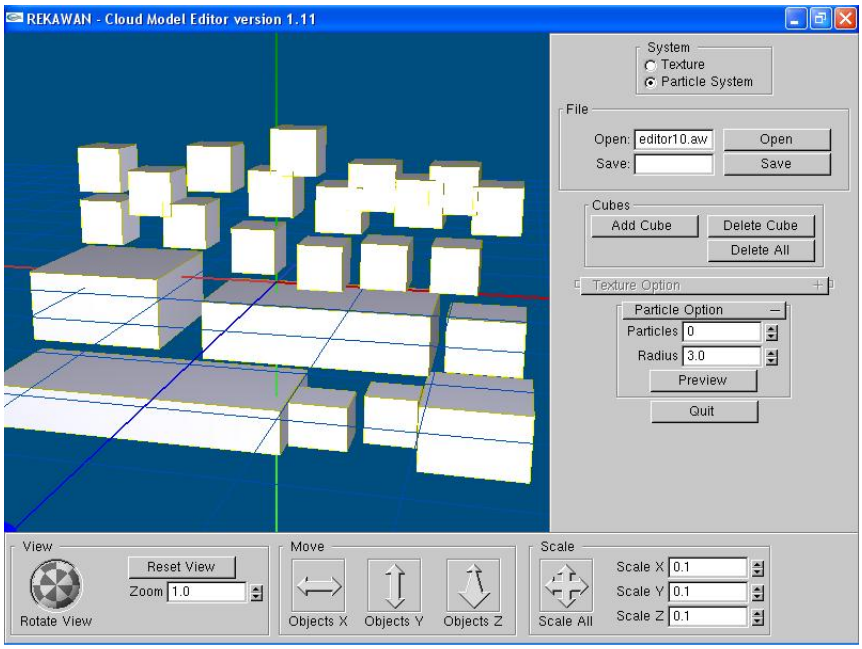

Figure A6: Designing cloud shape for particles system

Figure A7 and Figure A8 below show clouds modeled according to shapes designed by cloud editor as listed in Figure A5 and Figure A6, using particles system. The designer can add any number of cubes to model the exact shape of clouds he wants to create.

Figure A7: Cloud modeled with 554 particles

Figure A8: Cloud modeled with 615 particles