Knowledge-based 4D Visualization of Amorphous Phenomena in Complex Terrain

Dissertation

zur

Erlangung der naturwissenschaftlichen Doktorwürde
(Dr. sc. nat.)
vorgelegt der
Mathematisch-naturwissenschaftlichen Fakultät
der
Universität Zürich
von

Yi Wu

Promotionskomitee
Prof. Dr. Klaus I. Itten
Prof. Dr. Renato Pajarola
Prof. Dr. Daniel Nüesch
Dr. Britta Allgöwer

Zürich 2007

Abstract

This thesis aims to visualize natural amorphous phenomena, such as clouds, smoke, and flames photorealistically. Moreover, based on scientific simulation results, this thesis also aims to visualize large-scale amorphous phenomena in real-time. Therefore, methods for photorealistic and fast visualization are developed in this thesis.

This thesis introduces a fast and efficient method for the visually realistic display of natural volumetric amorphous phenomena. Taking multiple light sources and light multiple scattering among internal particles of gaseous phenomena into account, this method can display visually convincing natural amorphous phenomena with self-shading properties in various colors, no matter whether they are illuminated by spot or direction light sources. In addition, this method only requires small temporary saving space and can accomplish rendering processes much faster than the hardware acceleration method without losing visual quality. Moreover, with an adjustable scaling factor, it can further accelerate rendering processes efficiently or improve illumination effects according to user's requirements.

This thesis also presents a new approach for the spatiotemporal visualization of scientifically simulated migratory insect dynamics and resultant vegetation changes in real-time. The visualization is capable of displaying simulated ecological phenomena in an intuitive manner, which allows ecological research results to be easily understood by a wide range of users. In order to design a fast and efficient visualization technique, a simplified mathematical model is applied to an intuitive representation of migrating insect groups. In addition, the technique of dynamic impostors is used to accelerate rendering processes. The presented visualization method is implemented in an integrated spatiotemporal system, which models, simulates and analyzes ecological phenomena such as the insect migration through time at a variety of spatial resolutions.

A simplified Large Eddy Simulation approach is introduced to efficiently simulate the 2D turbulent motion of the incompressible viscous flow. This method conserves the small scale swirling and rolling characteristics of the flow which are normally sheared by the simulation on a coarse grid. A simple procedure allows the use of spectral methods to compute the velocity field of the flow in the Fourier domain. The numerical solver for the Navier-Stokes equations is therefore accelerated and easy to implement. This method also allows users to interactively add other influential factors, such as a turbulent wind field to a smoke model during its simulation process.

A particle-based modeling method is presented to efficiently simulate the 3D turbulent flow in real-time. This method generates the swirling and rolling characteristics of the flow by introducing vortical particles directly into the velocity field. These particles move together with other internal particles of the flow and influence the velocity field within their functional radiuses. An additional parameter is given to every particle to simply control its individual motion and status. With this method, turbulent smoke and flames can be modeled efficiently in a visually convincing way by only changing the value of a few parameters.

A new approach is presented to spatiotemporally visualize wildfire propagation in terrain. An efficient method is applied to visualize large-scale dynamic flames and smoke in real-time. This visualization is based on scientific outputs from a fire simulation system and is also integrated to the spatiotemporal system for efficient and real-time data exchanges and parallel computation.

This thesis is supported by the Swiss National Science Foundation within the National Research Program 48 'Landscapes and Habitats of the Alps' (Grant N° 4048–064432/1).

Zusammenfassung

Ziel dieser Doktorarbeit ist die fotorealistische Visualisierung natürlicher, amorpher Phänomene wie Wolken, Rauch oder Flammen. Darüber hinaus sollen solche grossräumigen Erscheinungen, basierend auf Ergebnissen der Simulation, in Echtzeit dargestellt werden können. Zu diesem Zweck werden in der vorliegenden Arbeit Methoden für fotorealistische und schnelle Visualisierung entwickelt.

Die Arbeit gibt eine Einführung in eine schnelle und effiziente Methode für die realistische Visualisierung natürlicher, volumetrischer Phänomene in Echtzeit. Unter Berücksichtigung mehrfacher Lichtquellen und Volumenstreuung beteiligter Partikel in einem gasartigen Volumen können mit dieser Methode optisch ansprechend natürliche Erscheinungen mit automatisch generierter Schattierung in verschiedenen Farben dargestellt werden, unabhängig davon, ob es sich um eine diffuse oder gerichtete Lichtquelle handelt. Zusätzlich wird nur wenig Hauptspeicher benötigt und die Darstellung erfolgt wesentlich schneller als es mit Methoden der Hardwarebeschleunigung möglich ist. Die optische Qualität wird nicht beeinträchtigt. Mit einem Skalierungsfaktor kann die Methode darüber hinaus noch beschleunigt oder an die spezifischen Anforderungen des Nutzers angepasst werden.

Diese Arbeit präsentiert ebenso einen neuen Zugang zur raumzeitlichen Visualisierung des Einflusses von Wanderinsekten auf Veränderungen in der Vegetation in Echtzeit. Auf unkomplizierte Weise ist es möglich, simulierte ökologische Prozesse zu veranschaulichen, so dass Ergebnisse ökologischer Forschung einem breiten Publikum leicht verständlich gemacht werden können. Um eine effiziente, schnelle Visualisierung zu ermöglichen, wurde ein vereinfachtes mathematisches Modell auf eine intuitive Darstellung von Wanderinsekten-Gruppen angewendet. Zur Beschleunigung des Darstellungsprozesses wurde die Technik dynamischer Impostoren eingesetzt. Die vorgestellte Visualisierungsumgebung ist in ein integriertes Raumzeitsystem eingebettet, welches ökologische Phänomene in einer Vielzahl von räumlichen Auflösungen modelliert, simuliert und analysiert.

Ein vereinfachtes Large Eddy Simulationsverfahren (LES) wird für die effiziente Simulation von turbulenter zweidimensionaler Strömung in nicht komprimierbarer viskoser Strömung eingesetzt. Diese Methode erhält die kleinmassstäbige Wirbel- und Drehcharakteristik der Strömung, die bei einer Simulation über ein grobes Raster abgeschnitten werden. Eine einfache Prozedur erlaubt den Einsatz von spektralen Methoden zur Berechnung des Geschwindigkeitsfeldes der Strömung im Frequenzraum. Die numerischen Lösungsroutinen für die Navier-Stokes-Gleichung arbeiten somit schnell und sind einfach zu implementieren. Die Methode erlaubt es ausserdem, interaktiv weitere Einflussfaktoren zu berücksichtigen, wie z.B. die Wirkung eines turbulenten Windfeldes auf ein Rauchmodell während des Darstellungsprozesses.

Eine partikelbasierte Methode zur effizienten Simulation von turbulenter 3D-Strömung in Echtzeit wird präsentiert. Diese Methode erzeugt die Wirbel- und Drehcharakteristik der Strömung durch direkte Einführung von Vortices in das Strömungsfeld. Diese Partikel bewegen sich zusammen mit anderen internen Partikeln der Strömung und beeinflussen das Feld innerhalb ihres funktionellen Radius. Jedes Partikel erhält einen zusätzlichen Parameter zur Kontrolle seiner individuellen Bewegung und seines Status. So können z.B. turbulenter Rauch und Flammen optisch überzeugend durch die Anpassung weniger Parameter modelliert werden.

Ein neuer Ansatz zur raumzeitlich variablen Simulation von Feuerausbreitung im Gelände wird vorgestellt. Auf effiziente Weise können dynamisch grossräumige Flammen und Rauch in Echtzeit visualisiert werden. Die Darstellung basiert auf den Ergebnissen eines wissenschaftlichen Feuersimulationssystems, und ist in das raumzeitliche System integriert, um effizienten Austausch von Daten in Echtzeit und parallele Prozessierung zu ermöglichen.

Diese Arbeit wurde durch den Schweizer Nationalfonds innerhalb des Nationalen Forschungsprogramms 48 ‘Landscapes and Habitats of the Alps’ gefördert (Grant N° 4048–064432/1).

Acknowledgements

This dissertation describes work done at the Remote Sensing Laboratories in the years 2003 to 2007 under the supervision of Prof. Dr. Daniel Nüesch and Prof. Dr. Klaus I. Itten.

First of all, Prof. Dr. Daniel Nüesch deserves thanks not only for the courage and confidence he put in me, but also for his unselfish guidance after his retirement. I thank him very much for his everlasting help and concern to me. His accurate proof-reading of this manuscript is gratefully acknowledged.

Without the appreciation and effort of Dr. Urs Frei in providing me with free space and his backup, this work would never have achieved the quality it now has. I thank him for his confidence and interest in the work.

Dr. Britta Allgöwer (project leader) deserves thanks for her contagious enthusiasm and help on this work. I thank her for her guidance not only on the work but also on the interesting lifestyle in Switzerland.

Dr. Andreas Fischlin deserves thanks for his enthusiastic suggestions.

Dr. Stefan Biegger deserves thanks for providing me with useful information in the initial phases of this work. I also thank him for his discussing with me about the work.

Prof. Dr. Klaus I. Itten deserves thanks for his effort in providing me with a comfortable working environment. Without it, this work could not be done so smoothly.

Dr. Daniel Isenegger and Dr. Bronwyn Price deserve thanks for their kind cooperation in the work and providing me with useful input data for this work.

I also thank Dr. Felix Morsdorf for providing me with valuable input data.

Dr. Sandra Eckert, Sarah Hangartner, Silvia Huber, Thomas Korner, and Philippe Meuret share not only the room with me, but also many cheerful moments. Sandra Altorfer, Rita Ott, and Bruno Weber deserve thanks for their kindly help on the working environment.

RSL colleagues and the Department of Geography are all thanked for the pleasant working atmosphere.

Zurich, spring 2007

Yi Wu

Abbreviations

A	Alpha channel of color with RGBA type.
API	Application Programming Interface
B	Blue
BDS	Backward Differences Scheme
CG	Conjugate Gradient method
CDS	Central Differences Scheme
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
DNS	Direct Numerical Simulation
FD	Finite Differential method
FDS	Forward Differences Scheme
G	Green
GIS	Geographical Information System
GPU	Graphics Processing Unit
GRASS	Geographic Resources Analysis Support System
LBM	Larch Bud Moth
LBM	Lattice Boltzmann Model (only in section 7.1.1.2)
IPODLAS	Interactive, Process Oriented, Dynamic Landscape Analysis and Simulation System
LES	Large Eddy Simulation
LGA	Lattice Gas Automata
LOD	Level of Detail
R	Red
RANS	Reynolds Averaged Navier-Stokes
GUI	Graphical User Interface
VTP	Virtual Terrain Project

Symbol List

A	Coefficient matrix of pressure
c_i	Constant parameter i
C_i	Center of a spherical primitive i
D	Matrix with only the diagonal elements
$\vec{D}$	Speed of the flame front in the normal direction at an interface
E	Light intensity scattered to the view point
E_k *or* E_p	Light intensity scattered by the metaball k or p to the view point
e_{qi}	Unit velocity vector in the direction qi
$\vec{f}$	External force
$f_{qi}(x,t)$	Microscopic particle density distribution at node x, at time t
f_{qi}^{eq}	Equilibrium density distribution
g	Constant related to the gravity of a particle
I_k	Intensity on metaball k
Is_i	The i^{th} light source
$I_s(p,v)$	Scattering light from direction v to metaball p
$I_{in}(p,w)$	Intensity of incident light from direction w to metaball p
$I_{out}(p,w)$	Intensity of light in direction w after traversing metaball p
K	Boltzmann's constant
l	Direction from anywhere to the view point
L	Lower triangular matrix
$life_i$	Life time of particle i
$loss$	Ratio of thermal dissipation
m_i	Mass of the i^{th} element
m_c	Molecular mass
N	Easily invertible matrix
n_t	Number of the air's internal particles in a cell at temperature t
P	Pressure
$p(x)$	Random place inside a calculating field
p_k	Particle or metaball k
P_s	Pressure of smoke
P_f	Pressure of fuel
qi	Velocity vector in a defined direction
Q	N-A
r_j	Random float number between 0 and 1 for j^{th} vortex
$r1$	Percentage of scattering light in the direct forward direction
$r2$	Percentage of scattering light within the solid angle β
$r(\theta)$	Phase function
R	Functional radius of the spherical primitive
R_m	Residual after m^{th} iterations
R_i	The i^{th} light ray
$\vec{S}$	Fuel reaction speed
s_m	Maximum value of vortex
t	Time
T	Temperature
$\vec{U} = (u, v, w)$	Velocity vector
$\hat{U} = (\hat{u}, \hat{v}, \hat{w})$	Fourier transform of the velocity
$\vec{U}_{mean}$	Mean velocity
$\vec{U}'$	Fluctuation velocity above the mean velocity

U	Upper triangular matrix
up_i	Upward velocity of particle i
$\vec{V}_s$	Velocity of smoke in the normal direction at an interface
$\vec{V}_f$	Velocity of fuel in the normal direction at an interface
$V(i,j)$	Element in the recording matrix at row i and column j
$vorInterval$	Average number of particles which contain one vortex source
$\vec{w} = (wx, wy, wz)$	Vortex
w_i	Weight of the i^{th} element
Δt	Time interval
Δx	Edge length of the cell in equidistant grid
ρ	Density
α	Albedo
α_k or α_p	Albedo of metaball k or p
ρ_s	Density of smoke
ρ_f	Density of fuel
ε^m	Error of the iterative solution at the m^{th} step
σ	Radius of particle
σ_g	Standard deviation of Gaussian function
θ	Angle between incoming and scattered light rays
β	Solid forward scattering angle
τ	Extinction coefficient
τ_k or τ_p	Extinction coefficient of metaball k or p
υ	Kinematic viscosity
η	Constant related to the material of a flow
ξ	Thermal exchange parameter
λ	Exponential decaying parameter
φ	Wavelength of light
δ	Expanding speed of blob
Ω_{qi}	Microscopic particle collision operator at a node
$\vec{\Re}$	Composite item of the Reynolds stress tensor and unknown external force
$\vec{\psi}$	Streamfunction
$\nabla=(\partial/\partial x, \partial/\partial y, \partial/\partial z)$	Gradient operator
$\nabla^2=\partial^2/\partial x^2+\partial^2/\partial y^2+\partial^2/\partial z^2$	Laplacian operator
$\nabla\times$	Curl of a vector field
$O(\ )$	Computation complexity

Contents

Chapter 1 Introduction

1.1 Background

Modeling the landscape evolution can be compared to a journey through time and space, involving different speeds and scales as well as varying thematic depth [AFF01]. To truly understand the evolution process, we need both the 'bird' view and the 'worm' view on the investigated phenomenon through the temporal scale, namely cross-scale modeling and visualization. Inspired by the idea of the spatiotemporal 'Stommel diagram' [Sto63], the scale-cube [AFF01], shown in Fig. 1.1, demonstrates a framework of spatiotemporal cross-scale modeling. Herein, the spatial scale ranges from small to large; the temporal scale ranges from fast to slow; and the model scale ranges from theoretical to empirical.

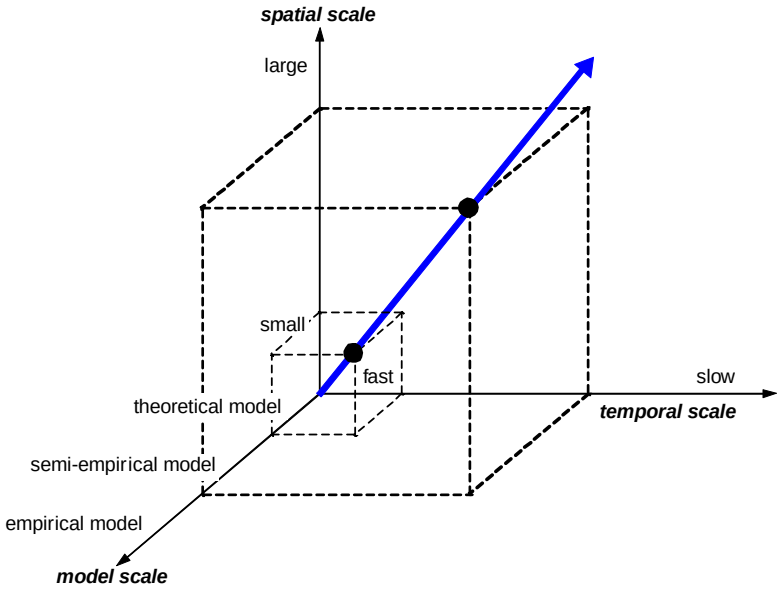


Figure 1.1: Scale-cube [AFF01].

The spatial scale concerns the spatial extent and resolution of the investigated phenomenon. The temporal scale determines the frequency of temporal changes over the investigated time period. Increasing the spatial and temporal scales can result in the increase of both the modeling extent and the modeling resolution. As we can observe from Fig. 1.1, detailed theoretical modeling with a high spatial resolution data and a short monitoring interval is located near the origin of the scale-cube. On the contrary, large-scale modeling with coarser spatiotemporal resolutions, which are often represented by models with strong empirical backgrounds, is located at the opposite corner. Depending on the available knowledge and the goal of the modeling, an evolution process can be located at any place in the scale-cube.

Accordingly, based on the disciplines of landscape visualization, ecological modeling systems, and geographical information science, an integrated spatiotemporal system, Interactive, Process Oriented, Dynamic Landscape Analysis and Simulation System (IPODLAS) is proposed to model, analyze and visualize landscape dynamics at various scales [AFF01]. IPODLAS aims to meet the needs of researchers, decision-makers, and laymen by profiting from visualization, ecological modeling, and geographical information subsystems. To reach this goal, 3 PhD theses worked independently on three

types of subsystems in parallel and worked together for their integration. This thesis is one of them and contributes to the knowledge-based 4D visualization of amorphous phenomena in complex terrain.

1.2 Motivation

Computer graphics is one of new and rising sciences. It started from the late last century after the emergence of computers. Computer graphics includes 3D modeling, animation, and rendering. Thanks to the fast development of computer hardware, nowadays normal personal computers can process data so rapidly and stably that 3D real-time visualization of large virtual terrain became possible [DWS97, RH98, PD04, CH06, and SW06]. In addition, simply utilizing a mouse to generate signals as inputs to a visualization system, the user can smoothly and freely change his view point and direction, and hence navigate inside the virtual world.

Based on the technique of real-time terrain visualization, geographic processes, ecogeographic processes, ecological processes, and other processes which can result in landscape changes are possible to be visualized in a spatiotemporal manner. Spatiotemporal visualization is 4D visualization, which displays those changes not only spatially in 3D, but also along the time axis. Using 4D visualization to display analytic results from scientific simulation systems is more understandable to decision makers and laymen than tables and 2D figures. For example, the spread of wildland fire, large-scale forest defoliation caused by migratory insects, intermittently changing land use patterns, notable conurbation and abandonment of marginal lands, do create new patterns and systems with new dynamics in space and time. There are hardly any instruments available to assess, analyze and visualize such landscape evolutions interactively with the processes triggering or forcing those changes. Therefore, there is a great need for a method that allows researchers, decision makers and laymen to understand, forecast and deal with increasing complexity. 4D visualization can act as an ideal interface of the other scientific analysis systems, through which the triggering or forcing processes can be redisplayed or displayed beforehand, accompanying with the resultant landscape changes. With the directly perceivable illustration, even a layman can utilize scientific analysis systems without understanding underlying complex models and interdisciplinary simulation technologies.

4D visualization based on the sound output of scientific simulation systems also provides the user an easily interactive interface. Through the visualization interface, modeling parameters can be easily changed by the user and then analysis can be automatically done by the connected simulation systems. The modified simulation results are then visualized again in 4D and predicted landscape changes are displayed to the user. Because of the time axis of 4D visualization, the user can not only look back but also foresee landscape changes. Hence, trainings can be done flexibly and freely through knowledge-based 4D visualization to select the best decision or solution without time and financial investments in the real world.

Knowledge-based 4D visualization of natural or man-made landscape evolution is a huge scope. This PhD thesis concentrates on the visualization of amorphous phenomena. Natural amorphous phenomena include atmospheric clouds, flames, smoke, fog, and other amorphous-like objects. These phenomena are very essential in many applications like virtual environment visualization, game, flight simulation, landscape design, and filmmaking. In addition to recreational purposes in games and movies, natural amorphous phenomena also play important roles in practice. For instance, wildland fire is one of ordinary hazards to human beings and has magnificent influences on local ecogeographical systems. Based on the knowledge of scientifically simulated fire propagation properties, 4D visualization of wildland fire in the virtual landscape can provide easily understandable information for the public and train the personnel to cope with it in a safe situation. Similarly, the appearance of virtual clouds in flight simulation can not only enhance the virtual reality but also help a pilot to deal with cloudy weathers.

Besides natural amorphous phenomena, other amorphous-like phenomena are also worth of concern. For instance, in visualization, migratory insects can be assumed as 'insect clouds' in the air because of the tremendous insect amount in each group. Since migratory insects normally cause conspicuous defoliation, namely obvious vegetation changes, visualization of migratory insect dynamics and resultant vegetation changes can provide decision makers and laymen an interface to directly observe the influence of those ecological processes and then utilize underlying simulation systems to make the most helpful decisions. Moreover, the combination of this visualization with the visualization of wildland fire propagation in terrain will allow the user to observe the fire propagation properties in places where vegetation has been changed by migratory insects. This combination can display cross-modeling results. It joints underlying temporal ecological simulation systems with spatial fire simulation systems through a 4D interface.

In general, knowledge-based 4D visualization of amorphous phenomena can display scientific analysis results in a spatiotemporal manner, and enhance virtual reality to other applications. Consequently, simulation outputs are more understandable to a wide range of users. Moreover, the application of computer graphics on the representation of real scientific problems stands also on the front fringe of the computer graphics research.

1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 discusses fundamentals of knowledge-based visualization and amorphous phenomena which this thesis aims to visualize. The physical properties of cloud, flame, and smoke are analyzed to easily describe their motions and appearances in the visualization. This chapter also discusses available data which are simulated for the knowledge-based visualization of migratory insect dynamics and wildland fire propagation respectively.

Chapter 3 describes the goal of this PhD thesis and existing challenges. Chapter 4 presents the software environment in which visualizations are implemented.

Chapter 5 presents methods which have been used in atmospheric cloud visualization and a new method developed in this thesis. Two kinds of cloud modeling methods are introduced for different types of clouds. Several cloud rendering methods are discussed based on their rendering time. To accelerate the rendering process without losing visual quality, a novel cloud rendering method is presented. This chapter also shortly introduces the technique of dynamic impostor [Sch95] which can accelerate static cloud rendering in the landscape. Time costs of different rendering methods are compared at the end of this chapter.

Real-time 4D visualization of migratory insect dynamics is presented in chapter 6. The dynamics of large bud moths in the Upper Engadine valley of the Swiss Alps is taken as a case study in this chapter. Available materials from Temporal Simulation System (TSS) and GIS are analyzed and supplementary assumptions are made to visualize temporal outputs from an ecological system in a spatiotemporal manner. In order to visualize many 'insect clouds' in real-time, based on the analysis in the chapter 5, simplified 'insect cloud' modeling and rendering methods are introduced and followed by results and discussion.

Chapter 7 concentrates on the visualization of dynamic flames and smoke. Different kinds of modeling methods are introduced and compared in this chapter. The turbulent behavior of flames and smoke is most concerned in all kinds of modeling methods. Elements in the physically based modeling are presented in detail to understand intrinsic movements of unsteady flows. Efficient rendering methods of flames and smoke are illustrated respectively at the end.

Chapter 8 presents the method to visualize large-scale wildland fire propagation. The visualization is based on simulation outputs of a fire simulation system. Methods of detailed flames and smoke visualization are simplified to allow the real-time 4D visualization of large-scale fire propagation in terrain.

Results obtained in this thesis are illustrated in chapter 9. Corresponding discussions and future works are presented in chapter 10. Chapter 11 draws conclusions on this thesis at the end.

Pseudo codes of the visualization methods developed in this thesis are demonstrated in the appendix for easier understanding and further extension.

Chapter 2 Fundamentals

This chapter presents fundamentals and meanings of knowledge-based visualization, as well as some related physical properties of clouds, flames, and smoke. Modeling and rendering methods developed in this thesis for the visualization of natural amorphous phenomena are derived from their physical properties in order to display them as visually realistically as possible. Therefore, serving as the fundamental of the whole PhD thesis, this chapter concentrates on the introduction of natural amorphous phenomena.

2.1 Fundamentals of Knowledge-based Visualization

Knowledge is information, which is data that have been given meanings through interpretations by way of relational connections and pragmatic contexts, and have been integrated into existing human knowledge structures [TK05]. Knowledge is dynamic and adaptable in coping with tasks. Accompanying with the fast development of computer hardware and visualization techniques, nowadays, the knowledge-based visualization plays an important role as a method or a tool. "One central reason is that visualizations capitalize on several characteristic features of the human cognitive processing system. The power of visualization comes from the fact that it is possible to have a far more complex concept structure represented externally in a visual display than can be held in visual and verbal working memories" [TK05]. In other words, visualization is a powerful cognitive tool for users and provides them an easy platform to understand existing complex knowledge.

Unlike visually realistic visualization for games and movies, the knowledge-based visualization views knowledge as a key resource and visualization is a supplementary tool or an interface to express it in a directly perceivable way. Therefore, the knowledge-based visualization is normally embedded in an integrated system. Herein, knowledge from other information subsystems or documents is visualized as the output and can be directly perceived by the user.

In some cases, limited to incomplete data for the realistic visualization of existing knowledge or capabilities of computer hardware, assumptions are introduced to visualize existing knowledge in an abstract way. Accordingly, knowledge-based visualization can be divided into two categories: one is the realistic visualization; the other is the abstract visualization. The realistic knowledge-based visualization, such as volume reconstruction from scanned data slices for medical usages, focuses on visualization precision. On the contrary, the abstract knowledge-based visualization aims to display intricate or incomplete information in an intuitive and heuristic way. Since the abstract visualization can not provide a realistic illustration for the user, it may encounter suspicions such as whether it brings the user misunderstandings when he believes that what he observes from the visualization is real [TK05]. Therefore, proper assumptions and explanations are required for the abstract visualization.

If we view knowledge as an "object", its properties should be analyzed carefully for proper visualization. In other words, knowledge should be broken down and categorized according to its respective properties to determine visualization methods [TK05]. Supplementary knowledge may be added to complete the database for its visualization. Therefore, the knowledge-based visualization is established upon a repository, which contains information for visualization in all aspects [TK05].

Physically based modeling and rendering are branches of the knowledge-based visualization and play important roles in the research of computer graphics. They can be considered as the combination of physical information and computer graphical techniques. As the increase of computing capabilities, complicated and time-consuming physically based simulations can be applied to the generation of realistic animation. The main advantage of physically based visualization is the capability of interactive visualization. When unpredictable objects or forces interfere with current animated objects

and influence their dynamics or appreances, physically based modeling and rendering can simulate and display interactions without precalculations and extra assumptions. Therefore, to fullfil the goal of this PhD thesis, knowledge-based visualization of natural gaseous phenomena, physical properties of clouds, flames, and smoke are introduced in the following sections.

2.2 Properties of Clouds

Clouds are a kind of common natural phenomena which have semi-transparent amorphous appearances. They are visible accumulations of water droplets and icy crystals with various sizes, floating in the lowest part of earth atmosphere, moving with the turbulent atmosphere, and undergoing condensation and other physical processes. Clouds are formed when water vapor condenses onto macroscopic dust particles or other tiny solid particles floating in the air. Water vapor may come from many evaporable resources, such as oceans, lakes, and vegetations. When water vapor rises up and cools as it rises, adhering to tiny solid particles, it turns into water droplets and icy crystals. This is because cool air can hold less water than warm air. Therefore, excess vapor condenses into either liquid or solid form according to the temperature of its surrounding air [MYD*01, SDY02].

Because of those numerous water droplets and icy crystals, clouds absorb some part of incident light when the light traverses them and scatter some part of it to all directions. When some scattered light passes through the atmosphere and enters our eyes, cloud images are formed. Therefore, the color and visibility of clouds are determined by the color and intensity of the incident light to a large extent. Due to different sizes, shapes, and properties of clouds' internal particles, incident light is absorbed to different extents and scattered to different directions with various intensities. Therefore, clouds have self-shading properties and their appearances vary a lot when they are observed at different places. In addition, when the scattered light of a particle encounters other particles, it will be absorbed and scattered again to other directions. These multiple scattering processes make rays change directions and intensities inside clouds so frequently that it is extraordinarily difficult to calculate the light scattered to the viewer by tracing rays. Moreover, the incident light of clouds comes from many sources, such as the sunlight, the skylight, light scattered by other clouds, and light reflected by oceans and the ground. The atmosphere also influences the light to some extent. Therefore, the appearance of clouds is determined by many factors.

The motion of clouds in the air also arises from various reasons. Movements resulting from self-driven forces, such as condensation and gravity, are not so apparent as movements driven by the turbulent air flow. The air flow occurs when there is a difference between pressures. The difference of pressures forces the air to flow from high pressure to low pressure. The bigger the difference is, the faster the air flows. When the air moves with high velocities, its flow tends to form eddies. This is called turbulent flow. Since clouds are floating in the air, they are carried by the air flow. Then the turbulence of the air flow is displayed by the continuous motion of clouds' internal particles.

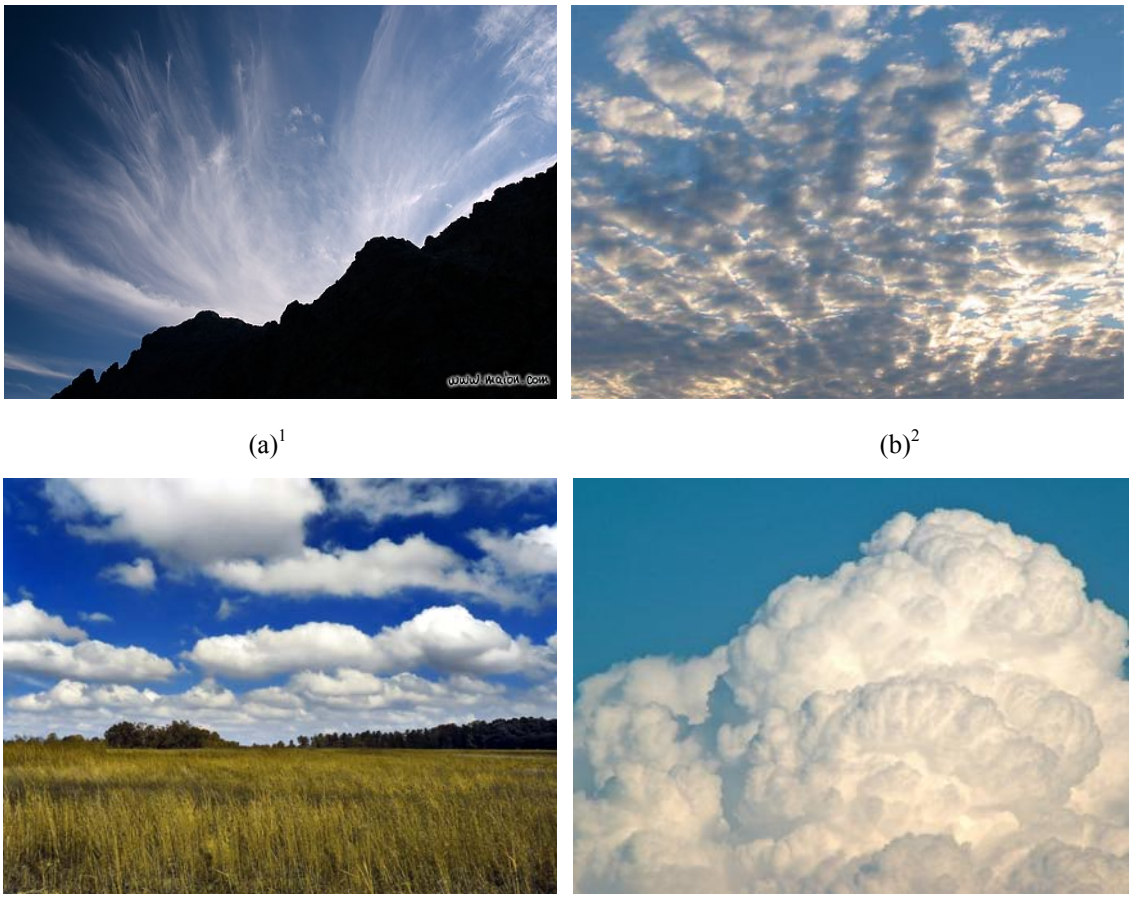
2.2.1 Atmospheric Cloud Types

Atmospheric clouds in the air have various shapes and sizes. A basic knowledge of cloud types can help us to build visually realistic cloud models and choose proper models under different environmental conditions. According to the height of clouds above the ground, clouds can be classified into the following types:

- High-level clouds. Because of the cold air temperature at the high-level (above 6'000 meters), clouds there are composed of ice crystals and appear thin and white or colorful when the sun is low on the horizon. Fig. 2.1(a) shows clouds in typically wispy cirrus appearance.
- Middle-level clouds. Clouds at this level (from 2'000 to 6'000 meters) are normally composed of water droplets and ice crystals, depending on their surrounding air temperature.

They may appear as parallel bands or rounded masses and they have cumulous properties. Fig. 2.1(b) shows clouds at the middle-level.

- Low-level clouds. Clouds at the low-level (below 2'000 meters) are normally composed of water droplets, which may turn into rain. If the air is cold enough, they may turn into ice crystals or snow. Fig. 2.1(c) shows cumulus clouds at the low-level.
- Vertically developed clouds. The most common and familiar vertically developed clouds are some kinds of cumulus clouds, which have notable heights Fig. 2.1(d) shows cumulus congestus clouds with the vertically developed property.



(a)[1] (b)[2]

(c)[3] (d)[4]

Figure 2.1: (a) High-level clouds; (b) middle-level clouds; (c) low-level clouds; (d) vertically developed clouds

The above classifications generally define the appearance of clouds at different heights from the ground. In order to easily construct cloud models, according to their appearances, we can also divide clouds into two categories: smooth fuzzy clouds and puffy cotton-ball cumulus clouds. That is, clouds which do not have puffy cotton-ball appearances do not belong to smooth fuzzy clouds, such as Fig. 2.1(a). Therefore, cloud models can be constructed by two kinds of methods. One is to distribute small particles with similar densities everywhere inside cloud models for smooth fuzzy appearances. The

[1] This Figure is downloaded from http://www.maion.com
[2] This figure is downloaded from http://www.notre-planete.info/geographie/climatologie_meteo=
[3] This figure is downloaded from http://en.wikipedia.org/wiki/Cumulus_humilis_cloud
[4] This figure is downloaded for weather.stives-town.info/clouds.htm

other is to accumulate internal particles at some places to create cumulous effects. These two methods will be discussed in Chapter 5.

2.2.2 Atmospheric Cloud Colors

The color of clouds is mainly determined by the color of their incident light. Since the incident light may come from many sources, the color of clouds is their synthetic illumination result. In the daytime, the most dominant light source is the sun. Blue and green parts of the sunlight, which have shorter wavelength than yellow and red parts, are more easily scattered by the atmosphere. When the sun is just above the horizon, before the sunlight reaches clouds, it travels the longest distance in the atmosphere and contains more yellow and red parts than blue and green parts due to scattering. Therefore, clouds at sunrise or sunset appear orange, pink, or red.

With the same reason, when the incident light is white, clouds may appear bluish because more blue light is scattered by their internal water droplets or ice crystals. Especially for clouds which are mostly composed of ice crystals, they may even appear somewhat greenish. However, the colorful effect caused by internal particles of clouds is much less obvious than the color directly given by the incident light. Therefore, till now only the color of incident light is taken into consideration for cloud rendering in the research of computer graphics.

Even with pure white incident light and without the consideration of differences in colors of scattering light, the color of clouds is not monochromic and has various degrees of grey. This is due to the self-shading property of clouds, as apparently shown in Fig. 2.1(b-d). Because internal particles of clouds absorb some part of incident light and hence reduce its intensity while it traverses clouds, places having less particles will be brighter than places having more particles under the same illumination condition. In addition, water droplets and ice crystals have strong scattering abilities; so that incident light is multiple scattered among internal particles. Taking multiple scattering processes into consideration will make clouds appear reasonably brighter [HL01]. Accordingly, the simulation of light absorption and scattering is essential to cloud rendering.

2.2.3 Migratory Insect Clouds

Migratory insects, such as Larch Bud Moths (LBM) which cause conspicuous defoliation in larch forests and planthoppers which affect the production of rice, often result in negative vegetation changes when they are in large numbers. Therefore, the knowledge of migratory insect dynamics and resultant vegetation changes is within the concern of decision makers, resource managers, and the public. As an interface, real-time 4D visualization of insect dynamics and resultant vegetation changes in terrain could make non-expert users understand these processes more easily.

Though photorealistic visualization of individual insect in migration is visually realistic, with nowadays common computers, the time cost of rendering processes will be so long that visualization is impossible to be real-time. Moreover, since the numbers of migratory insects are statistical outputs of simulation runs from ecological systems, individual insect migration time and route are hardly to be determined. A feasible solution is to assume insects migrating in groups as dynamic ‘insect clouds’. A more visually realistic visualization will enhance rendering costs other than the understanding of users.

As hinted by their name, migratory insect clouds are composed of dynamically migrating creatures. This specialty makes them different from atmospheric clouds which have relatively static shapes and ignorable inner movements in a short time. In general, comparing to atmospheric clouds, insect clouds have the following properties:

- Insect clouds are dynamic clouds. Because insects are living creatures, they do not keep static positions inside their groups. Their random movements make their groups appear dynamic.
- Though insect clouds do not have describable shapes, they have related sizes. This is because temporal simulation systems normally care and calculate the statistical number of insects when they migrate from one place to another, instead of the position of an individual during the migration process.
- Insect clouds head to places, where their favorite vegetation grows. Though temporal simulation systems normally do not record the track of an individual, they can simulate a group of insects migrating from one place with certain extent to another place. Therefore, the motion of insect clouds in the air has distinct directions, though the air flow may influence their shape and movements.
- Since an insect cloud indicates a group of migratory insects, its formation and dispersion at the beginning and end stages of their migration are related to their feeding areas. Therefore, in these two stages, the shape of insect clouds reflects the shape of their favorite vegetation area to some extent.

2.3 Wildland Fire

Wildland fire occurs over a wide range of spatial and temporal scales. Fire is an exothermic chemical reaction usually releasing heat and producing smoke. The existence of fire depends on three necessary elements: fuel, oxidizer, and heat[5]. The properties of fire spreading include moving sources, combustion spread, and interaction with objects[5]. When a combusting object is moved by outer forces, for example a burning branch falling down from a tree, its flame behavior is also influenced by its motion accordingly. Around a burning place, when the temperature of a combustible object is beyond its ignition point or a sparkle is splashed on it, the object will catch on fire and flames emerge in new places. During fire spreading, flames inevitably interact with objects. The interaction introduces many changes to the motion of flames and also influences the progression of fire spreading.

Physical conditions, such as wind, topography, fuel, and humidity, play important roles in fire spreading. Early in 1972, Rothermel described important parameters such as the rate of fire spreading in a model, which is extended in the ARC/INFO geographic information system (GIS) software to describe the spatial movement and spreading of wildland fire [Rot72]. Along with more and more deeply understanding of wildland fire properties and relations to surrounding conditions such as fuel, wind, and so on, users need a more simple but understandable interface to provide information outputted by underlying complex systems.

With this idea, some spatiotemporal visualization interfaces have been developed all over the world with various APIs to display fire spreading in 2D space. Comparing to the visualization of fire spreading in 3D space, 2D visualization with time axis can only provide information of fire spreading speed and direction. With the development of remote sensing techniques, vegetation distribution in terrain can be detected and hence fire flame length can be predicted by simulation systems [CRW03]. Hereafter, 2D space is not enough to express this depth information any more. Therefore, visualization of fire spreading in 3D space is now desirable, though it is more complicated and more time-consuming.

Flames are the visible part of wildland fire. In addition, wildland fire inevitably produces smoke, which also has turbulent dynamics. As shown in Fig. 2.2, though smoke does not have evident influence on fire spreading, it enhances reality in visualization and hence can help users to correctly recognize dangers of smoke and find right way to escape from burning and smoky places in training.

[5] From Wikipedia, http://en.wikipedia.org/wiki/Fire

Therefore, this thesis concerns both fire flame and smoke in the visualization of wildland fire. The properties of flames and smoke are introduced respectively in the following sections.

Figure 2.2[6]: Wildland fire.

2.3.1 Flames

According to the distribution of fuels and oxidizers, flames can be divided into two categories: premixed flames and diffusion flames[7]. A premixed flame looks bluish and it is generated when a fuel and its oxidizer are premixed before ignition. On the contrary, a diffusion flame normally has colors from red, orange, to white and is generated when a fuel and its oxidizer diffuse into each other at places where temperatures are higher than the fuel's ignition point. Most flames in the nature are diffuse flames, such as wildfire, fire in the oven and fireplace, torch, and so on. In general, the flame is not a matter. It is a region where exothermic chemical reactions take place with a temperature high enough to emit light[7].

The color of flames depends on several factors, such as the type of fuels involved in the combustion and spectral bands of involved components, but the most important factor is the blackbody radiation[9]. Soot particles, which are produced by incomplete combustion, are blackbodies. A blackbody is an object which absorbs all electromagnetic radiation without reflection[8]. Theoretically, it also radiates every possible wavelength of energy. Above a certain temperature, blackbodies start to radiate with visible wavelengths. Accompany with the increase of temperature, blackbody radiations make flames appear red, orange, yellow, and white before ending up at blue[9].

Because of the turbulent motion of fuels and soot in combustion, flames are dynamic and may produce dynamic smoke, which is composed of soot and other residues after exothermic chemical reactions. The movement of flames is determined by the movement of fuels and soot in the air. Because the temperature of flames is much higher than the temperature of smoke and many complicated chemical reactions existing in flames, smoke has different movements from flames in the air and can be viewed as another flow. For the visualization of dynamic flames, several related properties of flames are introduced in the following subsections. Since most natural fire has diffuse flames, premixed flames

[6] This figure is downloaded from www.wildlandfire.com
[7] From Wikipedia, http://en.wikipedia.org/wiki/Combustion
[8] From Wikipedia, http://en.wikipedia.org/wiki/Black_body
[9] From Wikipedia, http://en.wikipedia.org/wiki/Black-body_radiation

are not considered in this thesis. However, their visualization can be accomplished similarly as diffuse flames except that their radiations are most in the blue to green region of the spectrum[9].

2.3.1.1 Temperature Distribution

"Combustion or burning is a complex sequence of chemical reactions between a fuel and an oxidant accompanied by the production of heat or both heat and light in the form of either a glow or flames"[7]. Since the flame is the product of an exothermic chemical reaction, temperature in a combustion region can be used to determine the stage of combustion in each part of the region and hence to visualize flames properly at different places. To investigate the distribution of temperature in a combustion region and its contributions to the color and motion of flames, let us first look at the combustion process of diffuse flames.

The combustion process of non-gaseous fuels, such as wood and gasoline, can be divided into the following four phases[7]. In the first phase, non-gaseous fuels are heated and converted into gaseous fuels, which are more flammable. In the second phase, gaseous fuels rise in the air because of thermal buoyancy forces. In the third phase, gaseous fuels start exothermic chemical reactions with oxidizers and turn into soot, which may still contain some fuels under the condition of insufficient combustion. At the same time, driven by thermal buoyancy forces, both reacting and non-reacting gaseous fuels, soot, and other residues continuously rise in the air. In the fourth phase, other residues together with soot and perhaps also with unreacted fuels rise to places where temperatures are not high enough to induce combustion or there is lack of oxidizers. They are identified as smoke in this phase. The combustion process of gaseous fuels is quite similar, except that it does not need the first phase.

From the above analysis we can find that the temperature plays a very important role throughout the whole combustion process. In the first phase, the temperature determines the speed of nongaseous fuels transforming into gaseous fuels. In the second phase, the temperature determines when and where gaseous fuels pass their ignition points and start exothermic chemical reactions with oxidizers. In this phase, no soot is generated and the color of gaseous fuels looks bluish, like the dark blue core in the center bottom part of a candle flame shown in Fig. 2.3. Hence, the temperature also determines the size and shape of dark cores in flames. The third phase is the exothermic chemical reaction phase and soot is generated because of insufficient oxidizers in the quickly oxidizer-consuming combustion region. Due to the heat released by exothermic reactions in this phase, the blackbody radiation of soot produces red, orange, and white colors accompanying with the increase of temperature[12]. As shown in Fig. 2.3, the white region has higher temperature than the orange region. In addition, the outer layer of the flame in Fig. 2.3 appears blue. This is because there is sufficient oxygen supply. Therefore, no soot is generated here. Without blackbody radiations of soot, the color of this layer is mainly determined by molecule spectral band emission and hence is bluish. In the last phase, because of the lack of exothermic chemical reactions, the temperature turns gradually low here. Hence, soot produces very little blackbody radiations with visible wavelengths and smoke appears dark.

In addition, the motion of smoke, which is also influenced by thermal buoyancy, proves the importance of the temperature from another side. Therefore, the temperature distribution of a flame determines the appearance of the flame to a large extent and is worthy of careful computation for visually realistic flame visualization.

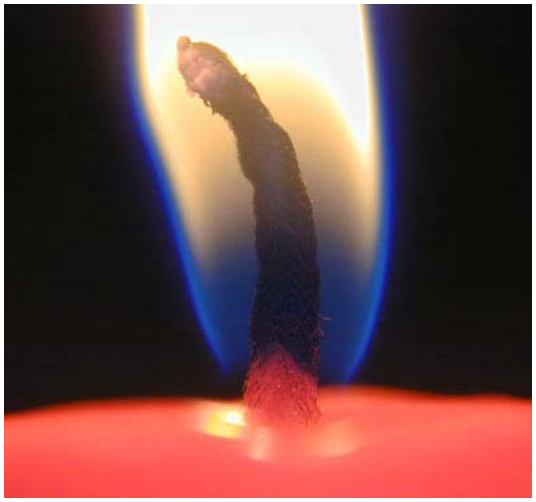


Figure 2.3[10]: A flame of a burning candle.

However, because exothermic chemical reactions continuously release unpredictable heat in the air, it is hard to track temperature changes in flames exactly. Approximately, we can assume that every gaseous fuel particle has the chance to undergo a chemical reaction with an oxidizer. Therefore, every fuel particle undergoes the temperature profile as shown in Fig. 2.4.

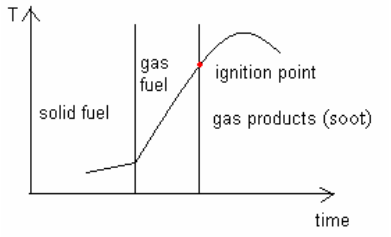


Figure 2.4: Temperature profile for a solid fuel [NFJ02].

The heat produced by exothermic chemical reactions also intends to transfer from hot places to cold places in flames. Heat transfer in flames may have all of the following three modes, conduction, convection, and radiation. Herein, conduction happens among solid particles such as fuels, soot, and other combustion products; Convection happens among gaseous fuels and gaseous products; radiation happens because of high temperatures of flames.

Heat transfer changes the temperature distribution in flames. Here, convection and diffusion are the most dominant modes of heat transfer because solid particles in flames and smoke can be thought as tiny particles carried by the air flow [NFJ02]. Therefore, heat transfer can be approximated by only taking convection and diffusion into consideration, namely:

$$T = -\vec{U} \bullet \nabla T + \upsilon \nabla^2 T \qquad (2.1)$$

where, T is the temperature, $\vec{U}$ is the velocity field of the air flow, ∇ denotes the gradient operator ($\partial/\partial x, \partial/\partial y, \partial/\partial z$), and υ is kinematic viscosity. $X = -U \bullet \nabla X$ is the general convection form of X, ∇^2 is the Laplacian operator ($\partial^2/\partial^2 x + \partial^2/\partial^2 y + \partial^2/\partial^2 z$), and $X = \upsilon \nabla^2 X$ is its general diffusion form. In general,

[10] This Figure is downloaded from **www.interactives.co.uk/candle.htm**

according to the above analysis, in order to visualize photorealistic animated flames and smoke, heat transfer and resulting temperature distribution need to be carefully simulated (see chapter 7).

2.3.1.2 Color Spectrum

When a beam of white light passes through a glass prism, we can observe the color spectrum ranging from violet at one end to red at the other. We can perceive different colors after light with different wavelengths enters our eyes and is recognized by our complicated intelligent recognition systems, brains. Light is a type of electromagnetic waves. The color of light is characterized by its wavelength (or frequency). Therefore, the color spectrum actually is the visible light spectrum, ranging from 380 nanometers to 780 nanometers defined by the Commission Internationale de l'Eclairage/International Commission on Illumination (CIE). It corresponds to the color ranging from violet, blue, green, yellow, orange, and red, as shown in Fig. 2.5. We can also observe that colors are not obviously isolated. Instead, they are smoothly blended one into the next. Light that is relatively balanced in all visible wavelengths appears white to the viewer.

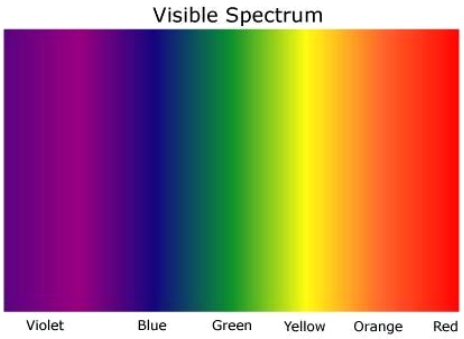


Figure 2.5: Visible color spectrum ranging from violet to red[11].

In addition to the visible spectrum, Light has invisible color spectra, ultraviolet light and infrared light. As hinted by their names, ultraviolet light is light with wavelengths shorter than the violet light; and infrared light is light with wavelengths longer than the red light[12]. They are both recognized as black color by our brains if no other visible light enters our eyes.

Nearly all objects above absolute zero degree radiate light. Light radiated by the soot determines the color of flames. Since the wavelength of light is related to the temperature, flames appear red when the temperature is not so high and turn into orange, yellow, and white as the temperature increases and more radiated rays have shorter wavelengths[9]. If temperatures in some regions of flames are high enough, flames may appear bluish in those parts. However, a more sufficient combustion produces less soot, such as the premixed flame, and makes its flame appear bluish. In this case, light is not dominantly radiated by soot. Instead, it is mainly produced by molecule band emission of the flame's internal particles.

Since the color of common diffuse flames is closely related to the temperature distribution of flames, one can calculate the temperature distribution in detail to visualize flames. Then approximations should be adopted carefully to map the temperature to red, green, and blue (RGB) before rendering flames on a screen.

[11] From http://www.gamonline.com/catalog/colortheory/visible.php
[12] From http://en.wikipedia.org/wiki/Light

2.3.1.3 Movements

As discussed in the section 2.3.1.1, the combustion process of non-gaseous fuels has four phases. The first three phases take place in the region of flames and the last phase is perceived by the viewer as dynamic smoke. The first phase happens at places which are very close to fuels and are hardly visible by naked eyes. Depending on the temperature distribution in flames, the second phase may result in a noticeable dark blue core inside a flame as shown in Fig. 2.3, or hardly noticeable areas in flames as shown in Fig. 2.6. The most remarkable phase is the third phase, which results in red, orange, white, and even blue flames under different temperature conditions, as shown in Fig. 2.3 and Fig. 2.6. Therefore, the movement of flame is mainly displayed by appearance changes of regions where the third phase takes place.

Figure 2.6[13]: Flames from burning wood.

The main components in combusting regions are air, gaseous fuels, and soot. Because of the turbulent motion of hot gaseous mixtures, most flames have not regular appearances as the one in Fig. 2.3. Instead, their appearances vary a lot in a short time, like flames in Fig. 2.6. The combustion process of common non-gaseous fuels can be classified as deflagration. Deflagration is a slow spending event, comparing to detonation. Hence, common natural flames can be modeled as incompressible flows. Soot is carried by those flows to everywhere and radiates colorful light to form turbulent appearances of flames.

An incompressible flow is a flow in which density does not change according to time. One may think that the density of soot and gaseous fuels changes dramatically in flames while fuels undergo reactions. Why can a flame be viewed as an incompressible flow? This is because the density here refers to the density of all kinds of particles in the combusting area. A small volume may lose some soot or gaseous fuels at one time, but at the same time, other kinds of materials like air will enter this volume and keep the density of the volume unchanged.

According to the research in the fluid dynamics area, the velocity change of any incompressible flow can be described by the incompressible Navier-Stokes equations:

$$\nabla \bullet \vec{U} = 0 \tag{2.2}$$

[13] This figure is downloaded from **awittyassniga.tripod.com/.../index.album?i=0&s=1**

$$\partial \vec{U} / \partial t = -\vec{U} \bullet \nabla \vec{U} - \nabla P / \rho + \upsilon \nabla^2 \vec{U} + \vec{f} \quad (2.3)$$

Where, $\vec{U}$ is the velocity field, t is time, P is the air pressure, ρ is fluid density, υ is the kinematic viscosity, and $\vec{f}$ is sum of external forces caused by the thermal buoyancy, gravity, and wind. Equation 2.2 is a mass conservation equation and equation 2.3 conserves the momentum of a flow. Equation 2.3 also tells us that the velocity change in a flow is determined by convection, air pressure, diffusion, and external forces, namely items on the right side of the equation respectively. For flames, which have small viscous effects [NFJ02], the simpler incompressible inviscid Euler equation can be used instead of equation 2.3, namely

$$\partial \vec{U} / \partial t = -\vec{U} \bullet \nabla \vec{U} - \nabla P / \rho + \vec{f} \quad (2.4)$$

Because the heat transfer in combusting areas depends on the velocity field of flames according to equation 2.1 and the temperature change also influences the velocity field inversely, dynamic flames can be visualized by interactively updating their velocity fields, temperature distributions, and positions of soot particles.

2.3.2 Smoke

Smoke is formed by soot and other combustion residues representing the fourth phase of the combustion process. Smoke generated by the wildland fire flows to a wider and higher region than fire flames, as shown in Fig. 2.2, and plays an important role in the wildland fire visualization. In order to efficiently visualize smoke, the appearance and movement of smoke and boundary conditions are introduced in this section.

2.3.2.1 Appearances

Smoke is a half-transparent amorphous phenomenon. Similar to clouds, smoke has self-shading properties which result from light multiple scattering among internal particles of smoke and the absorption of individual particle. Because smoke is dynamic at any time and internal particles do not keep static positions, light intensities on them are altered during every movement of smoke. Therefore, smoke can be rendered similarly as dynamic dense or sparse clouds. Fig. 2.7 shows dense smoke generated in a wildland fire, which looks apparently similar to clouds.

The temperature of smoke is much lower than the temperature of flames. Therefore, though soot in smoke absorbs light it encounters, it only radiates infrared light, which is dark to our eyes. Depending on the amount of soot in smoke, smoke generated by wildland fire appears in various degrees of grey. For example, when fuels in forests are too moist to have complete combustion, smoke may appear very dark. However, smoke from most chimneys is brighter, since materials especially for combustion are used. Generally speaking, smoke generated by more insufficient combustion contains more soot and looks darker than smoke generated by sufficient combustion.

Nevertheless, the color of smoke is not completely determined by incident light and light intensities on internal particles. Unlike clouds which are composed of water droplets or/and ice crystals, smoke may contain colorful components. For instance, unpurified smoke from the chimney of chemical factories may appear yellow or orange; and smoke resulting from collapses of buildings is in dark grey even illuminated by plenty of white light. Therefore, smoke rendering needs to take physical properties of internal particles into consideration.

Figure 2.7[14]: Dense smoke generated in a wildland fire.

2.3.2.2 Movements

In addition to the four main motions of flame, gravity also plays an important role in smoke movements. Unlike burning particles of flames which consume themselves very fast in a high temperature environment, internal particles of smoke travel long distances in the air till they are gradually out of sight. During this process, buoyancy, which is related to temperature, makes smoke rising quickly at first and then the balance of buoyancy, gravity, and air pressure slows this rising movement gradually down.

Like the motion of flames, the initial status of smoke determines its later journey to some extent. In addition, convection, diffusion, viscosity, and difference of pressures often produce many vortices in the velocity field of smoke and hence smoke normally appears turbulent. The motion of smoke is also easily disturbed by external forces such as wind. Therefore, smoke is unsteady and its velocity field changes continuously during every movement.

Accordingly, driven by the thermal buoyancy and interacting with the air flow, turbulent smoke can be simulated as incompressible flows and equations 2.1 to 2.3 can be used to model dynamic smoke directly [FM96, FM97]. However, smoke has many visible vortices in different sizes, as shown in Fig. 2.8, whether it is sparse or not. Therefore, adapting equation 2.3 with vortex $\vec{w}$ may help the reader to understand the motion of smoke.

Consequently,

$$\vec{w} = \nabla \times \vec{U} \tag{2.5}$$

$$\partial \vec{w} / \partial t = \vec{w} \bullet \nabla \vec{U} - \vec{U} \bullet \nabla \vec{w} + \upsilon \nabla^2 \vec{w} + \nabla \times \vec{f} \tag{2.6}$$

[14] This figure is downloaded from http://www.nps.gov/yell/slidefile/fire/wildfire88/crownfire/page.htm

(a) (b)

Figure 2.8: (a) [15] Sparse smoke; (b)[16] dense smoke.

Equation 2.6 is another form of equation 2.3. Without the item of pressure in equation 2.6, the change of vortex $\vec{w}$ is easier to be calculated. However, in this case, recovering the velocity field $\vec{U}$ from $\vec{w}$ becomes a difficulty. In order to obtain $\vec{U}$, another conception, streamfunction $\vec{\psi}$, needs to be introduced here. $\vec{\psi}$ has the properties of

$$\nabla \bullet \vec{\psi} = 0 \tag{2.7}$$

and

$$\vec{U} = \nabla \times \vec{\psi} \tag{2.8}$$

Take the curl ($\nabla \times$) of equation 2.8 gives:

$$\nabla^2 \vec{\psi} = -\vec{w} \tag{2.9}$$

Where $\nabla \times \vec{\psi} = ((\partial \psi_z / \partial y - \partial \psi_y / \partial z), (\partial \psi_x / \partial z - \partial \psi_z / \partial x), (\partial \psi_y / \partial x - \partial \psi_x / \partial y))$ Then, $\vec{U}$ can be recovered from $\vec{w}$ by solving equation 2.9, and 2.8 orderly. With continuously updating velocities in the region of smoke, smoke flows with the air and displays turbulent movements to the viewer.

2.3.2.3 Boundary Conditions

Unlike flames which take place in relatively smaller regions, smoke in the air may flow to much larger areas and has more chances to meet other objects. For example, smoke generated by the wildland fire often encounters vegetations in forests, hills, or mountains. Hence, in order to model the interaction between objects and smoke accurately, it is necessary to take boundary conditions into consideration.

Since the movement of smoke can be described by partial differential equations according to section 2.4.1, Dirichlet boundary conditions and Neumann boundary conditions are commonly used. The Dirichlet boundary conditions specify the value of a function on a surface directly. For example, we can specify that the normal velocity equals to zero at the non-penetration surface and the tangential velocity equals to zero at the non-slip surface. The Neumann boundary conditions specify the normal derivative of a function on a surface. For example, because of thermal conduction, the temperature at a surface will not change along the normal direction. Then, we can set the normal derivative of the temperature at the surface to be 0.

In addition, when the region of smoke is very large and no other objects are inside, repetitive boundary conditions can be applied. It is especially efficient when equation 2.3 or 2.4 is solved in the spectral domain, since the Fast Fourier Transform only works with repetitive boundary conditions.

[15] This figure is downloaded from www.newvictheatre.org.uk/plays/smoke.htm

[16] This figure is downloaded from fizyka.phys.put.poznan.pl/~pieransk/

That those boundary conditions are not discussed for flames does not mean there is no interaction between flames and objects. Since both flames and smoke can be modeled as incompressible flows, similar boundary conditions can be applied to the flame modeling as well.

2.3.3 Interfaces between Flames and Smoke

A flame and its smoke are two incompressible flows. Though smoke is generated by flames, they have different components, temperatures, and resultant colors. Because of the high temperature in flames, thermal buoyancy is the main driving force for the motion of flames. Moreover, unpredictable exothermic chemical reactions produce additional heat to flames from time to time. It results in more dramatically sudden changes to flames. Though the thermal buoyancy also plays an important role in the motion of smoke, the turbulence of smoke is mainly dominated by convection, diffusion, and other external forces such as wind. Therefore, a flame and its smoke need to be modeled as two flows.

However, since smoke is generated by flames, they have shared regions, and the front of a flame is called the interface between the flame and smoke. The velocity of flames and the velocity of smoke in these regions are related since smoke rises from flames. It is hard to determine which particles turn into smoke in flames. Therefore, the coupling of flames and smoke is difficult to model. For the visualization purpose, Nguyen et al. [NFJ02] simply assumed that smoke is the only residue of fuels; a flame and its smoke have the same tangential velocity at the front of the flame; but their velocities and pressures are discontinuous in normal directions of the front. They also derive balance equations under the assumption of conserved mass and momentum:

$$\rho_s\left(\vec{V}_s - \vec{D}\right) = \rho_f\left(\vec{V}_f - \vec{D}\right) \tag{2.10}$$

$$\rho_s\left(\vec{V}_s - \vec{D}\right)^2 + p_s = \rho_f\left(\vec{V}_f - \vec{D}\right)^2 + p_f \tag{2.11}$$

Where ρ_s and ρ_f are the densities of smoke and fuel respectively, $\vec{V}_s$ and $\vec{V}_f$ are the normal velocities of smoke and fuel respectively, $\vec{D} = \vec{V}_f - \vec{S}$ is the speed of flame front in the normal direction, $\vec{S}$ is an assumed fuel reaction speed, and *Ps* and *Pf* are the pressures of smoke and fuel respectively. With equation 2.10 and 2.11, the velocities of a flame and its smoke are coupled at their interface in normal directions. The tangential velocity of the flame can be directly used as the tangential velocity of its smoke according to their assumption. Their results proved that their assumptions and derivative equations are efficient.

This chapter introduces the meaning of knowledge-based visualization, and some physical properties of clouds, flames, and smoke. In general, natural gaseous phenomena have by far more complicated properties than we can perceive only from their appearances. Therefore, many approximations need to be done for the visualization purpose.

Chapter 3 Challenges and Objectives

This PhD thesis aims to visualize amorphous phenomena for the purposes of both visual reality and practicality. As we discussed before, visually convincing visualization of natural amorphous phenomena like clouds, flames, and smoke are widely required in games, movies, flight simulation, and so on. This area has been put into research for decades. Various modeling and rendering methods for the visualization of amorphous phenomena have been developed for different purposes and obtained promising results. However, because dynamic movements and self-shading properties of amorphous phenomena are hard to be described by simple linear equations, especially the turbulent behavior of fire flames and smoke, challenges are still remaining in either speeding up existing modeling and rendering methods or exploiting new fast and efficient algorithms. Faster visualization methods without losing visual quality are always pursued in computer graphics. Herein, this thesis intends to develop novel methods to accelerate the visualization of atmospheric clouds, flames, and smoke.

Except for virtual terrain rendering, advanced techniques in computer graphics are seldom used to display ecological processes and geographical processes based on scientific data. The role of knowledge-based visualization in this thesis is to make up this gap between pure imagination and scientifically simulated landscape evolution processes. Herein, knowledge-based 4D visualization can bring the public a wider view on what happened or will happen in the nature. Many ecological and geographical experts have devoted all their efforts to the analysis of recorded data and the prediction of successive processes. Their researches are invaluable to avoid the loss of properties and resources in future hazards and costly experiments. Hence, 4D visualization is worth being developed as an interactive interface for ecological and geographical simulation systems because of its attractive, intuitive, and easily understandable properties.

It is always desirable to visualize objects as photo-realistically as possible. However, limited to the computation and rendering speed of present hardware in common use, visual quality has to be compromised to achieve real-time visualization of large-scale amorphous phenomena. With the development of computer hardware, the speed of data processing will be faster and faster. When common personal computers are advanced enough, visually realistic modeling and rendering methods can then be applied to large-scale visualization without any compromise. In general, this thesis puts research on photorealistic visualization as well as real-time visualization with simplified methods. The methods without simplification can still be used to generate animations after all frames in series have been obtained; or be applied to large-scale photorealistic visualization in the future when the data processing speed of common computers is fast enough.

3.1 Photorealistic Cloud Visualization

Atmospheric clouds have amorphous appearances and various shapes which can not be described by simple functions. Hence, efficient modeling and rendering methods are necessary to visualize clouds as fast as possible without losing visual quality. Challenges existing in the visualization of atmospheric clouds can be divided into two main categories. One is to construct cloud models and simulate their turbulent movements; the other is to render them visually realistically.

3.1.1 Cloud Modeling

Cloud modeling is responsible to construct cloud models and simulate their turbulent movements. Undergoing convection, diffusion, vaporization, and other processes in the atmosphere, clouds have more than ten kinds of types with different appearances, and float at different altitudes in the air. Modeling clouds realistically determines the quality of cloud visualization to some extent. For some

rendering methods just with textures, such as 3D texture mapping technique, cloud construction is only to assign colors for texture grids. In this case, virtual clouds are neither interactive nor easily animated. Therefore, particles are used to construct clouds in most cases. Compared to the size of clouds, particles are very small and have primary shapes such as spheres or ellipsoids, individual positions, and other physical properties, for example albedos and extinction coefficients. Since the positions of particles can be easily changed and recorded, dynamic and interactive clouds can be easily modeled with the movement of particles.

In addition to self-driven movements, such as vaporizing, clouds also interact with the turbulent air flow. Hence, clouds also have turbulent movements. To describe cloud dynamics visually realistically in detail, physical simulation of unsteady velocity fields is desirable and straightforward. Fluid dynamics is a broadly used method to simulate turbulent fluidic velocity field and can conserve small-scale details; but realistic simulation of cloud velocity fields is hardly feasible because of the scarcely surrounding conditions of clouds and complicated self-driven factors. Therefore, this thesis utilizes the modeling method of smoke and flames to approximate the movement of atmospheric clouds.

Accordingly, the challenge of the atmospheric cloud modeling is to efficiently and automatically construct static and visually convincing cloud models with many small particles; and this thesis aims to develop a cloud modeling method to fulfill the following requirements:

- Randomness is the primary property of cloud models. For cloud models, positions of internal particles as well as their macrostructures should be irregular to avoid artificial results.
- Cloud models can be constructed automatically by a few given information. In general, a large number of particles are used to create one cloud model. It is more efficient to distribute particles automatically according to some rules other than manually arrange them with empirical knowledge.
- Two particles can partially overlap but not totally overlap. The overlapping will waste rendering time and memory space, and result in artificial speckles.
- Particles of one cloud model should not be distributed too sparsely. A sparse distribution can generate a fog-like cloud. However, a too sparse distribution can hardly create continuous amorphous appearance.

3.1.2 Cloud Rendering

Cloud rendering is to calculate the intensities of incident light scattered by clouds and transmitting through the atmosphere to the view point, and then display these clouds according to the calculated intensities. Clouds are lightened and colored by incident light. Incident light of clouds may come from various kinds of light sources. The most functionary light source is the sun. In some cases, the light reflected by lake, ocean, snow covered terrain, or other objects may also act as dominant incident light to clouds. Therefore, the intensities and directions of incident light are determined by the sunlight and the circumstances of clouds, and should be analyzed before the rendering process.

Because a cloud has many internal particles, when incident light transmits through the cloud, it is repetitively scattered by those particles and casts different light intensities on them. When some part of the light scattered by them transmits through other particles and intermediate atmosphere to the view point, cloud images are formed. A simple example is the moon, which scatters some part of light from the sun to our eyes to form its image. Consequently, clouds' self-shading properties appear very evident because of different intensities on their internal particles. As a typical example, Fig. 3.1 shows a cumulus cloud with self-shading properties. Therefore, complicated multiple scattering processes should be taken into account for photorealistic cloud rendering.

Figure 3.1[17]: A cumulus cloud with self-shading properties.

In addition to visual effects, rendering time cost is always the other primary concern of both researchers and users. Cloud models normally contain a large number of small particles. The problem of rendering clouds is that there are so many particles which allow some part of incident light on them passing through, but reflect some back, scatter some to other directions, and absorb some themselves. Those processes of light iteratively scattering back and forth will cost a lot of computing time. In addition, multiple light sources also cause extra computing burdens in the rendering process. Therefore, the challenge of rendering clouds is to develop an efficient and fast algorithm to trace the light multiple scattering among particles and display clouds visually realistically.

Accordingly, this thesis aims to develop an algorithm to:

- calculate intensity and direction changes of every incident ray when it passes through clouds and other intermediates before it arrives at the view point;
- take the contribution of every light source into consideration, as well as effects of multiple light sources on cloud colors;
- display cloud images according to calculated intensities;

3.2 Real-time Visualization of Migratory Insect Dynamics

Migratory insect dynamics and resultant vegetation changes have been put into research for many years and can be simulated by ecological systems based on field data collected in last periods [FB79, Fis82, Fis83, BF88, and BR99]. The objective of exploiting real-time 4D realization as the interface of underlying ecological simulation systems is to make simulation results to be displayed in a spatiotemporal manner and hence become more understandable to non-experts. In order to visualize them interactively in 4D, visualization system must work with ecological simulation systems and a system which provides further geographical information. Accordingly, a particular module for the display of migratory insect dynamics needs to be developed inside a visualization system and smooth

[17] This figure is downloaded from http://www.cloudman.com/gallery1_2.html

connection slots among the visualization system, ecological simulation systems, and a Geographical Information System (GIS) need to be built as well.

This thesis takes Larch Bud Moths (LBM) as a case study to illustrate how to visualize migratory insect dynamics and resultant vegetation changes in the landscape. Suppose a decision maker, with little knowledge about LBM migration, would like to see the patterns of larch forest defoliation over last 50 year time span in for example the Upper Engadine valley in Switzerland and the prediction of forest appearances in the next 50 years. He may change relative parameters through an interface and wait for the prediction shown in the 4D visualization. Here, the ability of fast data exchanges among systems becomes an important issue. In addition, when the user wants to interrupt a running visualization or simulation, commands should be transferred through systems and executed in real-time.

Outputs of LBM simulation system are statistical data without exact spatial information. Additional spatial information, such as where LBM migration takes place with various implications, should be obtained through a GIS. Moreover, the coordinates of new spatial data should be changed to match the coordinate system used in the visualization. In addition, since the outputs of the LBM simulation system also lack the exact information of LBM migration routes, an efficient solution is desirable to make LBM groups in the air fly to their respective destinations without any collision with the ground.

As we discussed before, groups of migratory insects can be visualized as 'insect clouds' in migration. Because LBMs are living creatures, they may change their relative positions quickly and actively. Hence, unlike normal atmospheric clouds, LBM clouds change structures very fast. Moreover, to make the analysis of LBM migration more precise, observing areas are normally divided into small pieces and many LBM clouds may appear in the air at the same time. Accordingly, simplified cloud modeling and rendering methods are required to achieve the visualization of many dynamic clouds in real-time.

In general, challenges of visualizing migratory insect dynamics in real-time are to:

- build smooth and fast connections between the visualization system and other information systems;
- calculate migration roads of insect groups;
- develop a simple and fast dynamic cloud modeling method;
- develop an efficient cloud rendering method which allows to display many insect clouds in real-time.

With LBM simulation outputs and spatial information, this thesis aims to develop a visualization module, which can:

- allow the user to select observing time span and area;
- display the area where migration takes place in 3D with vegetation appearance at that time;
- allow the user to distinguish insects migrating from different places in the air;
- display migratory insect dynamics in 4D;
- show resultant vegetation changes in terrain;
- allow the user to interrupt the visualization at any time and change parameters for a new simulation run.

3.3 Smoke Visualization

Smoke is an amorphous phenomenon and an unavoidable product of wildland fire. Smoke is always dynamic. Therefore, the visualization of smoke is to model its dynamics and display it in a visually realistic way.

3.3.1 Smoke Modeling

The motion of smoke is turbulent and varies in different circumstances. A small disturbance in the air can lead to a big disturbance in the motion of smoke because of the light weight of its internal particles. Smoke may interact with other objects, for example branches in wildland fire. Impenetrable interfaces can introduce additional turbulence to smoke and change directions of surrounding streamlines, in addition to simply bouncing back internal particles of smoke. Hence, the existence of other objects inside the smoke domain will result in more difficulties in the calculation of smoke velocity.

As introduced in the section 2.3.2.2, smoke can be simulated with the Navier-Stokes equations as a viscous incompressible flow. The Navier-Stokes equations are partial differential equations, which easily lead to divergent results in the computation process. The challenge of the smoke modeling is to solve these partial differential equations efficiently and as fast as possible.

Accordingly, this thesis aims to:

- simulate rising smoke driven by thermal buoyancy;
- simulate turbulent movements of smoke driven by convection, diffusion, viscosity, air pressure, and external disturbances with the Navier-Stokes equations;
- simulate the behavior of smoke when it encounters other objects;

3.3.2 Smoke Rendering

In addition to the effect of soot, the color of smoke also depends on incident light and the influence of its internal particles on the incident light. Therefore, in order to display smoke visually convincingly, it's necessary to trace incident light when it passes through smoke and calculate the light scattered to the viewer. Accordingly, besides dark soot, the process of light passing through smoke is similar as the process of light passing through clouds. Hence, smoke rendering can refer to the method of cloud rendering except that soot absorbs more light on it than other particles.

However, unlike clouds which are normally illuminated by nearly parallel incident light, smoke may be illuminated by light which has perspective directions starting from the spot light source. This causes extra computations, since in this case every ray of incident light has a different direction. Therefore, the challenge remaining in the smoke rendering is to efficiently display smoke illuminated by the spot light source. This thesis aims to develop an efficient and fast method to solve this problem.

3.4 Flame Visualization

Flames are composed of dynamic internal particles and the radiation of soot makes the appearance of flames colorful. Similarly to the smoke visualization, the flame visualization can be also divided into two main categories: flame modeling and flame rendering.

3.4.1 Flame Modeling

The motion of flame is turbulent and may be thought that it contains four main components: initial status, convection, diffusion, and buoyancy. Initial status of flame is essential in its combustion process. Because initial conditions determine speeds and directions of emitting gaseous fuels, their small changes can lead to radically different results. Moreover, due to small gravities of gaseous fuels, soot and other chemical components, the motion of flames can be easily disturbed by small external

forces or the heat released by internal chemical reactions. During the combustion process, gaseous fuels turn into gaseous products. Hence, flames have properties of convection, diffusion and buoyancy as flows, which can be simulated with the Navier-Stokes equations and the smoke modeling method can be applied in this case.

However, in addition to the motion like gaseous flows, flames have other apparent motional properties such as flickering, separation, and merging. As introduced in the section 2.3.1.1, temperature determines the status of gaseous fuels. After the temperature of gaseous fuels is higher than their ignition points, fuels start burning and changing into gaseous products. At the same time, heat is released and influences the velocity field of flames. Therefore, driven by the velocity field, some gaseous fuels may move to places that are detached from main combustion domains before they are ignited. It results in discontinuous flames. In other words, flames display flickering, separation, and merging properties under different conditions.

Since temperature plays an important role in the combustion process and has remarkable impact on the velocity field of flames, the challenge of the flame modeling is to simulate the temperature distribution in the overall combustion domain. This thesis aims to calculate the velocity field of flames by taking the influence of temperature on every internal particle into consideration.

3.4.2 Flame Rendering

Flame rendering is more complicated than the rendering of smoke and clouds, because flame is a participating medium which also emits light. The color of flames depends on the type of fuels, oxidizer being consumed, and the temperature distribution in the combustion zone. In most cases including wildland fire, a flame has a blue core, a bright orange-yellow inner layer, and a red outer layer. When temperatures in some parts of the combustion zone are high enough, those parts even appear bright white.

Because soot in flames absorbs almost all the light on it and emits light in different wavelengths according to its temperature, it should be treated differently from other internal particles of flames in the rendering process. Similarly to smoke, the cloud rendering method can be applied to render the effect of incident light passing through flames before it arrives at the view point except that soot influences the light intensity in a different way. Therefore, in order to display flames with proper colors, the challenge of the flame rendering is to find a suitable relationship between the simulated temperature and the intensity and color of the light emitted by soot. This thesis aims to display flames visually realistically by taking the influence of individual soot into consideration.

3.5 Real-time Visualization of Fire Propagation

This thesis does not aim to scientifically model the fire propagation in terrain, because it needs to investigate combustion conditions in the study area and there are already fire simulation systems, such as GRASS GIS, which can model the fire propagation by taking available influencing factors into consideration. Generally speaking, fire simulation systems can provide the ignition time and average flame length of every small region in the study area. However, these data are not enough for the 4D visualization of fire propagation in terrain, since the detail of fire spreading from one place to other places, initial conditions and movements of flames and smoke during the whole combustion process, which are essential information to the 4D visualization, are still missing. Fortunately, these details do not influence the user to understand overall fire propagation properties in a large scale. Therefore, simplifications can be made to approximate the intermediate process of fire spreading from one place to other places and the combustion process of flames.

Since properties of wildland fire propagation are analyzed and simulated in a fire simulation system, Communication between the visualization and the fire simulation system should be developed to achieve interactive visualization. In other words, it should allow the user to change fire simulation parameters through the visualization interface.

Accordingly, the challenge of the wildland fire visualization is to intuitively illustrate large-scale wildland fire propagation in terrain as visually realistically as possible. This thesis aims to visualize 4D wildland fire propagation in terrain in real-time, namely to.

- display terrain where wildland fire takes place in 3D;
- allow the user freely select an ignition point in terrain;
- communicate with a fire simulation system to obtain fire propagation information based on the selected ignition point;
- display fire propagation in terrain;
- visualize flames according to simulated flame lengths;
- visualize smoke generated by flames;
- display terrain changes caused by wildland fire;
- allow the user to interrupt visualization at any time and change parameters for a new simulation run.

Chapter 4 System Environment

This chapter introduces the system environment where the 4D real-time visualizations of LBM dynamics and wildfire propagation in terrain are implemented. In order to take the strength of existing ecological simulation systems and geographical information systems, a spatiotemporal system is integrated and a visualization subsystem is applied to display virtual terrains for the visualization of simulation results.

4.1 Integrated Spatiotemporal System

In order to interactively analyze, simulate and display spatiotemporal processes in terrain, an integrated system, Interactive, Process Oriented, Dynamic Landscape Analysis and Simulation System (IPODLAS), is being developed. Therein, a temporally explicit ecological modeling system is coupled with a spatially explicit Geographic Information System (GIS), and a real-time 4D visualization [IPW*06]. IPODLAS takes the strength of its subsystems to simulate and investigate spatiotemporal ecological or geographical phenomena and then display results in a visually perceivable way.

Accordingly, IPODLAS utilizes data and functionalities of three subsystems, GIS, Temporal Simulation system, and Visualization in a transparent and seamless way to profit from the individual strength of respective subsystems. Fig. 4.1 shows the software architecture of IPODLAS, which supports the smooth interaction of the subsystems and was designed by Daniel Isenegger in his PhD work.

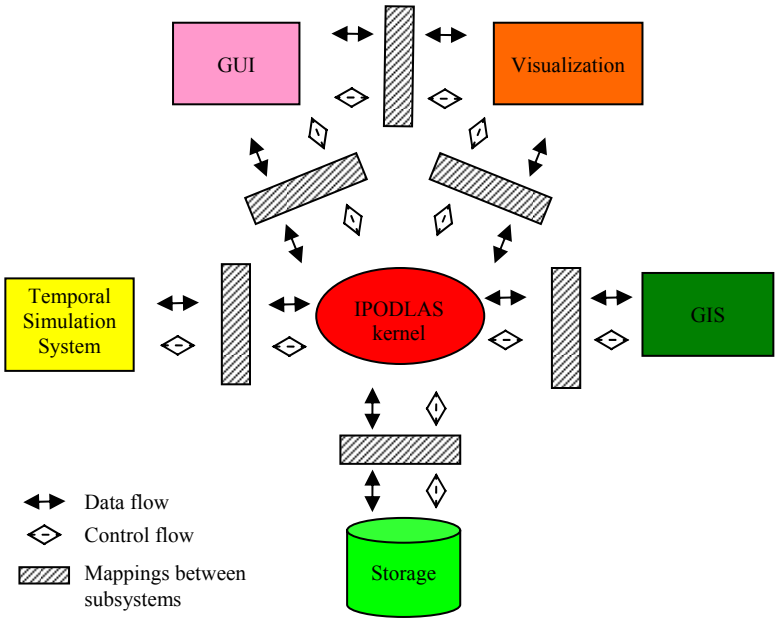


Figure 4.1[18]: The architecture of IPODLAS. The IPODLAS kernel is the central coordination component which controls the communication between the subsystems (GIS, Visualization, and Temporal Simulation), the Graphical User Interface (GUI), and the Storage. For performance reason, the design allows that the visualization subsystem can directly interact with the GUI, and not through the IPDOLAS kernel as required for the other subsystems.

[18] This figure was generated by Daniel Isenegger. GIScience Center, University of Zurich, disen@geo.unizh.ch .

IPODLAS is designed as a user-driven system, which means that the action of users in the GUI triggers the GUI to invoke activities, possibly in all other subsystems [IPW*06]. These activities are coordinated by a synchronization mechanism, which is one of the main functionalities of the IPODLAS kernel. The IPODLAS kernel is a server which is in charge of the communication among subsystems to avoid data jams. For example, when the IPODLAS kernel receives a user request from the GUI, it gives a primary feedback to the user and breaks the request down into subtasks, which are sent to appropriate subsystems. After completing subtasks, the subsystems send results via the IPODLAS kernel to the GUI and user. All the data transferred among subsystems must go through this kernel. It holds input data first and decides the right time to transfer the data to its objective subsystem. Because subsystems can be on different platforms, data are transferred as strings. Large data sets are not possible to be transferred through sockets in an efficient way. In this case, data are first stored as files and only their names and locations are transferred.

The IPODLAS kernel not only supports the interoperability of the subsystems, it also allows a subsystem to be easily exchanged [IPW*06]. For example, if the user wants to use another temporal simulation system, this system needs only to be registered in the IPODLAS kernel. Since all the subsystems communicate through the IPODLAS kernel, different protocols map the data models of the subsystems to the data model of the IPODLAS kernel. To support seamlessly accessing the required functionalities of the subsystems from the GUI, the mapping of the functions of the subsystems to the IPODLAS kernel is also mandatory [LBY96]. However, in order to speed up data exchange, some interactions between the GUI and Visualization do not go via the IPODLAS kernel.

The main advantages of IPODLAS are that the user can execute parallel simulations on different computers in different places and observe the results in an easily understandable way.

4.2 Virtual Terrain Project (VTP)

In the visualization subsystem, an efficient technique is necessary and essential for efficient terrain rendering. To visualize spatiotemporal processes in terrain, namely allowing extension of existing functions other than pure terrain rendering, the source code and access to the low level rendering Application Programming Interface (API) must be available. The Virtual Terrain Project (VTP) satisfies these requirements and is selected to be the base of the visualization subsystem in this thesis [RH98, DK00, and LH04].

VTP is a creative project for easy construction of virtual terrain in an interactive, 3D digital format. It achieves real-time terrain rendering with large data sets because its run-time environment makes use of continuous Level of Detail (CLOD) algorithms [RH98]. LOD simply means terrain near the view point is visualized in detail and terrain far from the view point is visualized gradually roughly. In fact, it is a typical cross-scale modeling technique. With the OpenGL API or its extension, OpenSceneGraph, a visual system dedicated to simulation and analysis of a dynamic process can be implemented as an embedded module of VTP.

As visualization software, VTP has its own interface, which allows the user to load various DTMs and image to build terrains and allows the user to select simple objects, such as trees, fences and buildings, and ‘plant’ them at particular places in terrain through the mouse. Because VTP is open-source software, we can freely add our menus in the interface and implement corresponding embedded modules.

VTP provides a good mouse navigation system, through which the user can freely navigate in the virtual landscape and change the navigation speed through the input of keyboard. It also has the time-dependent illumination system, with which the user can view the virtual environment at different time

of a day. Therefore, VTP provides the user an easy framework to add new dynamic modules in a virtual landscape without extra programming work.

VTP is programmed with wxWindows and OpenGL API. wxWindows is a cross-platform language based on C++. Hence, the user can always program with C++ or/and wxWindows in the VTP framework. OpenGL is a general API for 3D visualization. Though OpenGL has some extensions based on advanced GPU, its basic functions are able to meet our requirements and are available in almost all the GPUs which are installed in present computers.

The connection between VTP and the other subsystems in IPODLAS is given through TCP/IP socket. Both wxWindows and C++ have the potential to open a socket and write/read data through it. Because the IPODLAS kernel is responsible to dispatch tasks, VTP works as a client, which 'listens' to the command from a server and returns a message after it finishes its task. As a client, VTP cannot actively ask for connection to the server. Therefore, VTP can only communicate with the other subsystems in IPODLAS when the server is open to it.

This thesis implements the visualizations of migratory insect dynamics and wildland fire with C++ and OpenGL as embedded modules in VTP.

Chapter 5 Atmospheric Cloud Visualization

Atmospheric clouds are a common natural phenomenon. Their generation and motions in the air are too complicated to be described here. In general, clouds are composed of water droplets and many kinds of ice crystals. Due to different structures and ratios of components, dynamic air flows, and the light scattering effects of the atmosphere, clouds appear in a variety of pure and transitional forms and colors [Hof00].

The appearance of clouds in virtual scenes enhances the visual reality. Clouds play an important role in interactive flight simulations, games and virtual weather forecasts. Hence, cloud visualization, including cloud modeling and cloud rendering, has attracted graphics researchers for decades. The cloud modeling contributes to cloud construction and dynamics, while the cloud rendering contributes to displaying photorealistic cloud images. They support each other in cloud visualization. Without a realistic construction scheme, a perfect cloud rendering algorithm cannot generate a visual convincing cloud image. Similarly, an ideal cloud construction cannot produce a photorealistic cloud image without an efficient rendering method as well. Therefore, in addition to the contribution of this PhD work, developments and achievements on both sides for the cloud visualization are introduced and analyzed in this chapter.

5.1 Cloud Modeling

Since the cloud modeling contributes to virtual cloud construction and dynamics, it is so important that without a visual realistic cloud model, none of the cloud rendering algorithms can generate a photorealistic cloud image. Generally speaking, in recent years particle systems [DKY*00, HL01, and NDN96] are commonly used to create cloud models with experimental distributions in an initialization step. Particle systems are first introduced by Reeves [Ree83] to model clouds and other amorphous phenomena. To generate photorealistic images with self-shading properties for half transparent gaseous phenomena, we need to take the multiple scattering of light among internal particles of clouds into consideration. Moreover, particle systems provide an easy animation control. Continuously updating the position of particles according to calculated velocity fields, we can simulate and visualize the movement of dynamic gaseous phenomena, including clouds.

Modeling the motion of gaseous phenomena can be divided into 3 categories. The straightforward method is fluid dynamics [FSJ01, FM97, and Sta99]. Though it is physically based and realistic, currently it is rarely used to animate clouds since it is a computationally expensive scheme. For relative static clouds, it is too luxurious. Heuristic approaches are computationally inexpensive and much easier to implement [Eeb97, RE03], though they are neither flexible nor interactive when circumstances change. Cellular automation lies between the above two approaches [DKY*00, WZF*04]. It largely simplifies the computation of fluid dynamics but roughly keeps the influence of surrounding subjects into consideration. Because these methods will be discussed in detail in chapter 7 for describing the continuous motion of smoke and flames, this section concentrates on the construction of static clouds.

The generation of promising cloud models has been a challenge in computer graphics for more than two decades [Dun79]. Researchers have tried many methods; but a general and efficient method is still not available. The following two sections introduce two volumetric cloud modeling methods based on particle systems in preparation for dynamic clouds. With these two methods, various cloud models can be constructed after few trials.

5.1.1 Procedural Modeling

As a volumetric cloud modeling method, procedural modeling exploits a two-level model: the cloud macrostructure and the cloud microstructure. Implicit functions are used to model the cloud macrostructure because they can smoothly blend density fields from separate primitive sources. The implicit primitives can have any geometry. For computational convenience, spheres and ellipsoids are commonly used by users. With a selected type, the cloud macrostructure can be easily constructed by specifying the locations and weights of the implicit primitives. The following two equations [EMP*03] describe a simple example of a density field defined by spherical implicit primitives:

$$Density(p(x)) = \sum_i w_i f(|p(x) - C_i|); \tag{5.1}$$

$$f(|p(x) - C_i|) = 1 - |p(x) - C_i|/R; 0 < |p(x) - C_i| < R \tag{5.2}$$

where $p(x)$ is a random place inside the density field, w_i is the weight of the i^{th} primitive, C_i is the center of the i^{th} primitive, and R is the functional radius of the primitives. Here, the particle density field is smoothly assigned and a cloud macrostructure is defined.

The cloud microstructure is created by procedural turbulence and noise functions [Ebe97, EMP*03], which can generate small turbulent density fields. Herein, a uniformly spaced orthogonal 3D grid is constructed to fully contain a density field defined by implicit primitives. A procedural noise function first generates random values for all the grid points. The noises in other places are calculated by trilinearly interpolating these values. A procedural turbulence function, for instance a standard Perlin turbulence function [Per85], is then used to generate turbulent values for a turbulent density field according to the noise volume in the 3D grid. The final density field is defined by blending the turbulent density field with the density field defined by implicit functions. Spherical particles with different properties, such as transparence and radius, are generated to create cloud models according to the final density distribution.

Procedural modeling can create promising turbulent cloud models, which have especially 'fuzzy' edges and smooth appearance. However, for cumulus clouds, which look more like cotton balls, the marching cube algorithm [LC87] is more useful.

5.1.2 Marching Cube Modeling

Marching cube modeling is to generate cloud models with the marching cube algorithm. It allows producing random and visually convincing cumulus clouds [DKY*00, NDN96] with simple predefined macrostructures. The basic principle of the marching cube algorithm is to subdivide a space, which fully contains 3D targets, the cloud macrostructure in this case, with a uniformly spaced orthogonal 3D grid, namely a series of small cubes; and find out the intersections of the targets with the edges of those small cubes; then replace the targets with an appropriate set of polygons using the intersections as vertices. Here, the cloud macrostructure can be constructed by a set of large spheres and ellipsoids. A cloud model can then be constructed by iteratively locating the centers of small spherical particles inside the replacing polygons of the cloud macrostructure or its extending shape after the first cycle, which takes the already placed small particles into consideration. Since clouds have 'fuzzy' edges, the exact approximations of the macrostructure and its extensions are unnecessary. Therefore, to save computing time, the middle points on those cube edges are used to form the approximate polygons instead of the exact intersections.

To understand the algorithm more easily, a 2D example is shown in Fig. 5.1. To obtain the approximate lines, the grid points inside the round are marked first (indicated by the green points). Then the middle point of each edge with one marked vertex and one non-marked vertex are connected to form the lines. From Fig. 5.1 we can find that, how well the lines fit to the circle, depends on the distance between two grid points. With the same reason, the size of grid cells in 3D determines how well the overall constructed shape of a cloud model closes to the cloud macrostructure.

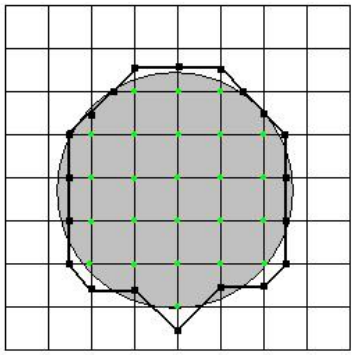


Figure 5.1: 2D approximate lines of a round.

Grid cells in 3D are cubes which have 8 vertices and therefore potential 256 possible combinations of vertex status. Taking into account the following conditions for any cube, we can group vertex status down to 15 combinations.

- Rotation around any of the 3 primary axis
- Mirroring the states of vertices across any of the 3 primary axis
- Inverting the states of all vertices

Fig. 5.2 gives an example of data sets covering all of the 15 possible combinations. Accordingly, appropriate surface approximations can be constructed more easily. The small green dots denote vertices that have been tested inside the volume of approximated 3D objects.

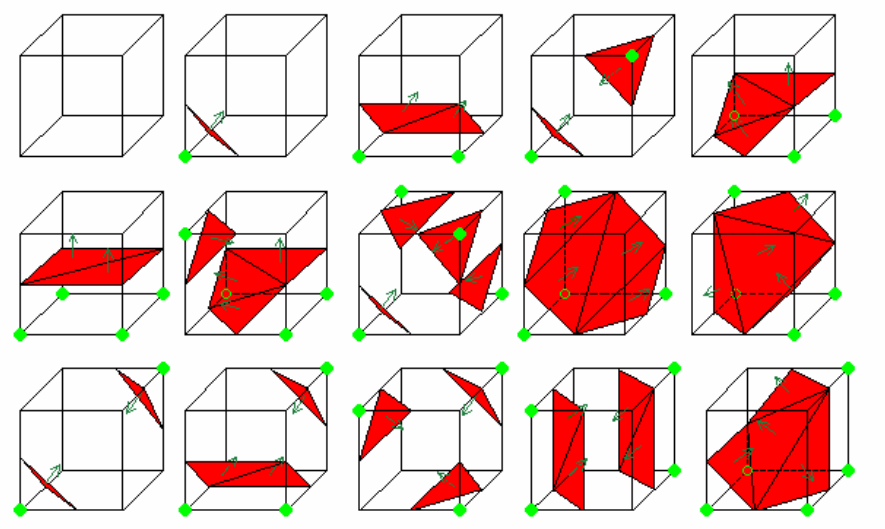


Figure 5.2[19]: 15 vertex status combinations.

[19] This figure is downloaded from http://www.essi.fr/~lingrand/MarchingCubes/algo.html

With the marching cube algorithm implemented according to the basic 15 combinations in Fig. 5.2, the cloud macrostructure can be approximated by a set of triangles inside a uniformly spaced orthogonal 3D grid, but some flaws may appear under some special conditions. However, since this algorithm is used to create cloud models, which allow and even demand irregular edges, it is unnecessary to spend extra modeling time on using more advanced refined algorithms. Therefore, cloud models can be constructed by repetitively going through all approximate triangles to randomly locate the center of a small spherical particle inside each triangle with individual properties, such as transparency and radius, and revising the vertex status of the grid according to the new inserted particle for new approximate triangles.

Fig. 5.3 shows a cloud macrostructure constructed by 3 big spheres and a corresponding cloud model constructed by 20000 spherical particles. Herein, the number of spherical particles is randomly selected. It is only limited by the size of the computer memory. Though particles are well placed surrounding the cloud macrostructure, the cloud model keeps the basic shape so strictly that it looks very artificial. Then to form a visually convincing cumulus cloud, the cloud macrostructure needs to be constructed by many big primitives and located in expected positions. Therefore, in order to create a cumulous effect with few primitives, a two-level marching cube method is exploited in this thesis. First a coarse uniformly spaced orthogonal grid is constructed for the cloud macrostructure. Accordingly, some small particles with different radiuses are located. Then a new uniformly spaced orthogonal grid is build with fine resolution (much smaller cell size). Basing on the already placed small particles, marching cube algorithm is preformed again for placing other small particles to create a final cloud model. Fig. 5.4 shows four cloud models created by the two-level marching cube method with the cloud macrostructure in Fig. 5.3.

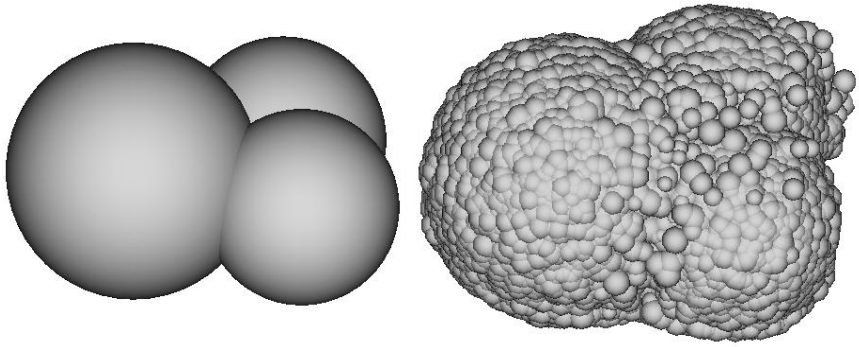


Figure 5.3: A cloud macrostructure constructed by 3 spheres and a cloud model constructed by 20000 spherical particles based on the macrostructure.

From Fig. 5.4 we can find that the choice of cube size for the second run plays an important role in cloud modeling. Because the first cloud model in this figure uses very small cell size compared to the radius of particles, they are placed very close to each other and the dense accumulation uses up all the particles before going through all the previous placed particles in the first run. Accompanying with the growth of the cell size, particles become so small comparing to the cell size that only some ‘lucky’ particles cause the cube vertices marked after the first run. Hence, most particles are placed around those ‘lucky’ particles. In general, a cube size which is smaller than the average radius of particles can be used to create a promising cloud model. Fig. 5.5 shows a cloud image based on the top right model in Fig. 5.4, rendered by the recording matrix method presented in section 5.2.7. This cloud is modeled by the two-level marching cube method with only 3 user predefined primitives as shown in Fig. 5.3.

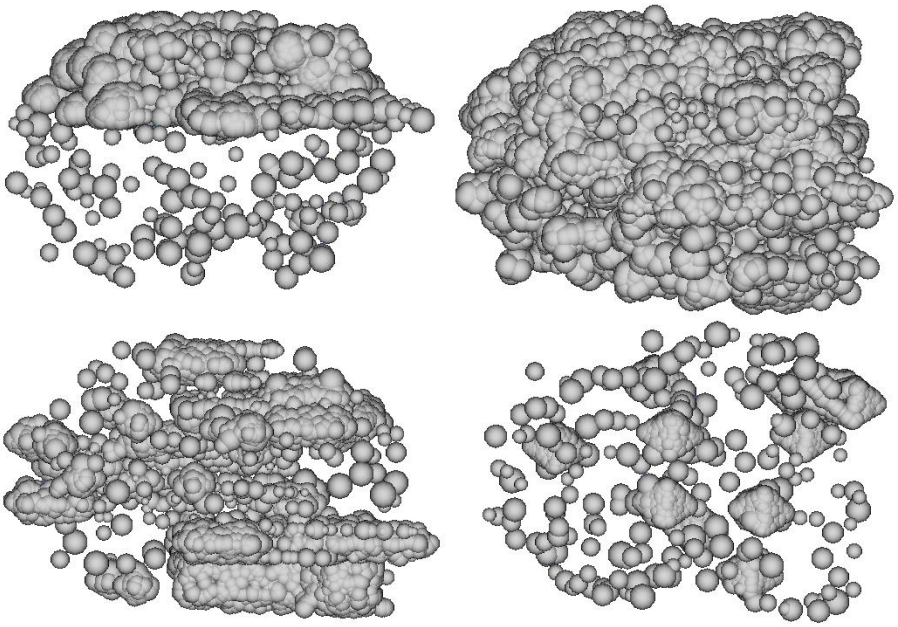


Figure 5.4: Four cloud models with 20000 particles created by the two-level marching cube method. From left to right and from top to bottom, the cloud model uses a finer resolution grid in the second run than the previous one. All models use the same cloud macrostructure in Fig. 5.3 and a same resolution grid in the first run.

Figure 5.5: A cumulus cloud image based on the top right model in Fig. 5.4.

5.2 Cloud Rendering

Rendering clouds efficiently and realistically is demanded in some applications such as flight simulations, games and weather forecasts to improve the virtual scenes. Especially for dynamic clouds, a fast and efficient rendering method is desired to achieve high frame rates. All cloud rendering methods aim to display photorealistic cloud images efficiently according to cloud models. However, because of the amorphous appearance and the self-shading properties of clouds, fast and realistic cloud rendering is still a challenge. Limited to the data processing speed nowadays, it is necessary to find an ideal compromise between rendering speed and rendering effect. Accordingly, previous work on cloud rendering can be divided into two categories, mapping techniques and physical models, which stress rendering speed and rendering effect respectively.

Mapping is a non-interactive rendering technique. Gardner mapped transparent fractal textures onto ellipsoids to represent clouds [Gar85]. Kluyskens used solid-textures to color clouds and control the transparency of particles which define general cloud shapes [Klu02]. Wang proposed a different, artistically based splat texture method, which uses few 2D cloud-like particles to give clouds better details (shape and shading)[Wan04]. It is a trade-off to physically based illumination and renders clouds like watercolor paintings. Their non-interactive results prove that displaying realistic clouds, especially animated clouds, requires the consideration of the complicated physical process of light transmitting through cloud volumes.

Hence, to reproduce cloud images as those perceived by human eyes, it is necessary to mimic the physical process of light affected by the internal particles of clouds and atmosphere before entering eyes. The internal particles of clouds have high albedos [Boh87]. Therefore, light multiple scattering and absorption due to the internal particles can not be ignored in the rendering process, though it is more time-consuming. Consequently, the light intensity on a particle is the result of light from light sources after absorption by intermediates and the scattered light from other particles. After the light scattered by internal particles of a cloud passes through the atmosphere and reaches the viewer, a reflection of the cloud is formed. Therefore, Nishita *et al.* applied anisotropic multiple scattering and took into account sky light by calculating its spectrum to render clouds[NDN96]. Stam *et al.* simulated light scattering by particles in smoke using a ray tracing technique [Sta99, FSJ01]. Dobashi *et al.* took the advantage of graphics hardware acceleration technique and applied an approximation of isotropic single scattering to accelerate the cloud rendering process [DNO98].

Herein, volume rendering is a popular and efficient method commonly used in medical visualization and amorphous phenomena rendering. It traces each ray or ray cluster transmitting through cloud volumes to calculate the light intensity scattered to the view point. Generally speaking, there are four rather popular volume rendering algorithms for directly rendering datasets, Raycasting [Lev88], Splatting [Wes90], Shear-warp [LL94], and 3D texture-mapping hardware-based approaches [CCF94]. In the last two decades, many researchers have investigated on refining these four algorithms. Due to multifarious efforts, now all methods have reached a high level of maturity.

In addition, a two pass method is recently broadly used to describe the physical processes of light passing through clouds and scattered by them to the view point. The two pass rendering method is first used by Kajiya and Herzen in computer graphics to display clouds [KH84]. Therein high order spherical harmonics are used to express the intensity distribution of each small subdomain. The two pass method is not only sound in theory, but also leads to visually convincing results no matter where a viewer stands. Compared to the second pass, the first pass, which calculates light intensity on each particle, is an expensive process. A hardware acceleration method was developed to fast process the first pass [DNO 98, DKY*00, and HL01]. To further reduce its computing time, a new method, called recording matrix, is developed in this thesis to efficiently obtain photorealistic results with ten time's faster speed than the hardware acceleration method. The recording matrix method is designed to not only accelerate static cloud rendering, but particularly enhance the rendering speed of animated clouds. For dynamic clouds, internal particles change positions continuously after every movement caused by wind, self evolution or other reasons in the atmosphere. Hence, light intensities on particles need to be recalculated again and again. Rapid movements of light sources in some games result in a mandatory recalculation of the first pass as well. In order to express this condition conveniently, it is considered in this thesis as animated clouds with static light sources.

The following sections present some popular methods used to render clouds and the recording matrix method developed in this PhD thesis. Since smoke and fire flames are also amorphous gaseous phenomena as clouds, especially dense smoke from chimneys, the cloud rendering methods can be directly used to render them.

5.2.1 State of Art

5.2.1.1 Ray Casting

Ray casting is a straightforward algorithm. Assuming a ray ejected from a view point, passing through an object volume and then reaching a plane perpendicularly to the view point, its light intensity on the plane decides the color of a pixel in the final visualized image. Fig. 5.6 shows the basic idea of ray casting. We can perform several operations along rays within the volume to obtain the color of each pixel of the image. The most commonly used operation is integration. It simply sums up data values along rays to obtain X-ray like images. In general, the sample distance of rays should be less than the cell edge to avoid any information loss. Cell is a single data sample within the volume, which is given at a well-defined position in 3D space. Each cell can have one or more attributes except position, such as density, pressure, transparence, albedo, color and so on, which determine the contribution of data samples to pixel color values. Another popular operation is the selection of the maximum values along rays, which is called maximum intensity projection (MIP).

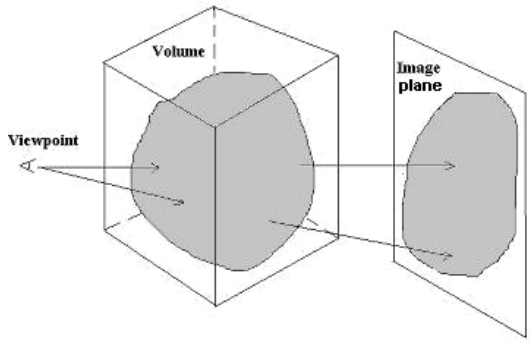


Figure 5.6: Ray casting.

Though ray casting is easy to understand and straightforward, it is a rather time-consuming method of volume rendering. From Fig. 5.6 we can observe that it is especially costly when an object appears large on the image plane (computer screen), namely the image containing more pixels and need more calculations since this method traces one ray for one pixel on the image. Its rendering in particular the calculation speed can be significantly improved, if regions, which do not contribute to the image, are skipped from rendering. Normally such regions contain either entirely transparent or high opacity cells. Transparent parts can be excluded by encoding each single cell. To omit high opacity regions, a popular method is early ray termination, where a ray can be terminated when its accumulated opacity has reached a value close to unity.

Besides those regions without any contribution to the image, there are some cells, which contain low-important information like low-valued background noise, in the including area. Neighborhood-based elimination (http://www.cg.tuwien.ac.at/~mroz/diss/html/node18.html) and shadow sweep elimination (http://www.cg.tuwien.ac.at/~mroz/diss/html/node19.html) are two powerful acceleration methods to optimize the rendering process. The first one excludes a cell by identifying its invisibility from any viewing direction based on its local neighborhood. The second algorithm groups possible viewing directions into several clusters and produces a set of potentially contributive cells for each cluster.

5.2.1.2 Splatting

Unlike ray casting, which traces along one ray for one pixel in an image at a time, splatting is an object-order algorithm, which computes the contribution of each cell to all pixels of an image at a time [Wes90, NMM*06]. Fig. 5.7 shows the basic idea of splatting. The area affected by the projection of a

cell is called footprint. Within the affected area, the cell contributes to the color of a pixel according to a Gaussian distribution (or similar), which has better anti-aliasing characteristics than linear filters, around the center of the footprint. An image is then generated by projecting all the contributive cells and accumulating their contributions for each pixel. The major advantage of splatting is that only cells relevant to the image need to be projected and rasterized.

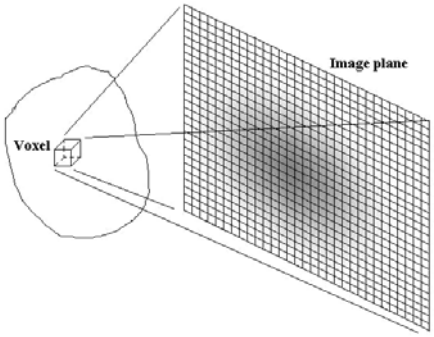


Figure 5.7: Splatting.

Splatting also has a similar acceleration algorithm like early ray termination [Lev88, Lev90], called early splat elimination [MSH*99]. It bases on a dynamically computed screen occlusion map that culls invisible splats early from rendering pipeline. Unlike early ray termination skipping remaining cells along ray directions, though early splat elimination saves the cost of footprint rasterization for invisible cells, their transformation must be performed to determine their occlusion.

5.2.1.3 Shear-warp

Shear-warp is recognized as the fastest volume rendering approach for non-orthogonal projection up to date [LL94]. This algorithm has to accomplish the following three conceptual steps:

1. Shear and resample volume slices, such that rays are perpendicular to the volume slices. Then the rays obtain their sample values through bilinear interpolation. Comparing to the trilinear interpolation in ray casting and the Gaussian interpolation in splatting, the bilinear interpolation is obviously easy to accomplish.
2. Project resampled cell scanlines onto intermediate image scanlines and compose the resampled slices together in front to back order. Then use the method similar to ray casting to obtain an intermediate image in sheared object space.
3. Warp the 2D intermediate image into the final screen image.

There are several kinds of projection in computer graphics. In this algorithm, since we need to shear and resample the volume slices, the way of projection should be taken into account. Two popular projections are discussed here; others can be analyzed in a similar way.

For the user's better understanding, Fig. 5.8 illustrates the basic steps of shear-warp algorithm for parallel projection and for perspective projection separately. The horizontal lines in this figure represent cross sections of volume slices.

From this figure, we can find that after transformation, the volume slices have been sheared perpendicularly to the viewing direction. The process for the perspective projection is similar as for the parallel projection. The only difference is that each slice of the volume data must be scaled as well as sheared for the perspective projection because of the perspective property. All the cells in a given slice are scaled by the same factor. In general, the slice closest to the view point is scaled by a factor one so that no slice is ever enlarged. Because extra time is needed to compute scale factors, the

perspective projection need more computing time than the parallel projection in the rendering process, but it is still much easier than the computation along each non-orthogonal ray. Therefore, this algorithm is very useful and efficient when the volume data can only be translated and scaled other than rotated.

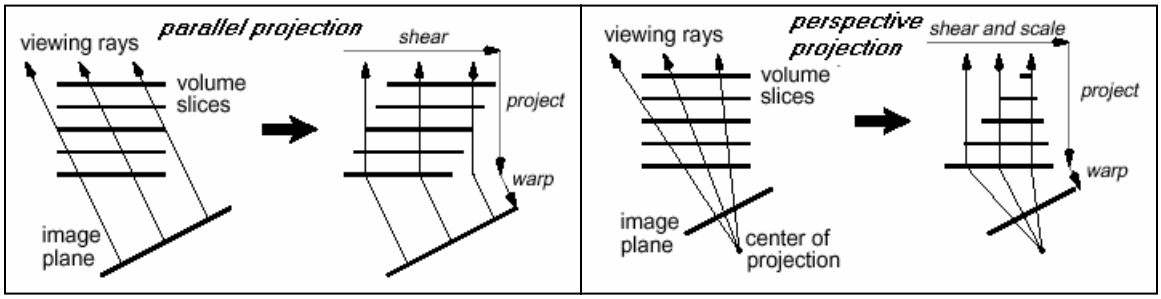


Figure 5.8: the shear-warp algorithm [LL94].

5.2.1.4 Warped Blob Integration

Like ray casting, warped blob integration is also a method which computes the integrals of density fields along given rays [SF93]. However, in this method, warped blobs instead of normal cells are used to make up of density fields. The motivation behind this method is to choose samples according to the locations of blob representations. It has two advantages. First, since we already know ray tracing directions, blob representation locations and their interactive functions, we can decrease the number of samples by choosing them at right places. Second, samples can move naturally with blob representations in an animation. It reduces the occurrence of visible artifacts over time. Comparing to ray casting, the basic idea of this method is illustrated in Fig. 5.9.

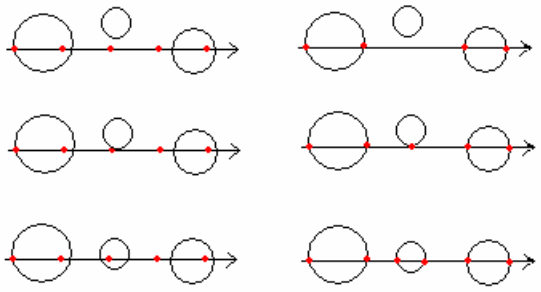


Figure 5.9: Uniform sampling for the ray casting (left) versus blob sampling for the warped blob integration (right).

Let's assume a mass density $\rho(x,t)$ at location x and time t, and $c_1(t),...c_n(t)$ are the centers of arbitrarily simple blobs which change locations and radius over time. Associating a set of mass $m_i(t)$, the mass density can be expressed as:

$$\rho(x,t)=\sum_{i=1}^{n} m_i(t)W(x-c_i(t),\sigma) \tag{5.3}$$

Here, σ is the radius of a smooth kernel $W(x-c_i(t),\sigma)$, which can be a simple triangle function:

$$W(x,\sigma)=\begin{cases}1-|x|/\sigma\,, when|x|\le\sigma;\\ 0, otherwise,\end{cases} \tag{5.4}$$

or a Gaussian distribution:

$$W(x,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/(2\sigma^2)}. \tag{5.5}$$

Then the influences of physical properties such as gravity and buoyancy on the density field can be described by function:

$$f(x,t)=\sum_{i=1}^{n}\frac{m_i(t)}{\rho(c_i(t),t)}f(c_i(t),t)W(x-c_i(t),\sigma) \tag{5.6}$$

These influences result in a deformation from the original spherical blobs to warped blobs. An example is shown in Fig. 5.10, where σ is the radius of the blob at time t-Δt, δ is the expanding speed of blob radius, and x^{-1} is the position of x at the time Δt before t.

Computing the integrals of density fields constructed by warped blobs over given rays can be achieved by two methods. One method is to integrate warped blobs along rays and the other is to backwarp rays and integrate unwarped blobs over bended rays. Fig. 5.10 shows a point in a warped blob and its corresponding point in its unwarped blob. The following Fig. 5.11 illustrates differences between the two approaches. Stam and Fiume adopted the method of integrating unwarped blobs over a curve, because it goes back to the spherical blob integration algorithm, which is easier to compute [SF93, Sta94].

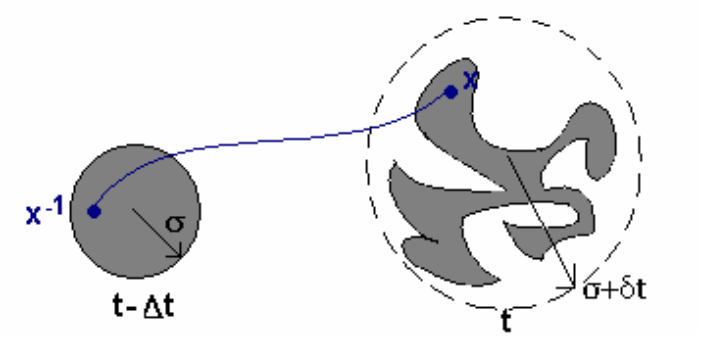


Figure 5.10: Original blob and blob after warping [Sta94].

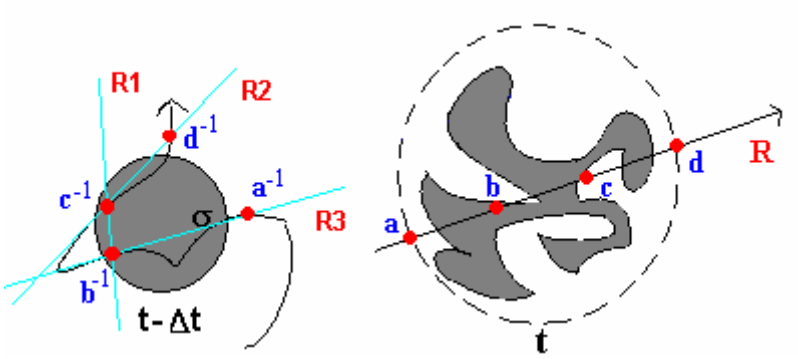


Figure 5.11: Integrating a warped blob [Sta94].

As an example, Fig. 5.11 shows four red sample points, *a, b, c*, and *d*, on the same ray, *R*, through the warped blob are back traced to the positions, a^{-1}, b^{-1}, c^{-1}, and d^{-1}, related to the original spherical blob at the time Δt before *t*. Making a line through arbitrary two successive traced-back points in the spherical blob generates three lines (rays), *R1, R2,* and *R3*. The approximate integral over the ray, *R*, in the warped blob can then be obtained by summing up the integrals over three segments on rays *R1, R2,* and *R3* through the original spherical blob. Because the integrals of intensity changes along segments in the spherical blob are easier to be calculated than the integral of intensity changes along ray *R* in the warped blob, the backwarp method makes sense.

This method is effective for warped blobs, which have short lifetimes only. The additional cost of the warping is thus linear to the number of chosen samples and is independent of the number of blobs.

In general, the above four rendering methods can be applied to render photorealistic clouds, fire flames and smoke constructed by 3D particles, because these methods are capable of directly representing objects with transparency and light-emitting properties. Among them, the shear-warp algorithm is the fastest one. Its bilinear interpolation takes less time than the trilinear or Gaussian interpolation in other algorithms because rays are only along the normal direction of volume slices in this method.

5.2.1.5 3D Texture Mapping

The 3D texture-mapping hardware-based approach depends on the capabilities of Graphics Processing Unit (GPU). Users can utilize a standard Application Programming Interface (API), notably OpenGL or Direct3D, to render objects directly with 3D textures, if it is supported by a GPU. Polygons perpendicularly to the view direction can be placed within a texture volume and textured with color or other attributes. By blending the textured polygons from back to front, objects are rendered. So the quality of a rendered image depends on the number of rendered slices. If the 3D texture mapping is not supported by a GPU, 3 perpendicular sets of polygons and corresponding textures are prepared. The set of polygons, which are most likely perpendicular to the view direction, is rendered.

The 3D texture mapping is one of the volume rendering methods. Like other advanced volume rendering methods, it deals with cells, which represent data as volume-based entities. Herein, GUP does linear interpolation for the color on each cell throughout the volume automatically with several parallel inner processors. Therefore, the 3D texture mapping method can highly reduce the work of developers and accelerates rendering processes. For small volumes containing up to 256^3 cells, this method can achieve high frame rates faster than 25 frames per second.

However, the 3D texture mapping only provides a subset of software-based approaches. For example, OpenGL APIs require the numbers of pixels in height, width, and depth for 3D texture images to be 2^n, it limits the usage of the 3D texture mapping.

Generally speaking, the 3D texture mapping is suitable to render precalculated objects from different view directions. However, it lacks flexibility and efficiency when the colors of an object change dynamically, the shape of the object becomes irregular, or the object interacts with other objects. In the following sections we introduce a more flexible and efficient rendering method, which is combined with particle systems and calculates the light intensities on the internal particles of gaseous phenomena before rendering them from various view directions.

5.2.1.6 Hardware Acceleration Method

Instead of the direct integration of light intensities along given rays through the data volume to the viewer, the hardware acceleration method is a two pass method. The first pass is to calculate the light

intensities on internal particles of gaseous phenomena, for instance clouds and smoke. The second pass is to render all the particles by setting the multiplications of their light intensities and incoming light color as their colors. These colors can be linearly readjusted according to the positional relationship between particles and the viewer. Then, 2D polygons textured with the images of representing particles are blended to display the final image according to extinction factors. The main advantage of the two pass method is that the first pass only needs to be calculated once unless incoming light changes direction or gaseous phenomena deform themselves. Since the first pass is the most computationally consuming process, the two pass method largely saves the rendering time when the first pass is only necessary to be recalculated once. For example, a viewer looks at a static cloud from different directions when its light source, the sun, is considered to be static. Because the time cost of the second pass is linearly proportional to the number of particles and largely depends on the quality of GPU, graphic researchers are endeavoring to accelerate the calculation of the first pass. Both the hardware acceleration method introduced in this section and the recording matrix method in the next section are fast algorithms which efficiently complete the calculation for the first pass and result in photorealistic images after the second pass.

The hardware acceleration method was first introduced to render clouds by Dobashi *et al.* [DKY*00]. They represented internal particles of clouds as metaballs, and implemented an isotropic single scattering approximation to calculate the light intensities on metaballs before rendering final images. Metaballs are spheres with different radius. When rays pass through a metaball, their light intensities decrease according to the distances they travel inside the metaball. Hence, a Gaussian textured billboard was used to represent the light intensities after rays originally with the same light intensity pass through a metaball, shown in Fig. 5.12.

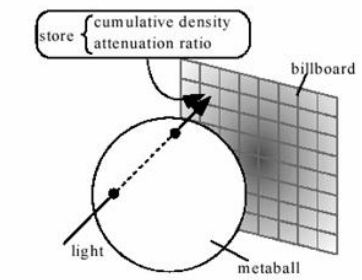


Figure 5.12: A metaball and its corresponding billboard [DKY*00].

To accelerate the rendering process, Dobashi *et al.* rendered all metaballs with Gaussian textured billboards. Therefore, light intensities on a metaball are approximated by calculating the intensity of light reaching the center of the metaball with the isotropic single scattering approximation method, which only takes light absorption inside metaballs into consideration. To record light intensities along rays after they pass through metaballs, a color frame buffer in the GPU is utilized as a temporary memory, where the calculation of light extinctions can be executed by alpha blending capability of the GPU.

Fig. 5.13 illustrates the algorithm of calculating the intensities of light reaching the centers of metaballs. As shown in Fig. 5.13 (a), a color frame buffer is initialized to be the color of a light source, the sun in this example. Billboards are placed through the centers of metaballs and are sorted based on their distances from the sun. In Fig. 5.13 (b), from far to near, the billboards are projected on to the image plane. The colors in the frame buffer are multiplied by extinction factors stored in billboard textures. The light intensity at the center of a metaball can be read out from the frame buffer after its projection. In Fig. 5.13(c), after the projection of all the billboards, the image stored in the frame buffer can be used as a shadow texture.

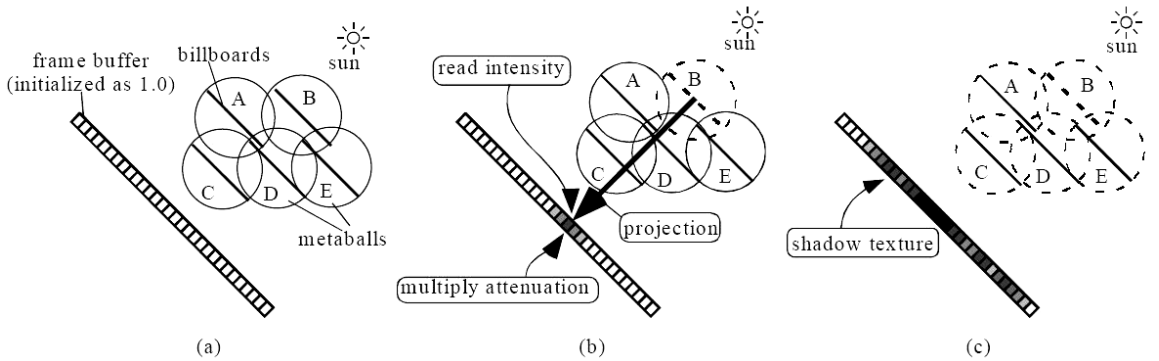


Figure 5.13: Calculation of the light intensities at the centers of metaballs [DKY*00].

After the calculation of light intensities for all metaballs, their colors are obtained by multiplying their light intensities by the color of light source. Then the second pass is applied to generate final images. All the objects other than clouds are first rendered and stored in a frame buffer. Then Billboards of clouds are oriented to the view point and sorted based on their distances from the view point. Images are generated by projecting and blending again all billboards from the one further away from the view point to the one closest to the view point in the frame buffer. That is, the new image in the frame buffer represents the sum of the old image in the frame buffer multiplied by the attenuation ratio of a billboard texture and the incoming billboard texture. Final images can be obtained by repeating this process for all billboards.

To take the multiple scattering propertys of clouds' internal particles into consideration, Harris and Lastra [HL01] developed a multiple forward scattering method for the first pass, which leads to brighter images than the isotropic single scattering approximation method. Direct calculation of multiple scattering inside clouds is very complicated and time-consuming. Fortunately, Nishita *et al.* [NDN96] found that the first and second order scattering have dominant effects in multiple scattering. To fully make use of GPU and simplify the calculation, Harris and Lastra further approximated multiple scattering to multiple forward scattering (in the light direction) and anisotropic single scattering in the eye direction [HL01].

Multiple forward scattering approximates light passing through clouds and scattered forward by clouds' internal particles. If we consider rays from direction $\vec{w}$ starting from outside and traversing a cloud constructed by metaballs, the intensity I_k on any metaball p_k is equal to the intensity of light scattered to p_k from p_{k-1} plus the intensity transmitted through p_{k-1} (as determined by its transparency). Here p_{k-1} is closer to the light source than p_k along the light direction. According to this approximation, Harris and Lastra [HL01] developed equation 5.7 for the calculation of I_k:

$$I_k = \begin{cases} I_{k-1} \cdot \left(a_k \cdot \tau_k \cdot p(\vec{w}, -\vec{w}) \cdot \beta / 4\pi + e^{-\tau_k} \right), 2 \le k \le N \\ I_0, k = 1 \end{cases} \tag{5.7}$$

where I_0 is the intensity of incoming light on the first metaball in the queue sorted by the rule introduced in the above isotropic single scattering approximation method, β is a small solid forward scattering angle and assumed to be constant, a_k is a constant albedo and τ_k is a constant extinction coefficient for all the metaballs, and $r(\vec{w}, -\vec{w})$ is the phase function, which determines the extent of light from direction $-\vec{w}$ scattered to direction $\vec{w}$. In their work, the simple Rayleigh scattering phase function is used:

$$r(\theta) = 3/4\left(1 + \cos^2\theta\right) \tag{5.8}$$

where θ is the angle between the incoming and scattered light directions.

Multiple forward scattering is easy to be implemented with a common GPU through a popular API like OpenGL or Direct3D, since the blending functions of these APIs allow accumulating incoming texture colors after scaled by a parameter with existing colors in the destination frame buffer after scaled by another parameter, and saving the accumulation result in the GPU. If we take a look at the color format used for the blending purpose, we can find that there is another channel other than Red (R), Green (G) and Blue (B), called Alpha (A). Alpha channel controls the degree of transparence of both the source color and the destination color by functioning as a scale parameter, which can be used to scale the source color, or the destination color, or both in the frame buffer, depending on the choice of the user.

With the blending capability of the GPU, I_k can be easily calculated according to equation 5.7 by setting $I_{k-1} \cdot a_k \cdot \tau_k \cdot r(\vec{w},-\vec{w}) \cdot \beta / 4\pi$ as the source color for each billboard with scale parameter 1.0 and setting $e^{-\tau_k}$ as the scale parameter of the destination color, which equals to I_{k-1} in this equation. Therefore, in order to set the color of every billboard, I_{k-1} needs to be read out from the frame buffer. Here, the small solid angle is to determine the area which scatters light to p_k. In other words, it determines how many pixels in the frame buffer need to be read out. In order to accelerate the computation, Harris and Lastra assumed that only a tiny difference between the light intensities before and after rays traversing a metaball [HL01]. Then the average of the read out values is used to approximate I_{k-1} in equation 5.7 and also saved as the color of the current billboard, namely I_k. As we introduced before, billboards are textured with a Gaussian function to approximate the absorption of rays traversing metaballs. With OpenGL, a texture mode called modulate can make the color of each billboard to be scaled first by its texture, before it is blended with the image already in the frame buffer. The first pass is done after all billboards are projected and blended in a distance-ascending order to the light source.

Besides the approximation of incident light multiple forward scattering inside clouds, single scattering towards the viewer was also implemented to readjust the color of metaballs in the final image by Harris and Lastra. Assuming l is the view direction; the light E_k transmitted by metaball p_k to the view point is computed similarly as I_k above:

$$E_k = a_k \cdot \tau_k \cdot r(l,-\vec{w}) \cdot I_k / 4\pi + e^{-\tau_k} \cdot E_{k-1}, 1 \le k \le N \tag{5.9}$$

In equation 5.9, the first item presents the light scattered from a metaball to the view point and the second item approximates the amount of light towards the view point after passing through the metaball.

The idea of making use of GPU rose from the fast development of GPU capabilities and the slow development of CPU in recent years. Nowadays, one GPU has several processors which can process data in parallel. Therefore, taking the advantage of the computational capabilities of GPUs now becomes very popular in the computer graphic and image processing areas. However, we must notice that for the cloud rendering, color values need to be continuously read out from a GPU and transmitted to a CPU for every metaball. Because the comparatively slow speeds of data transfer between GPUs and CPUs, the computation of the first pass is slowed down, since the number of metaballs for a cloud is easily beyond one hundred thousands. Hence, in order to render clouds or smoke more efficiently, a new method called recording matrix is developed by this PhD work. It

largely accelerates the calculation of light intensities on metaballs caused by light sources and inner light multiple scattering. The recording matrix method is very easy to implement and has a flexible acceleration parameter. Though it achieves ten times faster speed than hardware acceleration method developed by Harris and Lastra, it can photo-realistically display self-shading properties of half transparent gaseous phenomena as well.

5.2.2 Recording Matrix

All the previously introduced methods obtained promising results with various computation costs. Even in the work of Dobashi and Harris *et al.* [DKY*00, HL01], where the first pass rendering process profits a lot from hardware acceleration techniques, the illumination phase still need seconds for a cloud containing hundreds of thousands metaballs. For static clouds and static light sources, these methods are acceptable, since the light intensities on metaballs only need to be calculated once. Then real-time rendering can be achieved by applying dynamic impostors [HL01]. However, for animated clouds or dynamic light sources, a faster method is desirable to achieve higher frame rate, since the recalculation of light intensities is unavoidable to display the self-shading properties of clouds after every movement of metaballs. To present the efficient rendering method developed in this thesis [WAN06], this section starts from the basic illumination model of clouds.

Fig. 5.14 shows the basic idea of applying the two pass method to render a cloud in the sunshine. In the first pass, the dominant sunlight and the ambient light, with light intensities I_{S1} and I_{S2} respectively, transmit through the cloud with multiple scattering among metaballs. The ambient light is composed of sky light and the light reflected from other clouds and the ground, which should be taken into account according to the research of Nishita *et al.* [NDN96]. This process results in various light intensities on metaballs. Then the light scattered by metaballs passes through the atmosphere and arrives at the view point with intensities I_C , which are recognized as a cloud. Hence, the calculation of light attenuation and multiple scattering inside clouds is very important, because it determines the brightness of a cloud image to a large extent.

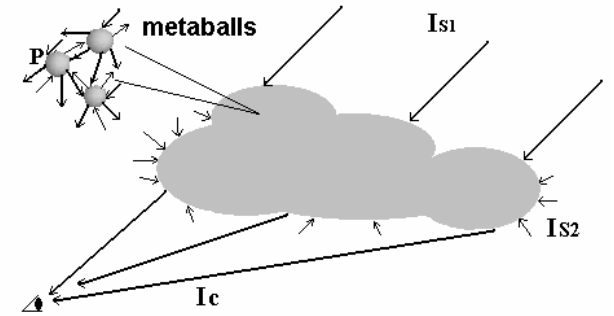


Figure 5.14: Light scattered to the view point by a cloud.

While considering the average intensity, I_p, on a metaball p is equal to the average intensity of incident light after traversing intermediates between the light source and p in direction w (pointing from the light source to p), $I_{in}(p,w)$, plus the average intensity of light scattered to p by other metaballs; namely:

$$I_p = I_{in}(p,w) + \oint_v I_s(p,v)dv \tag{5.10}$$

where $I_s(p,v)$ is the intensity of light scattered to p in direction v. When the incident light in direction w traverses p, some part of it is absorbed and scattered to other directions. After it goes out of p, its intensity, $I_{out}(p,w)$, changes to:

$$I_{out}(p,w) = I_{in}(p,w) \cdot (1.0 - e_p - \alpha_p) + I_{in}(p,w) \cdot \alpha_p \cdot r(w,w) + \oint_v I_s(p,v) \cdot \alpha_p \cdot r(v,w) dv \quad (5.11)$$

where e_p is the extinction coefficient of p and α_p is its albedo; $r(v,w)$ is the phase function, which determines the percentage of light scattered from the direction v to the direction w.

Suppose the incident light reaches metaball k in succession. Then, $I_{out}(p,w)$ is also $I_{in}(k,w)$. Therefore, along the direction of incident light, the average intensity on any metaball can be calculated successively according to equation 5.10 and 5.11. Herein, average intensity of incident light on the first metaball is the intensity of the light source. However, it is complicated to fully calculate multiple scattering inside a cloud according to equation 5.10 and 5.11 directly, because each cloud is composed of a lot of metaballs and light is repeatedly scattered among them back and forth.

Fortunately, the work of Nishita *et al.* [NDN96] demonstrated that the contribution of multiple scattering is dominated by the first two orders. Furthermore, as shown in equation 5.11, the phase function plays an important role in the calculation of multiple scattering. Hu *et al.* found that the phase function for water clouds generates high values along forward directions within a narrow solid angle less than 15 degree, and then values sharply decrease according to the angle between incident and scattered light. In addition, for icy clouds, the phase function only has a small backscattering peak, compared to the forward scattering [HWL*00]. Hence, in order to accelerate the rendering process, the multiple scattering of the dominant light is simplified to multiple forward scattering and the multiple scattering of the ambient light is simplified to multiple supplementary forward scattering. Herein, scattering light on a metaball is only resulted from those metaballs which are closer to the light source along the direction of incident light and within the narrow solid angle. In addition, we must notice that the ambient light is a general name for light having much weaker intensities than the dominant light. For example, when the sky light becomes comparatively strong, it must be considered as another light source and multiple forward scattering is calculated for it independently.

To further simplify the illumination model, we assume the varying albedo α_p to be a constant α and the extinction coefficient e_p to be a constant e for all the metaballs. In addition, since the forward scattering occurs within a narrow solid angle, the values of the phase function are assumed to be a constant r_1 for the incident light $p(w,w)$ and r_2 for the scattering light $p(v,w)$. Accordingly,

$$I_p = I_{in}(p,w) + \int_{-\beta}^{\beta} I_s(p,v) \cdot dv \quad (5.12)$$

$$I_{out}(p,w) = I_{in}(p,w) \cdot (1.0 - e - \alpha) + I_{in}(p,w) \cdot \alpha \cdot r_1 + \int_{-\beta}^{\beta} I_s(p,v) \cdot \alpha \cdot r_2 \cdot dv \quad (5.13)$$

where β is the forward scattering angle. Fig. 5.15(a) shows the basic idea of incident light along the direction w passing through metaball p and light scattered on and from p.

Multiple forward scattering can achieve photorealistic results in some cases. However, if we consider a thick cloud in the sunshine, multiple forward scattering may result in very small light intensities on the side far from the sun, because of the accumulation of extinction in the forward direction. However, it is unrealistic and artificial since the skylight also contributes to the illumination. Therefore, the ambient light should be taken into account as a supplement.

Since the ambient light is coming from many sources, it is hard to simulate it exactly. Hence, we assume that the ambient light has isotropic distribution surrounding a cloud. Furthermore, its contribution on the same side of the dominant light source to the cloud has been taken into account as part of the dominant light. Since the dominant light is much stronger than the ambient light and is most attenuated along its forward direction, we only calculate the effect of the ambient light along the

backward direction of the dominant light as a supplement. This also helps to avoid the time cost of rearranging the calculation order of metaballs.

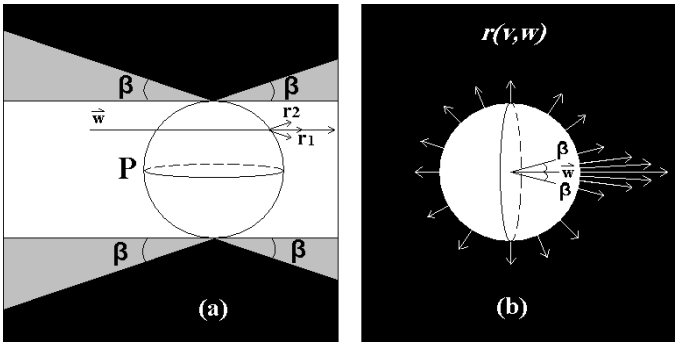


Figure 5.15: (a) Incident light in white along the direction *w* passing through metaball *p* and light scattered on and from *p* in grey; (b) values of the phase function for any metaball in the second pass (light scattered to the view point).

Because the average light intensity on a metaball is related to its color in the second rendering pass, we simply assume that a light source with a weaker intensity can not enhance the intensity on an object having a stronger intensity. This assumption prevents the intensities on metaballs from becoming unreasonably strong, namely the colors of metaballs all turning into purely white, after considering the effect of the ambient light. This assumption precludes the elimination of the self-shading properties of clouds. Hence, the ambient light can transmit among metaballs which have smaller intensities. The supplementary intensity to a metaball is obtained by calculating the multiple forward scattering of the differences between the intensity of the ambient light and the intensities already on the metaballs resulting from the dominant light. In addition, when the ambient light is colorful, the differences of intensities should be counted in red, green and blue colors respectively. For the same reason, if there is another light source, we only need to sum up the contribution caused by multiple forward scattering of the light with the intensity difference again to clamp the brightness of clouds.

As we introduced in the last section, the graphics hardware technique accelerates the illumination phase by considering ray clusters through metaballs. However, the most essential step in the hardware acceleration method is to read pixels back from a frame buffer in GPU [HL01], which is the bottleneck of speedup. Profiting from the idea of processing a segmental ray cluster through each metaball, this PhD work records light intensity changes along ray clusters into a 'matrix', namely an array, and this matrix is set to be perpendicular to the incident light direction, shown in Fig. 5.16.

Since we only simulate the light transmitting among metaballs in the illumination phase, a matrix is set to fix the physical boundaries of every independent cloud. To be understood more easily, a matrix can be viewed as an image perpendicularly to the incident light and fully containing the projection of a cloud. As shown in Fig. 5.16, each pixel in this image represents an element *V(i,j)* of the matrix. Setting the resolution of the image to be the same as the computer screen, we can calculate light intensities on metaballs with enough resolution by storing intensity changes along the direction of incident light, through the whole cloud in the recording matrix. It is important to mention that there is no need to store the intensities on metaballs in red, green, and blue respectively, because they change in a uniform ratio. A colorful cloud can be generated by multiplying the color of the incident light and the calculated average intensities on metaballs.

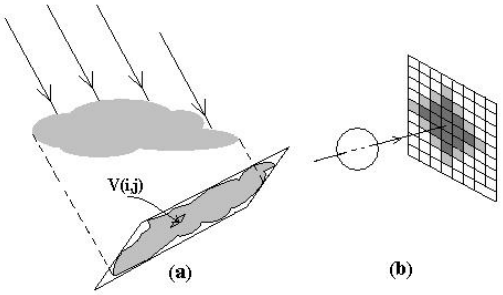


Figure 5.16: A recording matrix and related elements for calculating the average intensity on a metaball.

Before the computation of the first pass (illumination phase), all elements in the recording matrix are set to be the intensity of the incident light. Then the computation is performed for each metaball in distance ascending order to the light source. The first step when calculating the intensity on a metaball p is to project p on the recording matrix like the one in Fig. 5.16(b). The projection occupies some dark gray elements shown in the figure. These elements record the intensities which are in front of p along the incident light direction. Their average is $I_{in}(p,w)$ in equation 5.12 and 5.13. In addition, the light grey elements in this figure are intensities which contribute to the scattering part. In other words, they are integral items in equation 5.13. In order to speed up the first pass, we assume that they are always the elements next to the projections of metaballs, since multiple forward scattering only considers the scattering within a narrow solid angle in forward directions.

The second step is to find the extinction ratio for every element in the matrix. As metaballs are spheres, they attenuate the incident light to different extents. For instance, a ray passing through the center of a metaball obviously travels longer than it passing through other parts of the metaball. Therefore, the intensities along rays through the centers of metaballs are most attenuated. A Gaussian distributions is normally used to control the transparency of metaballs [DKY*00, HL01]. It describes that the extinction ratio continuously decreases from the center to the edge on the round projection of a metaball. However, when a Gaussian distribution is applied to the projection of a metaball, which always has a small size, it is heavily discretized. Hence, to avoid the expensive exponential computation of the Gaussian distribution, we only take the exponent as ratio. The product of the ratio and extinction coefficient is the extinction ratio e in equation 5.13. Then the intensity on p is computed. The matrix is updated to record the new intensities in the dark grey elements and their values are averaged to be the color of p. Therefore, after the calculation orderly going through all the metaballs in the sorted queue, the contribution of the dominant light has been calculated. The matrix finally records the intensities on the metaballs that lie on the cloud surface and on the far side of the incident light source.

Since we assume that the ambient light only transmits in places that have lower intensities along the backward direction of the dominant light, the contribution of the ambient light is calculated by multiple forward scattering the differences between the intensities of the ambient light and the values in the recording matrix. Hence, the matrix is updated to store the differences and multiple forward scattering is performed again through all the metaballs in the inverse order. The final intensity on a metaball is the sum of the intensities resulting from the dominant light and the ambient light.

Suppose that two clouds are identical and in the same illumination environment except for their sizes, then the incident light undergoes similar scattering processes among metaballs when transmitting through these clouds. Hence, this PhD work shrinks cloud models by a selectable scaling factor to further accelerate the first pass. For example, for a cloud constructed by 20000 metaballs, which have radiuses of around 10 pixels on the computer screen, a scaling factor of 0.5 causes a hardly distinguishable difference in the final image as compared to the image rendered without the scaling factor, but with a speed increase of five times.

After obtaining the average intensity (color) of every metaball, it is ready to perform the second pass to render cloud images. The second pass is to compute the intensities of light which is scattered by metaballs and transmits through clouds and the atmosphere towards the view point. If we ignore the scattering and extinction of the atmosphere, when light on metaballs of a cloud is scattered to view points at different positions, various reflections are formed. Suppose the direction pointing from p to the view point is l. According to our illumination model, the intensity of light in the direction l after passing through p, called E_p, is determined by both the incident light and the scattering light on p. Here, the scattering direction is not limited to the forward solid angle any more to make reasonable visual effects in a view from any direction.

According to equation 5.11, the phase function, $r(v,w)$, which is determined by the property of medium, plays an important role in the light scattering process. Nishita et al. used a refined Henyey-Greenstein function as their phase function in cloud rendering [NDN96]. Its computation is so time-consuming that Harris and Lastra replaced it with a simple Rayleigh phase function later on [HL01]. However, the simple Rayleigh phase function is a symmetric function which has the same value in the forward and backward directions. In another word, if the user looks at a cloud no matter from the bottom or top, he can observe a shining sun behind the cloud. To overcome this short-coming and also keep the simplicity for the purpose of fast rendering, we suppose that the phase function has quickly decreasing values within a narrow forward angle and the same value elsewhere, as shown in Fig. 15(b). Our phase function simply is

$$r(\theta(v,w)) = \begin{cases} c_1(1-\sqrt{1-\cos 4\theta}); \theta \in [-\beta, \beta] \\ c_2; \theta \in (\beta, 2\pi - \beta) \end{cases} \tag{5.14}$$

where θ is the angle between v and w, and β is less than $\pi/8$. In addition, according to the definition of $r(\theta(v,w))$, we have

$$\oint r(\theta) d\theta = 1 \tag{5.15}$$

$$c_2 = c_1(1-\sqrt{1-\cos 4\beta}) \tag{5.16}$$

Therefore, $r(\theta)$ can be uniquely defined by a selected β. We use $\pi/12$ as β in this paper and our results prove that this simple phase function is practical for the cloud rendering.

Consequently, E_p results from light in the direction l after passing through p as well as the light scattered by p to the direction l. There are two possibilities for E_p: E_p is simply equal to $I_{out}(p,w)$; or according to equation 5.11,

$$E_p = I_s(p,l) \cdot (1.0 - e - \alpha) + \oint I_s(p,v) \cdot \alpha \cdot r(v,l) dv + I_{in}(p,w) \cdot \alpha \cdot r(w,l). \tag{5.17}$$

In the former case, l is the same as w. Therefore, $I_{out}(p,w)$ can be directly used to represent the intensity along l. To simplify the calculation of E_p, we use I_p along w to approximate the light on p which scatters light to l. Then

$$E_p = I_{in}(p,w) \cdot (1.0 - e - \alpha) + I_p \cdot \alpha \cdot r(w,l). \tag{5.18}$$

Similarly, equation 5.17 changes to

$$E_p = I_s(p,l) \cdot (1.0 - e - \alpha) + I_p \cdot \alpha \cdot r(w,l). \tag{5.19}$$

In addition, when the light along l after passing through p encounters another metaball q in succession, E_p is equal to $I_s(q,l)$ in the calculation of E_p. The only exception is the calculation of those metaballs, in front of which light doesn't encounter any other metaballs along l. The approximation of forward scattering is applied again to simplify the calculation. Consequently, light along l through those metaballs are only determined by their own scattering.

This pass is to generate fuzzy images with calculated intensities according to equation 5.18 and 5.19. Clouds are rendered upon a background by continuously blending billboards in a distance decreasing order towards the view point. The color of each billboard is the maximum scattering light intensity of a metaball in a model if it is illuminated by multiple light sources. Final images are generated after all the metaballs are blended upon the background. Because of the extinction coefficient and various intensities on metaballs, clouds appear distinct half-transparent with self-shading properties.

This section presented a new method called recording matrix to accelerate the illumination phase. This method is drawn from the idea of the graphics hardware acceleration method developed by Harris and Lastra [HL01]. However, it achieves similarly visually convincing results with ten time's faster speed comparing to the hardware acceleration method. A user-controllable parameter can further reduce the computing time. This method is easy to implement and is independent of graphics hardware. Moreover, it only needs a small temporary saving space during the computation of the first pass. Accompanying with the fast development of high speed computational processors and dynamic simulation algorithms, this method may allow to render dynamic clouds and other animated gaseous phenomena in real-time using a normal PC in the near future.

5.3 Dynamic Impostor

Though the cloud rendering methods described above are fast and efficient, real-time rendering can hardly be achieved with clouds in virtual landscape because of the large amount of metaballs used to construct clouds. Dynamic impostor is a technique which is designed for real-time cloud rendering and can achieve this goal under some conditions. An impostor is a semi-transparent quadrangle textured with an image of the object which is replaced by the impostor in the virtual world. Impostors have two categories. One is to compute images for an object from multiple viewpoints in advance [SLS*96]. It requires large storage space in the running time if unnoticeable substitutions are wanted. The other is to generate impostors dynamically when they are zoomed beyond a certain tolerance or viewed from another direction [HL01]. It is easy to find out that though the dynamic impostor technique requires more rendering time for the generation of a new impostor, it is more flexible, continuous and does not require extra storages.

During the implementation of dynamic impostor, the position of an object in a view volume, which will be replaced by an impostor, and the position of its viewer need to be found out first. The view point is the point from which an impostor will be viewed. It must be in the same coordinate system as the impostor. To achieve realistic results, impostors must always face their view point because they are only 2D polygons. In other words, they are always parallel to the near and far clipping planes in a view volume, and are perpendicular to the line through the view point and its projections on the clipping planes. On the contrary, if they are parallel to the planes, they are just lines in rendered images. Fig. 5.17 illustrates a cloud represented by impostors in a view frustum with two different view points.

Whenever the view point changes position beyond a certain threshold, it is necessary to dynamically render the represented images of impostors again and transform them to face the new view point. As we introduced in the last section, billboards textured with a Gaussian function are used to display cloud images. Herein, billboards are all 2D polygons. After all the billboards for a cloud are blended in a frame buffer according to equation 5.9, the resultant image is read out and used to texture an

impostor lying on the x-y plane and centering at the origin. The next step is to rotate the impostor to face the view point. As shown in Fig. 5.18, because the current normal direction of the impostor (in black) is along z-axis, having direction $\vec{z}$, rotation axis $\vec{r}$ is then perpendicular to both $\vec{z}$ and new normal direction $\vec{n}$. Hence, $\vec{r}$ is the normalized cross product of $\vec{z}$ and $\vec{n}$. Fig. 5.18 (a) shows two impostors before and after rotation around $\vec{r}$. The impostor is originally in the place of the black polygon and then it is rotated to the place of the red polygon with normal $\vec{n}$.

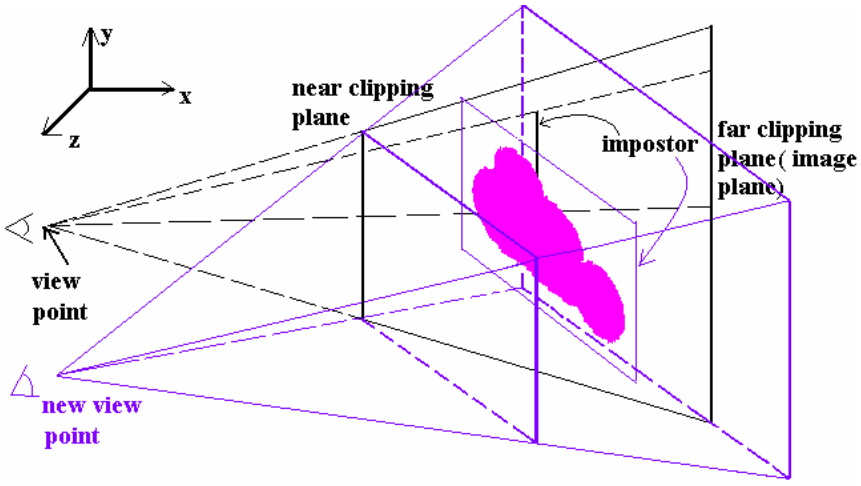
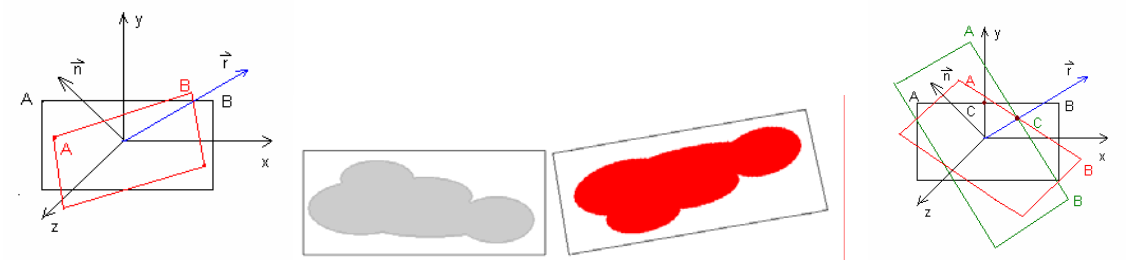


Figure 5.17: Impostors in a view frustum with two different view points.



(a) (b) (c)

Figure 5.18: (a) Impostors before and after rotation around $\vec{r}$. The impostor in the place of the black polygon is rotated to the place of the red one, having the new normal $\vec{n}$. (b) Impostors with the normal $\vec{n}$. The grey one is in a proper place but the red one is not. (c) A complete rotation procedure. The impostor in the place of the back polygon is first rotated to the place of the green one and then rotated to the place of the red one.

It is easy to figure out that the impostor may look upside-down after the above rotation. For example, in Fig. 5.18 (b) the grey impostor is in a proper position, but the rotation may make the impostor like the improper red image. To solve this problem, another rotation process is necessary to adjust the impostor's location before it is rotated around $\vec{r}$. Because $\vec{r}$ is perpendicular to $\vec{z}$, it is always on the x-y plane. Therefore, there are two intersections between the line through the origin along $\vec{r}$ and the edges of the impostor. The positions of the two intersections stay unchanged during rotation. Hence, before the impostor is rotated around $\vec{r}$, it is first rotated around $\vec{z}$ to let $\vec{r}$ be perpendicular to $\overline{AB}$. The green impostor in Fig. 5.18 (c) shows the intermediate position after this adjustment. Then the green impostor can be rotated around $\vec{r}$ and translated to the position of its representing 3D object in the view frustum. It is more understandable when we think about 3D objects shown on computer

screens. They are all 2D projections and we still have 3D impressions. The impostor technique just utilizes this property before a computer displays images on its screen.

5.4 Discussion

This chapter introduced some efficient cloud visualization methods applied by graphics researchers recently on the cloud modeling and rendering. Because of the amorphous appearances and self-shading properties of atmospheric clouds, fast and visually realistic cloud visualization is still a challenge. The two-level marching cube algorithm for the cloud modeling and the developed recording matrix method for the cloud rendering are developed in this PhD thesis and can visualize photorealistic static or dynamic clouds fast and efficiently.

Two efficient cloud modeling methods, the procedural modeling and the marching cube modeling, are introduced in this chapter. The procedural modeling method utilizes a two-level model. In this method, the macrostructure of a virtual cloud is defined by implicit functions because of their smooth blending properties. Procedural turbulence and noise functions are used to create the cloud microstructure by generating a small turbulent density field. Then it is combined with the density field defined by the cloud macrostructure to determine the final density distribution for an amorphous cloud model. Consequently, cloud models created by this method have very good random and turbulent properties.

However, cloud models created by the procedural modeling method have too smooth densities, which can hardly generate bumpy surfaces for cumulus clouds. The marching cube modeling method is an efficient method to make up this shortage. This method only needs several primitives to construct the macrostructure for a cumulous cloud model. Small particles are accumulatively placed surrounding the surfaces of the primitives by the marching cube algorithm. This method creates bumpy surfaces for cumulus clouds when the cell sizes of grids which fully contain cloud volumes are smaller than the size of the biggest particle. Since the property of the marching cube algorithm is to place particles around the cloud macrostructure, cloud models directly created by it keep the simple shapes of their macrostructures strictly. The two-level marching cube modeling method developed in this PhD thesis overcomes this shortcoming. The first level is to apply the marching cube algorithm on a coarse grid to obtain initial vertices and the second level is to recursively place particles and regenerate new replacing polygons according to continuously placed particles in a fine grid. This modeling method roughly keeps the basic shapes of cloud macrostructures, locates particles accumulatively around initially marked vertices, and has two controllable cell sizes to create different cloud models with the same primitives. The scope of cell sizes is considerably larger than the scope of particle sizes. Some large cube sizes for the second level can create randomly beautiful sparse clouds. Fig. 5.19 shows several clouds in virtual terrain. These clouds are modeled by the two-level marching cube method and rendered by the recording matrix method developed in this thesis.

For the display of clouds' amorphous appearances, the volume rendering methods from section 5.2.1.1 to 5.2.1.4, the hardware acceleration method in section 5.2.1.6, and the recording matrix method in section 5.2.2 are presented in this chapter. Those methods can render photorealistic cloud images with various time costs. If we divide them again according to rendering steps, the volume rendering methods are one pass methods, and the other two methods are two pass methods. One pass methods calculate the light intensities of rays or ray clusters influenced by cloud volumes and other intermediates before they reach the view point with software or hardware acceleration approaches. Two pass methods first calculate the light intensities on internal particles of clouds by tracing along ray clusters which are repeatedly scattered inside clouds; and then calculate the intensities of light scattered by those particles through other intermediates towards the view point. Comparing to one pass methods, two pass methods are more flexible for cloud visualization. As long as light resources and clouds keep static, the first pass only needs to be calculated once wherever the view point is. Moreover, if the view direction is not parallel to incident light directions, the computation of one pass

methods is very complicated because it is necessary to consider both the forward scattering and the scattering towards the view point.

Another special cloud rendering method is the pure hardware method, 3D texture mapping. If the color values of a 3D texture which represents a cloud have already been calculated and keep constant in the virtual world, the 3D texture mapping method is the fastest manner to rendering clouds, which automatically projects and interpolates color values within GPUs. However, in most interactive cases, color values change continuously in the animation, for example dynamic clouds simulated with random interfering factors and dense smoke emitted from a chimney.

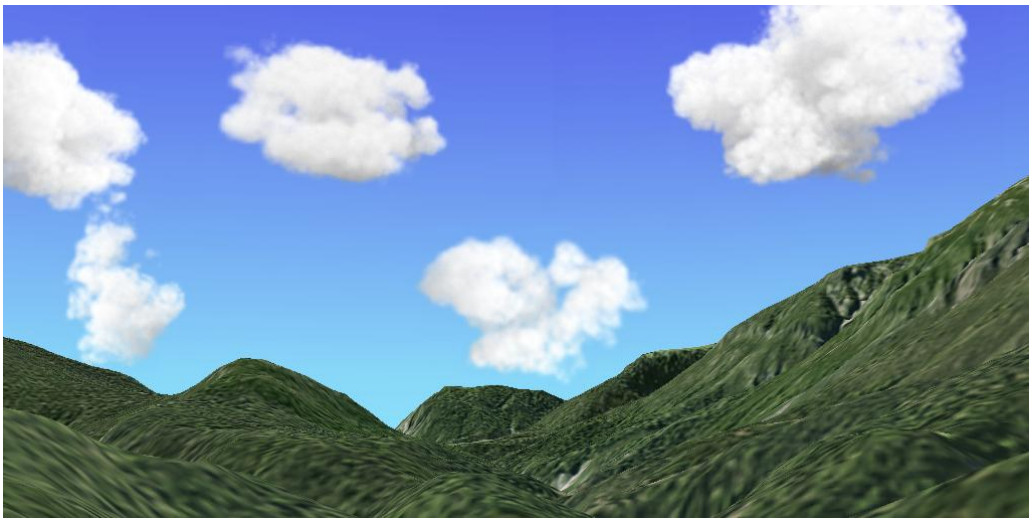


Figure 5.19: Clouds illuminated by the sunlight in the landscape. Cloud models are created by the two-level marching cube modeling method and rendered by the recording matrix method. Both methods are developed in this PhD thesis.

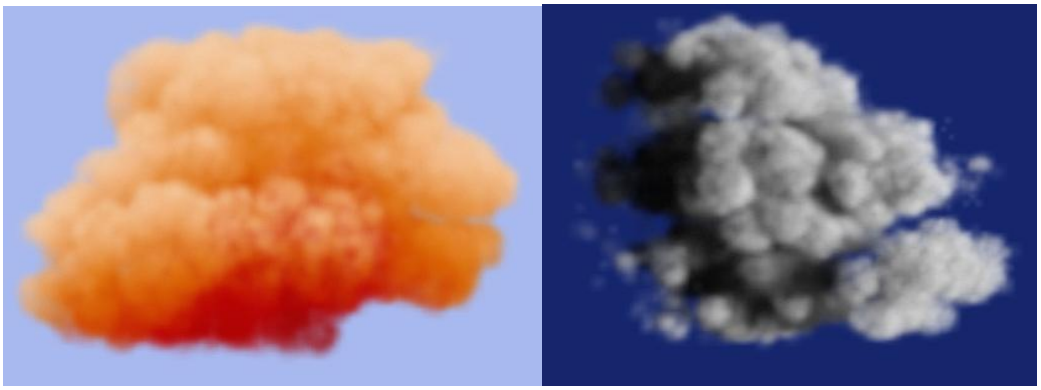
The recording matrix method efficiently simulates the multiple forward scattering of light among the internal particles of clouds. Though multiple scattering is approximated by multiple forward scattering, it produces photorealistic self-shading results with considerably lower cost. It calculates the effects of ray clusters on particles with the help of an auxiliary array for each cloud. This array is called recording matrix in this thesis, as it records the intensities of particles. The size of a matrix is not bigger than a computer screen size because it tightly binds the size of a cloud in pixels on the screen. Therefore, unlike numerous times of reading pixels from the GPU in the hardware acceleration method, the matrix method directly accesses a small array in the CPU. It is so fast that the computation of the illumination phase is largely accelerated.

The values in a recording matrix are light intensities on metaballs, which determine their colors together with the color of incident light. Hence, rendering process can be accelerated when only one float value is necessary to be recursively accessed and saved instead of 3 values (R, G, and B). Fig. 5.20 shows clouds illuminated by light in different colors and viewed from different directions. Clouds display different colors by multiplying computed light intensities with different incident light colors. In Fig. 5.20 (c), the cloud is viewed towards the sun. Because of different values generated by the phase function according to the angles between view directions and the sunlight direction, some part of the cloud bottom appears much brighter than other places. As a comparison, clouds shown in Fig. 5.21 are rendered with incident light in two colors from two different directions. The illumination effects caused by the white sunlight are calculated with only one float value. However, illumination effects resulted from the nacarat skylight have to be calculated with 3 values to determine the

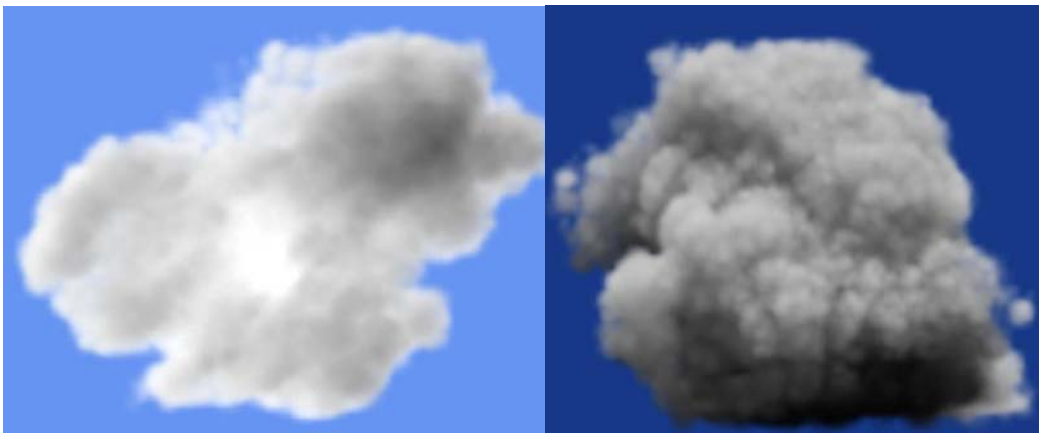
strongest intensities on each metaball in R, G, and B channels respectively. Fig. 5.22 shows the effects of a cloud illuminated by two light sources from different directions.

The values in a recording matrix also help to determine how much ambient light will transmit through clouds. The contribution of ambient light is the compensation for multiple forward scattering and the supplement for other weaker incident light. Taking it into consideration prevents generating unreasonably dark cloud images. Fig. 5.23 shows the difference between the presence and the absence of ambient light for a dense cumulus cloud. It is noticeable that the one rendered with ambient light is brighter at the bottom.

The recording matrix method has a selectable scaling factor, which results in hardly noticeable differences in final images, but can accelerate the calculation of the illumination phase several times faster. This scaling factor depends on the sizes of particles and the resolution required by the user. Generally speaking, a scaling factor which is no less than 0.5 produces reasonable final results. Fig. 5.24 (a-b) show a cloud model rendered with different scaling factors. The images appear very similar, though (a) costs much less computing time than (b). As a comparison, Fig. 5.24 (c) shows the same cloud model rendered by the hardware acceleration method without taking any ambient light into consideration.



(a) (b)



(c) (d)

Figure 5.20: (a) A cloud illuminated by a colorful skylight; (b) a cloud viewed from a direction nearly vertically to the direction of incident light; (c) a cloud viewed towards the sum; (d); a cloud viewed from a direction having a small angle with the direction of incident light.

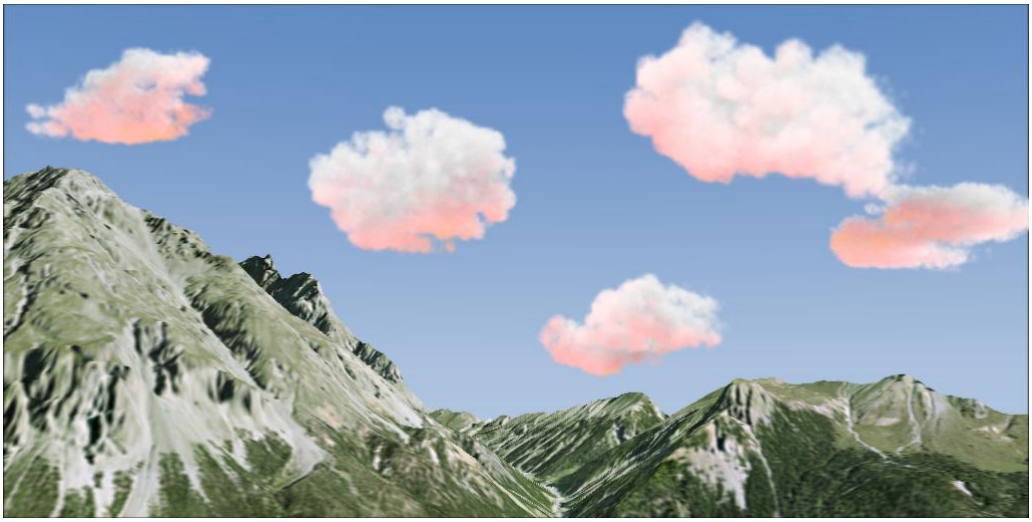


Figure 5.21: Clouds illuminated by the white sunlight and a nacarat skylight in the landscape.

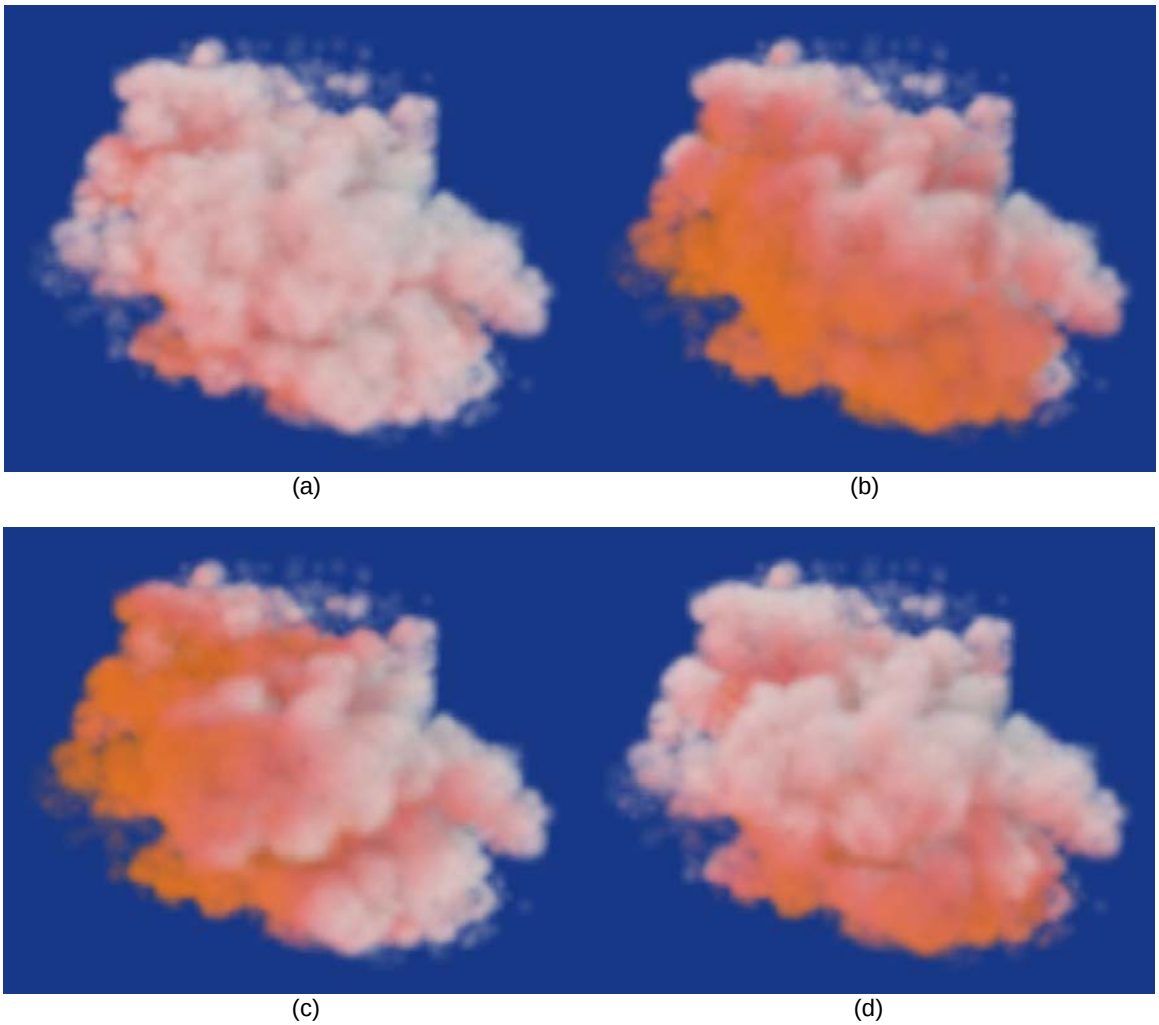
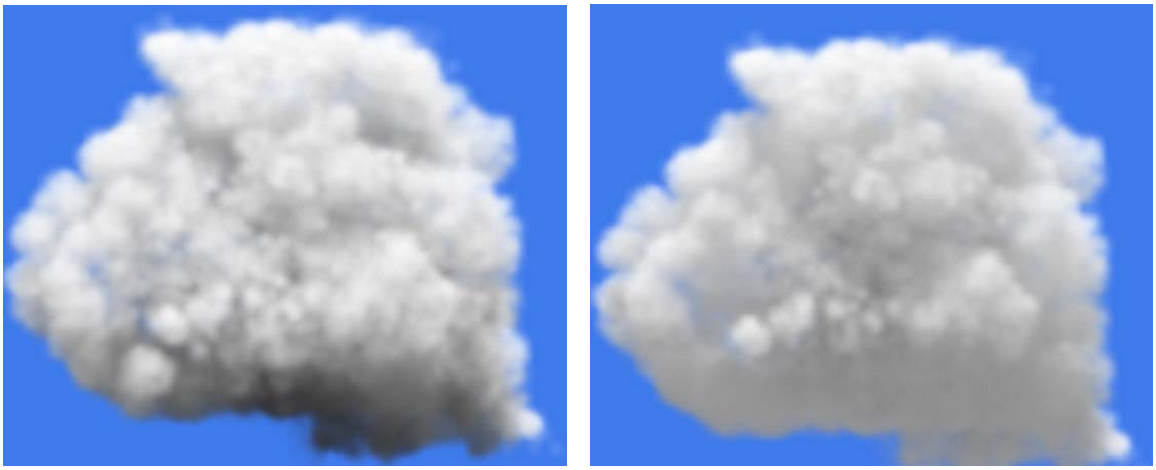
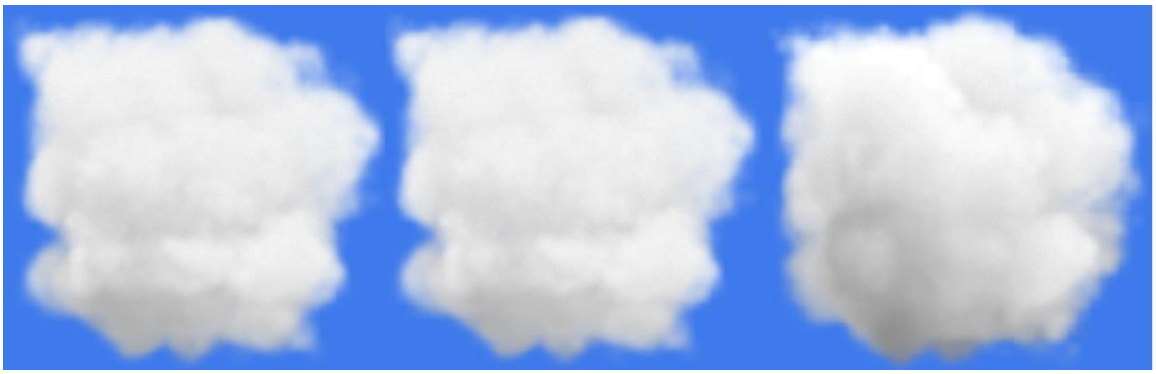


Figure 5.22: A cloud illuminated by two light sources from different directions.



(a) (b)

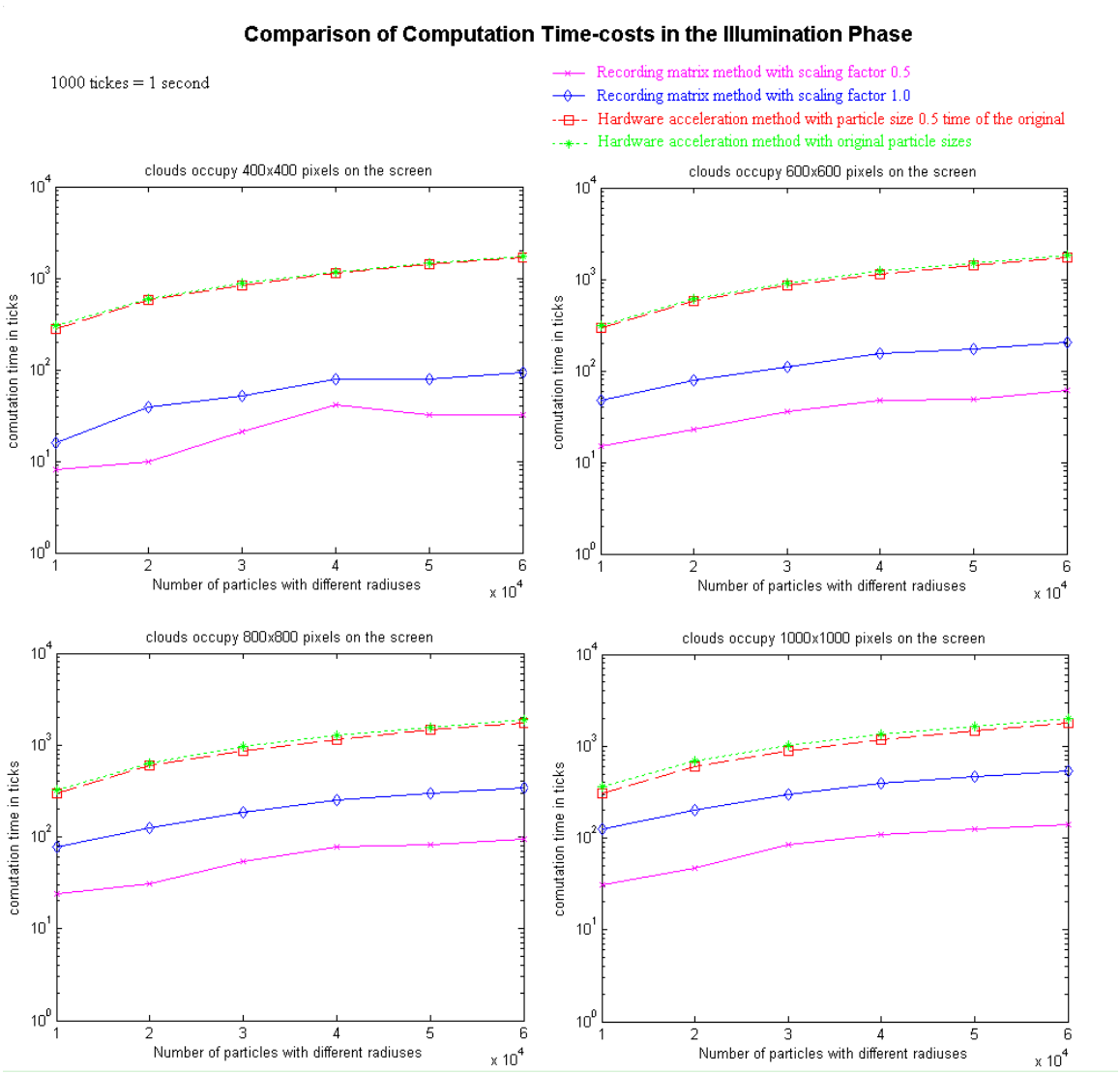
Figure 5.23: (a) A cloud rendered without the ambient light; (b) the cloud rendered with the ambient light.



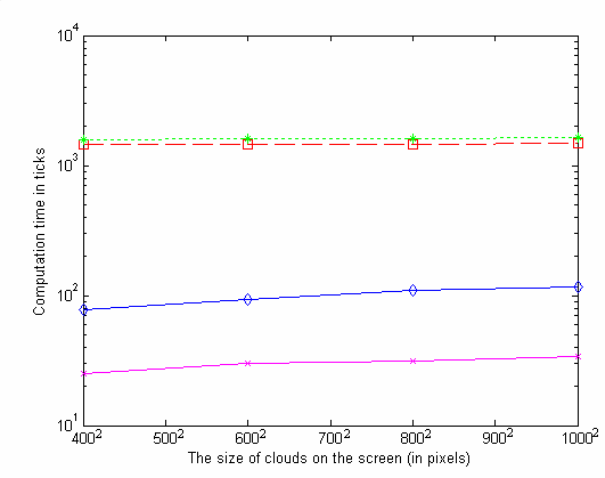
(a) (b) (c)

Figure 5.24: (a) A cloud rendered with scaling factor 0.5. (b) The cloud rendered with scaling factor 1.0 (without shrinking the model). (c) The cloud rendered with the hardware acceleration method.

The second rendering pass of the developed recording matrix method is also a simple anistropic scattering process and can be directly executed by blending Gaussian textured billboards, which take the products of the intensities on particles and incident light as their colors. This process totally depends on the capabilities of GPUs and a current common GPU nowadays can render hundred thousands of textured billboards in real-time. Hence, we only compare the computing time of the first pass between our method and the graphic hardware acceleration method developed by Harris and Lastra [HL01], which is already faster than other ray tracing methods under the condition of generating interactive photorealistic results. Fig. 5.25 displays the comparison of computation time costs between their hardware acceleration method and the matrix method. In Fig. 5.25(a), experiments are carried out on the same models prospectively projected on parallel planes in different distances. In other words, the same particle has different sizes on different planes and has the maximum size on the farthest plane. In Fig. 5.25(b), experiments are carried out on different models with 50000 particles. When the size of clouds changes, particles keep the same sizes and only their relative distances are changed in different prospective projection planes. All the data in Fig. 5.25 are obtained on a PC with an Intel Core 2 6600 2.4GHz CPU and a NVIDIA GeForce 7900 GS GPU.



(a)



(b)

Figure 5.25: The comparison of computation time costs for the first pass rendering (the illumination phase).

From Fig. 5.25 we can observe:

1. The power of GPU is parallel computation. When the sizes of particles increase along with the size of cloud models, there is only slightly influence on the computation time of the hardware acceleration method. On the contrary, it has apparent influence on the recording matrix method. However, we must notice that a too large particle size will reduce the visual quality of the final image with both methods. So only small particles are used in cloud models. Therefore, it is not a shortcoming of the recording matrix method.
2. Taking the advantage of the scaling factor, the recording matrix method can largely accelerate the computation, while changing particle sizes in the hardware acceleration method does not have any obvious effects.
3. When the particle sizes keep constant, the cloud sizes also have little influence on the recording matrix method.

Hence, we can draw the conclusion that the reason of the recording matrix method is more efficient than the hardware acceleration method is that each particle has a small size and needs to be processed individually, which makes cloud rendering is different from the normal volume rendering and makes the parallel computation strength of GPU can not be fully exert.

Chapter 6 Real-time 4D Visualization of Migratory Insect Dynamics within an Integrated Spatiotemporal System

6.1 Introduction

Migration is a fundamental component of many insect life cycles. Most migratory insect species can cause severe damages in their occupied geographic areas where the temperature, moisture or food is favorable. As an example, larch bud moths (LBM) are often capable of migrating long distances and causing conspicuous defoliation on Alpine Larches (*Larix decidua*) across extensive areas [BF88]. During peaks of population cycles, the over-crowding of LBM larvae causes interrupted feeding behavior, which leads to partially eaten needles drying out, such that large-scale defoliation results during summer months. These consequences directly impact on tourism and forest succession [BF88, BR99], in addition to other related effects in the local ecological chain. Knowledge of the migration dynamics of such insects is therefore of value within ecology and forest management.

Nowadays ecologists are capable of modelling and analyzing dynamics of migratory insects within efficient simulation systems [Fis82, FB79]. For example a LBM dynamic model is implemented within an interactive modeling and simulation environment known as RAMSES (Research Aids for Modeling and Simulation of Environmental Systems) [Fis91]. The model, known as LBM-M9, couples a local dynamics model under a food quality hypothesis with a migration model describing migrations within the Upper Engadine valley [Fis82, Fis83]. The food quality hypothesis states that defoliation results in an increase in raw fibre content of larch needles during a larch bud moth outbreak and the raw fibre content has a strong negative effect on larval survival and female fecundity [BF88]. The migration part of the model simulates migration of female LBM depending on wind conditions, degree of defoliation, number of larch trees and other specific geographical parameters.

However, current systems do not allow for the analysis and display of simulation results in a visual manner that may be easily interpreted by non-experts. As an example, Fig. 6.1 and table 6.1 show two results from a LBM simulation model run. To be homogeneous with respect to aspect and altitude for the purposes of formulating the migration model, the Upper Engadine valley was divided into 20 areas, called sites [Fis82].The 2D illustration in Fig. 6.1 describes the temporal variations of LBM larval density for the entire Upper Engadine valley aggregated to a single spatial unit, while table 6.1 shows the numbers of female moths that migrate from a particular site to other sites with the resulting defoliation percentages of all the larch forest sites. For further explanations and information of LBM simulation, please refer [BF88]. From Fig. 6.1, it is difficult for a layperson to imagine the spatiotemporal dynamics of LBM migration. Even the table 6.1 already contains all data to describe these dynamics in space and time; the figures are detached from their spatial locations. Therefore, interpretation of results from ecological simulation systems is often limited to experts. In this thesis, LBM dynamics is taken as an example to demonstrate how a real-time 4D visualization in an integrated spatiotemporal analysis system brings easily understandable illustration of insect migration and resultant vegetation changes to a wide range of users. The visualization method developed by this PhD thesis has been published in [WPI*06].

To improve the understanding of the ecological processes of migratory insects in both temporal and spatial dimensions, an integrated system, Interactive, Process Oriented, Dynamic Landscape Analysis and Simulation System (IPODLAS) [IPW*06], is being utilized. IPODLAS allows simulation and investigation of spatiotemporal ecological phenomena such as LBM migration and representation of simulation results in an intuitive manner. Here, a temporally explicit ecological modeling simulation system (Temporal simulation) is coupled with a spatially explicit Geographic Information System (GIS), and a real-time 4D visualization.

In this integrated spatiotemporal system, real-time visualization plays the role of representing results from ecological systems. To interactively visualize the simulation results in a 4D virtual environment, large data sets such as a digital elevation model and a terrain texture of study area must be additionally rendered in real-time. Moreover, for the visualization of insect migration dynamics, there are two alternating computing processes: 1) Insects migrating paths must be calculated according to the simulation output and terrain and 2) lively insects in the air have to be rendered dynamically. In addition, once insects have landed, the new defoliation level resulting from a change of local insect density needs to be visualized by changing the appearance of vegetation in terrain.

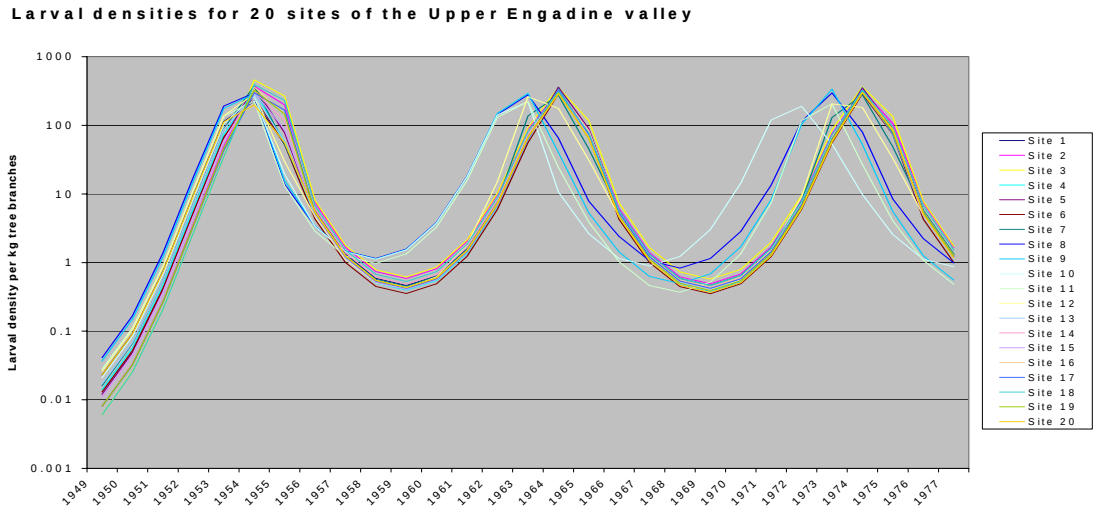


Figure 6.1: LBM larval density variation in the Upper Engadine valley results from the LBM-M9 model [BF88].

Table 6.1: Data table of LBM female migration number and defoliation percentage of every site in one year results from the LBM-M9 model [BF88].

Site	1	2	…	20
1	4765	126	…	286
2	0	64	…	103
3	324	0	…	32
…	…	…	…	…
20	712	21	…	124
Defoliation	0.00364	0.03211	…	0.04321

Unfortunately, memory sizes and computing speeds of today's common computers are insufficient to compute all these changes and photorealistically visualize all insects individually in real-time [RE03, RHS*98]. For the convenience of a wide range of users, it is necessary to explore efficient methods based on existing computing and rendering algorithms to visualize both insect dynamics and resultant vegetation changes in virtual terrain in real-time. The resulting spatiotemporal visualization achieves easier understanding of interactive and explorative simulation results, whereby simulation results depicting insect dynamics and vegetation changes are visually attached to the relevant 3D spatial locations. In the following sections, LBM dynamics in the Upper Engadine valley is used as an example to illustrate the potential of the visualization methods developed by this PhD thesis for displaying the dynamics of migratory insects.

6.2 Basic Data

When IPODLAS is applied to model and simulate LBM dynamics, both public domain GRASS (Geographic Resources Analysis Support System) and ESRI's ArcGIS/ArcInfo are used to obtain geographical locations and extents of the study area. Spatial data and functions from the GIS subsystem are used within the Temporal Simulation subsystem, RAMSES in this example, to simulate LBM spatiotemporal dynamics based on the LBM-M9 model. Here the visualization subsystem functions as an interface to users and displays the simulation results from RAMSES. Because of its extensible property and fast terrain rendering capabilities, the Virtual Terrain Project (VTP) servers as the software foundation for the real-time visualization of LBM dynamics in this case.

VTP is a creative project for easy construction of virtual terrain in an interactive, 3D digital format. It has open source code and hence allows the extension of existing functions and the access of the low level rendering Application Programming Interface (API). One of its main advantages is that its run-time environment makes use of Continuous Level of Detail (CLOD) algorithms for terrain rendering [RH98]. The work of Biegger (2004) shows that VTP can act as a suitable base for implementation of a visual system dedicated to simulation and analysis of a dynamic process, i.e. glacier fluctuations. Therefore, visualization of migratory insect dynamics and resulting vegetation changes can be developed as an embedded module within VTP.

6.2.1 Study Area

The Upper Engadine valley, located in the Swiss part of the European Alps, forms the study area. Herein, quantitative data on host trees, LBM population dynamics, numbers of annually migrating females, LBM larval densities, defoliation, and natural enemies have been collected continuously from 1949 to 1977 for each of the 20 sites [BF88, BR99]. Thus LBM dynamics in this valley represents an ideal study case or test bed for the spatiotemporal real-time visualization of migratory insects.

The information of geographical locations and extents of the 20 spatially homogenous LBM study sites within the Upper Engadine valley shown in Fig. 6.2 was generated by the GIS subsystem and has the same resolution as the digital terrain data used for landscape visualization. It is used by the visualization system, VTP, to determine the locations of LBM dynamics.

6.2.2 Data Preparation

In general, besides the geographical information of the study area, the following data sets are necessary for the spatiotemporal visualization of LBM dynamics and resultant vegetation changes in the Upper Engadine valley:

- A Digital Elevation Model (DEM) [© Tydac AG, Bern] to describe the geometry of the ground in the Upper Engadine valley. The used DEM is a raster-based elevation model with an original grid size of 50m x 50m. It was resampled to 10m x 10m using a bilinear interpolation scheme.
- A satellite image [© ESA/Eurimage, CNES/Spotimage, swisstopo/National Point of Contact NPOC] to describe the appearance of terrain in a photo-realistic manner. A mosaic of Landsat Thematic Mapper imagery with a raster size of 25m x 25m was used as a basic data set. Its spatial resolution was improved to 10m x 10m by image fusion with panchromatic SPOT data.
- Annual number of simulated female LBM migrating from each site to all other sites.
- Simulated defoliation percentage of the Larch trees following insect migration for each site in each year. This simulation was performed within RAMSES. Defoliation was categorized to 4 different levels following Baltensweiler and Rubli [BR99], colors for visualized forests were

chosen accordingly to represent a gradient from green, for no defoliation, to brown, for heavy defoliation, shown in table 6.2 and corresponding to the forest colors used in Fig. 6.5.

Table 6.2: Defoliation levels for categories of simulated defoliation percentage and the corresponding visualized forest color.

Defoliation percentage	Defoliation level	Forest color
0.00-0.01	None	Green
0.011-0.33	Light	Yellow
0.331-0.66	Medium	Orange
0.661-1.0	Heavy	Brown

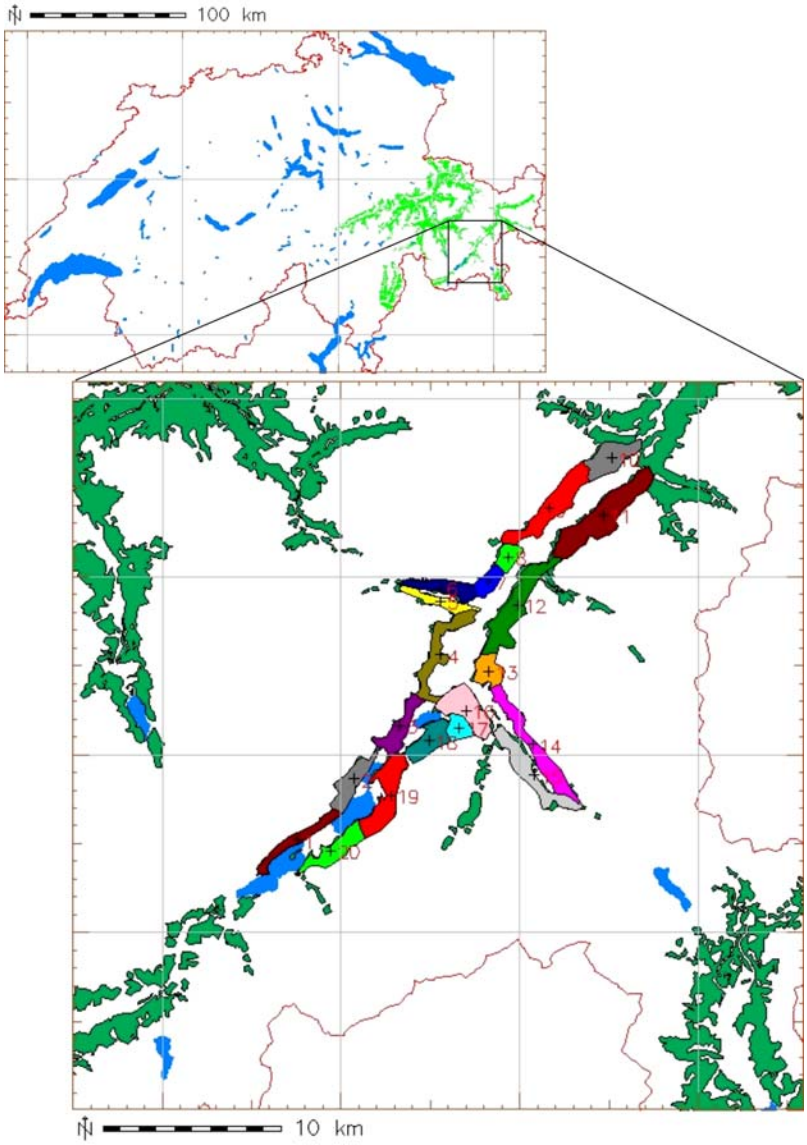


Figure 6.2: The distribution of larch forest sites in the Upper Engadine valley. Different colors are randomly used to distinguish sites.

6.3 Insect Cloud Modeling

According to the simulation results from RAMSES, many thousands and even millions of LBM may migrate from one site to others [BR99, Fis82, and Fis83]. Since the number of migratory female LBM from each site each year is given statistically, we can simply assume that moths from each site within the Upper Engadine valley move together and thus can be thought of as a 'cloud' with each smaller group of moths representing an internal particle of the cloud.

LBM researchers found that LBM migrating from each site normally land in several sites separately. Hence, during migration, some LBM may land in a particular site, while the others continue flying. Accordingly, the LBM cloud can be considered as being split into several small clouds flying to different target sites after taking off from each site. If a user views a visualization of migration from several different sites in a given time period, he may find it difficult to distinguish from which site a migrating cloud originates. For convenience, during visualization a different random color is assigned to individual insect groups originating from different sites.

Because LBM clouds are composed of living creatures that do not keep relative static positions within clouds during migration, they need to be visualized as dynamic clouds. Since the visualization subsystem aims to render real-time dynamic clouds in large numbers and at the same time the main task of CPU and GPU within the subsystem is to render a large terrain data set with a high-resolution satellite image, a simple and efficient modeling method is required to construct and simulate the dynamics of LBM clouds in a visually convincing way. As discussed in chapter 5, photorealistic random cloud models can be constructed by the procedural modeling method or the two-level marching cube method. In order to save the modeling time, a simplified procedural modeling method is used.

Ellipsoids, shown in Fig. 6.3, are used as implicit primitives and assigned locations, sizes, and weights to construct the macrostructure of a LBM cloud. A Gaussian distribution combined with a random noise function is applied to create its microstructure, which describes the distribution of particles within each ellipsoid. Moreover, the Gaussian center is irregularly located inside the ellipsoid to introduce more randomness to the particle distribution. Each internal particle is represented by a metaball with a density function identifying the distribution of a LBM subgroup within the sphere. To reduce the computation cost and accelerate rendering process, the center of a metaball is supposed to have the local highest density of a small LBM group and its density decreases according to a Gaussian distribution. Both ellipsoids and metaballs in a LBM cloud have different sizes and densities. The volume of total ellipsoids for a LBM cloud decides its size. Meanwhile, the number of the metaballs also increases or decreases accordingly to fill in the volume. As an example, Fig. 6.3 shows two LBM cloud models constructed by 4000 and 1000 metaballs respectively in 3 ellipsoids.

With the above approach, every LBM cloud has different initial structure. For realistic dynamic cloud simulation, fluid dynamics is a straightforward method to calculate velocity fields with an arbitrary initial structure [FSJ01, FM97, and Sta99]. However, this is impractical since it is computationally too expensive. An easier approach is to use the cellular automation method, which simplifies dynamic cloud motions into 8 directions in each cell of a grid [DK00]. The above two methods require to store every metaball's position and continuously update it according to a calculated velocity at that position. Since each LBM cloud may have hundreds or thousands of metaballs, these methods can cause a large burden on CPU and main memory. Hence, we assume that moths randomly fly inside each LBM cloud. Accordingly, we only need to store the macrostructure of each LBM cloud and Gaussian centers to distribute metaballs. To simulate dynamic LBM clouds, they are dynamically generated in each frame with random function under the constraints of Gaussian distributions. The reader may think that LBM clouds will change appearance unreasonably quick since every metaball has a new

position in every frame. However, this is not true since statistically metaball distributions have been decided by macrostructures and Gaussian centers. After rendering, discontinuous changes can only be distinguished at places having few metaballs. Therefore, the above simulation method exactly reflects our assumption of the unordered flight of moths inside LBM clouds.

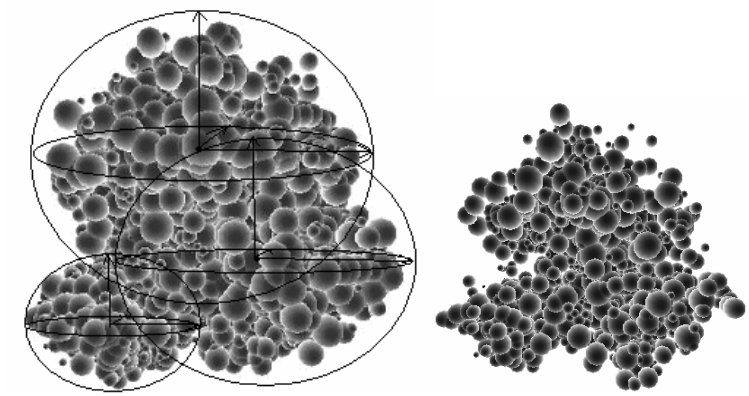


Figure 6.3: Two clouds shaped by 4000 and 1000 metaballs respectively in 3 ellipsoids.

6.4 Real-time Insect Cloud Rendering

"Rendering clouds is difficult because realistic shading requires the integration of the effects of optical properties along paths through the cloud volume, while incorporating the complex scattering within the medium" [HL01]. As discussed in chapter 5, the recording matrix method developed in this thesis is capable of rendering photorealistic clouds in a short time and is an efficient method to render dynamic clouds. However, it still cannot meet the requirement of rendering many dynamic LBM clouds at the same time in real-time, because of its time cost on calculating and displaying self-shading properties of clouds. Hence, instead of precisely computing light absorption and multiple scattering caused by metaballs inside clouds, we imitate the result by assuming that light intensity for each big ellipsoid in a cloud can be mapped onto an elliptic figure through its centre and facing the viewer. The intensities on metaballs inside this ellipse have an anisotropic Gaussian distribution. That means if we place a Gaussian center at the centre of the ellipse, light intensities will isotropically increase from the centre of the ellipse to its border. Accordingly, light intensities have inverse relationship to LBM density distributions in ellipsoids. It coincides with the assumption of LBM density distribution in the last section. Therefore, light absorption among metaballs can be directly simulated by particle blending.

In order to produce better visual effects, single scattering of light towards the view point is taken into account to re-adjust the color of each metaball in eyes. Approximate values are used as albedo and extinction coefficient to save rendering time. Gaussian textured billboards are then used to represent metaballs. According to equation 5.9, billboards are blended with each other and their alpha color channel, which equals to the extinction coefficient, controls the absorption extent of each metaball. The color blending technique is a standard capability of today's common GPU. Hence, all blending processes are executed automatically with a cheap cost. As an example, Fig. 6.4 shows three clouds, with a little more complicated macrostructures, fast rendered by this simplified method.

Since all the metaballs of a LBM cloud are finally blended into one cloud image, the dynamic impostor technique is used to insert the LBM cloud into the 3D virtual landscape. Because users are allowed to navigate freely inside the virtual landscape to observe the dynamics of LBM clouds, impostors must always be placed on planes facing the viewer, namely the direction from the centre of

an impostor to the viewer is always the normal direction of the impostor. Further knowledge about dynamic impostors has been discussed in section 5.3.

Figure 6.4: Clouds fast rendered by the developed simplified method.

6.5 Migration Path of Insect Cloud

Migration paths of LBM in the real world are complicated and difficult to determine. To simplify the computation, they are assumed by cooperative LBM experts [PIA*] to migrate in the following manner:

- When moths start migrating from each site, they gradually fly to a certain height directly above terrain. As the number of migrating LBM varies largely in different sites and years, we suppose that migration starts from a region near the centre of a site. As the migrating LBM number becomes larger, the region extends gradually to the whole site.

- Once moths reach the assumed height above terrain, they split into smaller LBM clouds, which fly directly to their respective destination sites. To enhance visual effects, we assume migration paths moving up and down according to terrain. Since every LBM cloud heads to its own destination site, there is a computable shortest path on terrain between the projection point of a LBM cloud center and the center of its destination site. We further assume that every LBM cloud flies along the shortest path and keeps the height above terrain. As soon as the projection point of a LBM cloud center exceeds the centre of its landing site, the LBM cloud is ready to land.

- The landing process is also a dispersive process. During the landing of a LBM cloud, its density is reduced, while its vertical extension becomes shorter and its cross section becomes gradually larger. Finally, the LBM cloud covers a certain region of the landing site and moths disappear from the users view. Herein, assumptions about the landing region are the same as the assumptions for the take-off region described above. The landing paths are implemented as individual straight lines between some sample points in the LBM cloud and points inside the landing region.

6.6 Resultant Vegetation Changes in the Landscape

Depending on LBM larval density, larch forests experience different degrees of defoliation. This results in significant color changes of the larch foliage during the summer, changing the appearance of the forested landscape within each LBM site considerably [BF88, BR99]. However, satellite images are not available for all the investigated years (1949-1977), and thus cannot fulfill our requirements. Moreover, since a large raster based satellite image is used as a texture layer for virtual terrain, it is time-consuming to change some parts of the image and reload it back to the texture cache to display

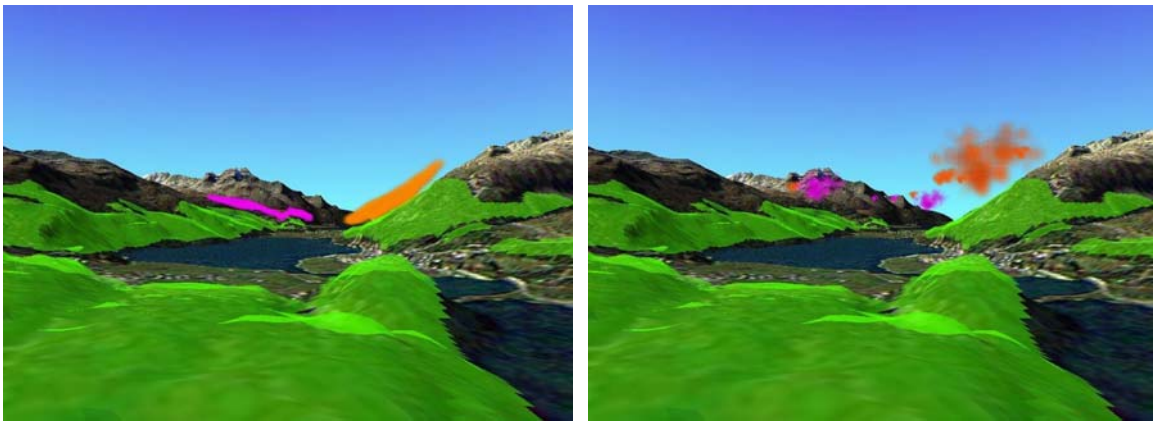
the yearly resultant vegetation changes in the whole study area in real-time. Therefore, geographical information about the larch forest sites is imported from the subsystem GIS in advance. It has the same resolution as the DEM used to form the virtual terrain. Then polygon-based representations of these sites are built slightly above the virtual terrain. By doing so, the highlighted resultant vegetation areas can easily be distinguished by viewers.

With the alpha blending technique once again, the colors of the sites can be combined with the underlying satellite image. Therefore, dynamic vegetation changes in real-time is possible to be visualized by changing only the colors of these sites. This method illustrates apparently annually resultant vegetation changes to users with low computation costs.

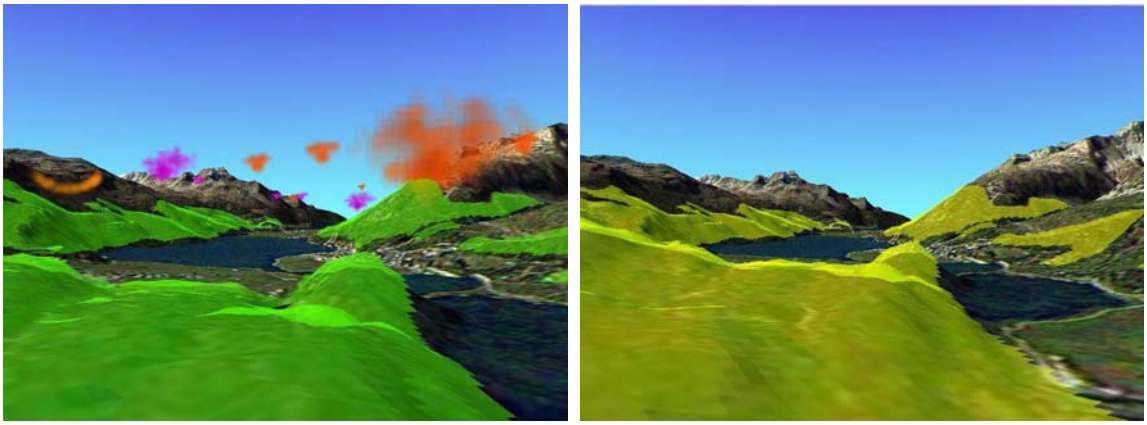
6.7 Results

To present spatiotemporal simulation results in a visually intuitive manner, an approach for 4D real-time visualization of migratory insects and resultant vegetation changes was developed. Standard digital terrain models and satellite images are used to render virtual pseudo-realistic landscapes. Against this background, groups of migrating insects are represented as continuously animated clouds. A simplified mathematical model is applied to make cloud modeling and rendering fast and efficient. Thus, the analysis outputs of LBM migration from the temporal ecological simulation system, namely data tables which detached from spatial locations, can be visually approximately expressed by the dynamic processes of ascending, splitting, flying, descending and dispersing of LBM clouds. Simultaneously, resulting from varied larch forest defoliations caused by annually different LBM densities, the appearance of the landscape is changed by dynamically blending colors of influenced areas with the underlying terrain texture.

Fig. 6.5 (a-d) show LBM clouds in migration and resultant vegetation changes with four phases. All orange LBM clouds originate from a same site while purple LBM clouds come from another site. In Fig. 6.5 (a) two LBM clouds ascend from their sites. Fig. 6.5 (b) shows migrating LBM clouds in groups after splitting. In Fig. 6.5 (c), an orange LBM cloud is landing and the other LBM clouds are still heading to their target sites. Because LBM cloud migration paths are moving up and down according to the ground, the LBM clouds are flying at different heights. Once each LBM cloud has landed, the simulated resultant level of defoliation in each larch forest site is displayed by changing the site colors according to the output of the temporal ecological simulation system. In Fig. 6.5 (d), originally green forest sites (none defoliation) in the Upper Engadine valley turn into yellow (light defoliation) after all the LBM clouds landed to their attractive sites.



(a) (b)



(c) (d)

Figure 6.5: (a) LBM clouds after taking off from their forest sites. (b) Several small clouds after a LBM cloud splitting. (c) An orange LBM cloud landing to its attractive forest site, while others heading to their target sites. (d) The appearance of forest sites in the next year.

Though several LBM clouds appear in Fig. 6.5, they are only originally from two sites. If all the 20 sites in the Upper Engadine valley are taken into consideration, many LBM clouds may fly in the air according to the simulation output of RAMSES and can be visualized by our simplified rendering method in real-time. To achieve this goal, LBM clouds are rendered not photorealistically as atmospheric clouds introduced in chapter 5. The real-time rendering method introduced in this chapter is a compromise between virtual quality and rendering speed. When the common computer's CPU and GPU is developed to be fast enough to render LBM cloud as atmospheric clouds in the future, the recording matrix method analysed in chapter 5 can be used to render dynamic photorealistic clouds efficiently.

6.8 Extensible Usages and Limitations

This chapter presented efficient methods developed in this thesis to visualize the simulation results of LBM dynamics and resultant vegetation changes in virtual terrain in real-time [WPI*06]. Because the simulation is based on statistical data, LBM are assumed to migrate in groups and visualized as clouds in the air. Consequent 4D dynamic scenarios give users a better understanding of LBM migration and its influence on local vegetation through a visual manner. Comparing to a table or a 2D picture, simulation results can be more understandable and directly perceivable by users.

Though the methods developed in this chapter are tested in the Upper Engadine valley, they could be also applied to other regions with different numbers of sites. As introduced in section 6.1, corresponding virtual terrain is constructed by a digital elevation model and a satellite image. Here, the digital elevation model determines the resolution of 3D terrain and the satellite image gives it a visually realistic appearance. The geographical information of a site can be analyzed and transferred from the GIS subsystem to the visualization subsystem by IPODLAS kernel. With a new simulation output from the ecological subsystem, the visualization subsystem can show users LBM dynamics and vegetation changes in a new study area.

Moreover, other migratory insect dynamics, which have similar migration properties as LBM, can be displayed by the same methods. Each subsystem is plugged in IPODLAS and communications among subsystems are taken care by its kernel. Therefore, the temporal ecological subsystem is changeable and corresponding simulation outputs based on the research of other migratory insects can be

transferred to the visualization subsystem. Their dynamics then can be displayed in virtual terrain in real-time.

The modeling and rendering methods presented in this chapter treat insect movements as a 'self-contained' process and do not take further physical parameters into account. For example, wind speed and direction could change cloud geometries and dynamics so heavily that insect cloud models should be accordingly reconstructed for each frame. Therefore this visualization is under the assumption of migrations taking place in a stable and undisturbed circumstance.

Because the input data of the visualization subsystem are simulation outputs of ecological subsystems where the simulation is normally based on statistical field data, the visualization subsystem can only display insect dynamics also in a statistical manner and hence assumptions are made to visualize the movements of migratory insects in groups. Therefore, the reality of visualization depends on the details of provided data and the computation capabilities of computers. In general, visualization can display simulation results in a spatial-temporal manner, but it does not mean what it shows is real.

Chapter 7 Flame and Smoke Visualization

Flames and their hot gas products, smoke, are common natural phenomena. We can find flames from torches, fireplaces and ovens, and smoke from chimneys in the real world quite often. Consequently, their appearances in virtual scenes can enhance the visual reality. Moreover, based on scientific simulation data and proper geographical information, the visualization of wildfire propagation in forests can serve as a training or education tool for the public. Therefore, the visualization of flames and smoke is meaningful in both entertainment and practice. However, because of the turbulent and easily changeable motion of flames and smoke as well as their abundantly colorful appearances, challenges still remain in how to visualize them efficiently and photorealistically.

Visualization of dynamic phenomena such as flames and smoke contains two aspects, modeling and rendering. The modeling contributes to the construction of initial shapes and the simulation of dynamic movements under initial conditions, boundary conditions and other constraints. The modeling of flames and smoke is extremely difficult since their movements can be easily changed dramatically under a small disturbance and have diverse characteristics under different conditions. Moreover, floating in the air with fast speeds, they are subject to convection and diffusion of the air flow and contain distinct vortices especially in the flow of smoke. The difficulty which remains in modeling smoke and flames is how to simulate their movements fast and visually realistically.

Because of the limitation of computation resources and the time required to update each frame in the visualization, engineering methods can not be directly applied to visualization. Therefore, many simplified methods are developed in the computer graphics area. This chapter presents some representative modeling methods developed by other researchers [Ebe97, FM97, Sta99, FSJ01, NFJ02, EMP*03, SRF05, and TLP06] and methods developed in this thesis for our particular purpose. Another important thing we need to bear in mind is that though most modeling methods for the visualization purpose are derived from methods for engineering usages, they are much simpler than those applied in engineering. This is because physical accuracy is the primary concern in engineering, while visualization aims to display phenomena in a visually convincing way as fast as possible.

Unlike their modeling methods, rendering methods of flames and smoke are different. The color of flames is determined by the blackbody radiation of soot and is closely related to the temperature distribution in the region of flames; but soot in smoke only appears black because the temperature of smoke is not high enough for soot radiating visible light and consequently the color of smoke is determined by the light scattered to the viewer by internal particles of smoke. Therefore, methods for flame rendering and smoke rendering are analyzed separately in this chapter.

7.1 Modeling Methods for the Visualization Purpose

The basic requirement for 4D (spatiotemporal) visualization is to visualize objects in a certain time as visually realistic as possible; and real-time visualization is most preferred. Therefore, all modeling methods for the visualization of flames and smoke take visual effects, modeling speed, or both as primary concerns. Accordingly, modeling methods can be divided into three categories: heuristic modeling method, compromised modeling method, and physically based modeling method.

A heuristic modeling method is not based on fluid dynamics. It only has one purpose: modeling the motion of flame and smoke as fast as possible without compromising too many visual effects. A compromised modeling method takes their physical motion properties into consideration while simplifying those properties heavily in order to accelerate the modeling process. As hinted by its name, a physically based modeling method concentrates more on physical motion properties of flame and smoke. Though the computation of physically based modeling methods takes more time than the other two kinds of methods, they are more interactive and their modeling results are more close to reality

than the other two methods. However, we must notice that physically based modeling methods in computer graphics are still simplifications of the methods for engineering usage, since modeling speed is a primary concern for the visualization purpose. The three categories have their own advantages and can be selected according to the requirement of individual applications. To give the reader an overview, representative modeling methods are introduced and analyzed in the following section. Inspired by these methods, this thesis develops two other methods which are presented in section 7.1.2 and 7.1.3 respectively.

7.1.1 State of Art

7.1.1.1 Heuristic Modeling Methods

The aim of the heuristic modeling method is to generate the motion of a flow as fast and visually realistic as possible. Ebert et al. [EMP*03, KPH*02, and SSE*03] modeled turbulent smoke with the **(a) single column method**, which only requires nearly ignorable computing time. The single column method initializes the smoke shape as a vertical cylinder. While the smoke rises, it disperses outwards and the center line of the cylinder continuously and randomly changes shape to introduce turbulence. The higher the smoke rises, the larger the turbulence is added. To make the smoke appear more turbulent, random values are added to the height of the smoke's internal particles and swirling effects are created by displacing internal particles according to a vertical helix. In other words, if we take the smoke rising direction as y axis, the x and z coordinates of each particle are displaced by the cosine and sine values of the helix rotation angle. In order to create more turbulence, another helix can be used for the displacement of the center of the first helix at different height levels. This is called **(b) two column method** in [EMP*03]. Though the computation speeds of the column methods are very fast, both the single column method and the two column method can only model small-scale smoke in a visually convincing way. Moreover, only a few small-scale vortices can be observed from their results. As an example, Fig. 7.1 shows a smoke generated by this method.

Another representative work of heuristic modeling methods is **(c) texture method**, which moves small textures inside the volume of smoke and flames directly [KCR00]. Unlike the former column methods which use particle systems to represent smoke and flames, this method utilizes images to represent densities of amorphous phenomena in small regions. To create turbulent effects, many small images, called footprints, are generated first by the procedural modeling method discussed in section 5.1.1 and weighted by a Gaussian function to create amorphous edges. Then the motion of smoke and flames can be divided into three categories: object dynamics, global dynamics, and local dynamics.

Figure 7.1: Smoke generated by the column method[20].

[20] This figure is downloaded from http://www.cs.umbc.edu/~ebert/procedural_images.html

The object dynamics is the motion or deformation of the entire object. It changes the overall shape or location of the object. These dynamics are accomplished by changing the entire volume of smoke and flames.

The global dynamics is the dominant behavior of smoke and flames, for instance, the upward movement and outward diffusion. These dynamics are accomplished by propagating small textures for each frame with user predefined speeds and small automatically generated random disturbances. Therefore, the global dynamics only displays linear movements.

The local dynamics creates swirling and flickering movements for smoke and flames. These dynamics are accomplished by randomly choosing some textures from the small texture stack to replace some textures used for the previous frame and changing texture coordinates to create texture animation.

(a) (b)

Figure 7.2: (a) Smoke generated by the texture-related method; (b) flames generated by the texture method [KCR00].

The texture modeling method takes the advantage of precalculated texture images, which can visually display some small-scale turbulence directly. It also largely reduces the amount of particles since one of its textures can represent the effect of a group of particles. Since the texture is also related to the final rendering process, the overall visualization speed of this method is very fast. However, the results, shown in Fig. 7.2, have detectably sharp edges because of its oversimplified movements and coarse density representations.

7.1.1.2 Compromised Modeling Methods

Lattice boltzmann model (LBM) is a representatively compromised flow modeling method. As an extension of the lattice gas automata method, it can be also regarded as an approximate representation of the Navier-Stokes equations [WZF*04]. It can be applied to modeling both compressible and incompressible flows, and investigating the vortex dynamics. To better understand the LBM, the lattice gas automata method is first introduced here.

(a) Lattice gas automata (LGA) is an application of cellular automata used for the simulation of fluid dynamics [WLM*03]. The first LGA, called **(a.1) HHP model** [HPP49], represents the flow region in a 2D grid. The motion of microscopic particles with unit mass and unit speed on the 2D grid have two types of rules: propagation and collision. Propagation means particles move along grid links and only one particle in a given direction can be found at a given time at each grid point, namely node. When two microscopic particles arrive at a node from two different directions, they have collision and they

will leave the node in the other two, previously unoccupied directions at the next time step. The velocity at each node is determined by the average velocity of up to four particles, whose velocities point into different directions. This model is simple and conserves mass and momentum, but it only has four velocity directions and hence lacks rotational symmetry.

Later the **(a.2) FHP model** was introduced for more velocity directions and rotational symmetry [FHP86]. This model uses a 2D hexagonal grid, namely each cell has six nearest neighbors and consequently six possible velocity directions, and the propagation and collision rules are still used to update particles. Accordingly, microscopic particles may move to the nearest six neighbors along their velocity directions and change directions when they have collisions. As shown in Fig. 7.3, the collision rule satisfies the mass and momentum conservation. When multiple collision states are possible, a random selection is made.

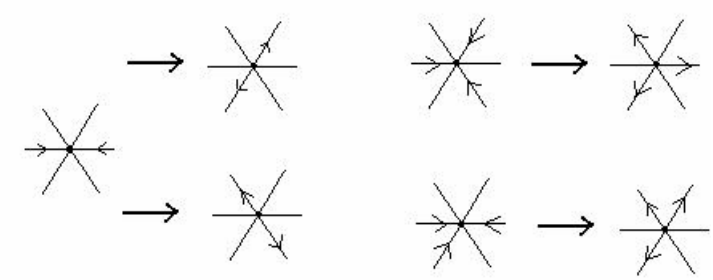


Figure 7.3: Collision rules of FHP model. Arrow denotes moving direction of a micro-particle [FHP86].

Following the same idea, the LBM was introduced to solve the statistical noise caused by the single Boolean operation in LGA models. It replaces the microscopic particles with real-valued densities [LWK03, WLM*03, and WZF*04]. The LBM is based on a cubic grid. Hence, there are 26 neighbors for each individual cell. So the LBM defines four symmetrical sub-lattices and their velocity directions are shown in Fig. 7.4.

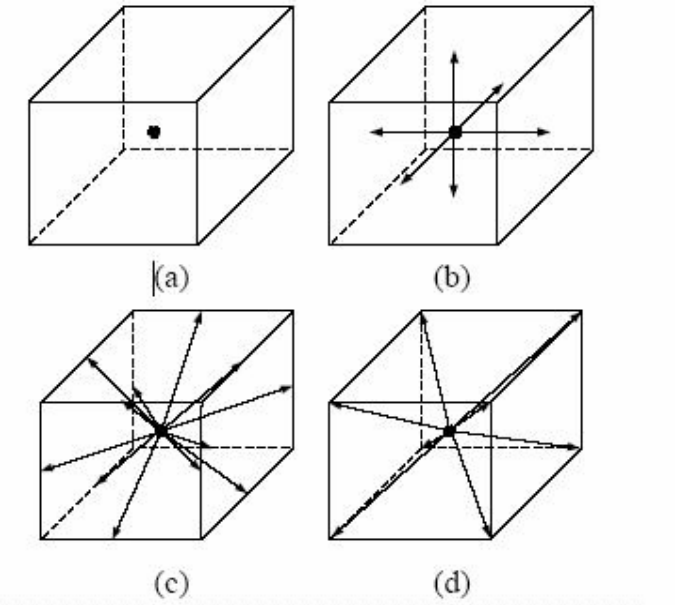


Figure 7.4: Four symmetrical sub-lattices defined on a cubic grid [WLM*03].

The velocity vectors in each cell are denoted as arrows in Fig. 7.4 (a-d).Particles move with the same velocity 0.0, $1.0, \sqrt{2}$, and $\sqrt{3}$ in (a), (b), (c), and (d) respectively. In practice, to insure efficiency,

isotropy, and stability, only three sub-lattices, (a), (b), and (c), are selected to compose a LBM lattice geometry.

Similar to LGA models, the LBM also updates the density value at each node based on collision and propagation. They can be described by the following equations:

$$\text{Collision: } f_{qi}^{new}(x,t) - f_{qi}(x,t) = \Omega_{qi} \tag{7.1}$$

$$\text{Propagation: } f_{qi}^{new}(x,t) = f_{qi}(x + e_{qi}, t+1) \tag{7.2}$$

where $f_{qi}(x,t)$ is the microscopic particle density distribution at node x, at time t, and moving in the direction of qi. Here, q denotes the type of movements, namely q equals to *0* for zero velocity shown in Fig 7.4 (a); q equals to *1.0* for velocities shown in Fig 7.4 (b); q equals to $\sqrt{2}$ for velocities shown in Fig 7.4 (c); and q equals to $\sqrt{3}$ for velocities shown in Fig 7.4 (d) . i counts the sub-lattice vectors; Ω_{qi} is a general collision operator; and e_{qi} is the unit vector, representing the particle velocity along the lattice link. At a time step t, the density distribution value at every node is updated based on the collision operator. Then, in the next time step, the new density distribution value propagates to the nearest node along the velocity vector e_{qi}.

It is crucial to select Ω_{qi} in such a way that the mass and momentum are conserved locally. Wei et al [WLM*03] assumed that for the density f_{qi} in each cell, there is always a local equilibrium density distribution f_{qi}^{eq}, whose value only depends on the conserved quantities, density ρ and velocity $\vec{U}$. Based on equation 7.1 and 7.2, Wei et al brought out a new kinetic equation:

$$f_{qi}(x + e_{qi}, t+1) - f_{qi}(x,t) = -1/\tau\,(f_{qi}(x,t) - f_{qi}^{eq}(\rho, \vec{U})) \tag{7.3}$$

where τ is the relaxation time scale, which controls the density change at each time step. It is another form of viscosity and must be greater than 0.5 in the LBM. The macroscopic density ρ and velocity $\vec{U}$ of a cell can be calculated by the following equations.

$$\rho = \sum_{qi} f_{qi} \tag{7.4}$$

$$\vec{U} = 1/\rho \sum_{qi} f_{qi} e_{qi} \tag{7.5}$$

According to the work of Chopard and Masselot [CM99], the equilibrium distribution can be presented by a linear formula [WLM*03]:

$$f_{qi}^{eq} = \rho(A_q + B_q(e_{qi} \bullet \vec{U}) + C_q(e_{qi} \bullet \vec{U})^2 + D_q(\vec{U})^2) \tag{7.6}$$

where the coefficients A_q, B_q, C_q and D_q depend on the applied sub-lattice geometry. A coefficients table 7.1 is given in [WLM*03], based on the model containing (a), (b), and (c) sub-lattices shown in Fig. 7.4.

Table 7.1 coefficients of a sub lattice geometry [WLM*03].

	Sub-lattice (a)	Sub-lattice (b)	Sub-lattice (c)
Aq	1/3	1/18	1/36
Bq	0	1/6	1/12
Cq	0	1/4	1/8

Dq	-1/2	-1/12	-1/24

For each time step, ρ and $\vec{U}$ are first computed. Then the new density distribution is calculated according to equation 7.6 and 7.3. At last, they are further modified according to boundary conditions. There are different methods to meet boundary conditions. Among them, the direct bounce-back is the simplest one. In order to get better results, the method of modifying equilibrium equations is introduced by Wei et al [WLM*03]. As shown in Fig. 7.5, this method computes a virtual density distribution value $f_{qi}(x_b,t)$ at the node b , as if it is a node in the region of the flow. Then this virtual value can be used to obtain the density distribution at the node f according to the propagation rule.

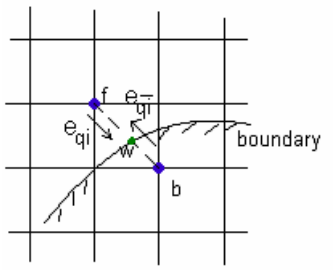


Figure 7.5: A 2D projection of two nodes on different sides of a non-penetrable boundary [WZF*04].

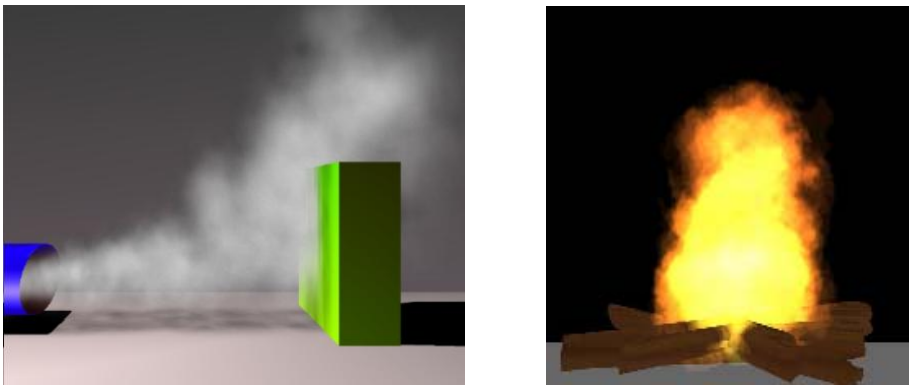
To accelerate the computation of the density distribution, taking the strength of the Graphics Processing Unit (GPU), Wei et al [WZF*04] grouped the density distribution into arrays according to their velocity directions. All density distributions with the same velocity direction are grouped into an array, while keeping the neighboring relationship in the original model. For a 3D model, each array forms a volume and is stored as a stack of 2D textures, namely each layer in the 3D model is saved as a 2D texture. For example, there are 19 arrays for a LBM lattice geometry constructed by sub-lattices in Fig. 7.4 (a-c). If it is divided into n^3 cells, each array has n 2D textures and each texture contains n^2 values.

At each time step, multiple textures are mapped from the texture space to the image space by setting the texture interpolation to be nearest, and choosing the resolution of framebuffer in GPU and texture coordinates properly. The propagation of $f_{qi}(x,t)$ is accomplished by shifting textures in the directions of their associated velocities. Since textures are always projected onto a plane orthogonally to the view direction, velocity directions can be decomposed into two parts, velocity components within the plane and velocity components orthogonally to the plane. Then for the propagation of $f_{qi}(x,t)$ driven by the velocity components within the plane, the texture of $f_{qi}(x,t)$ only need to be translated appropriately. Propagating $f_{qi}(x,t)$ in the orthogonal direction can be simply done by renumbering the textures in their stack.

The LBM provides the motion of gaseous phenomena in a large-scale. To add small-scale turbulent details and render gaseous phenomena in real-time, textured splatting is applied in [WLM*03], which has been discussed in section 5.2.2. Their rendering approach is as follows: First, a set of textures containing turbulent details, which match the modeling phenomenon, are obtained from real images or created according to the noise function. To ensure properly fuzzy appearances, textures must be weighted with a smooth function like Gaussian. When a particle enters the fluid field, an initial texture is selected from the texture database randomly and assigned to this particle. Then, the particle moves with its texture in the simulated velocity field. The longer the particle stays in the field, the lighter its color appears, till it eventually disappears.

As an example, Fig. 7.6 shows two frames respectively from real-time smoke and campfire visualizations in [WZF*04]. As we can observe, it achieves high frame rates with the compromise of

visual effects. Generally speaking, the LBM focuses on physical-based fast visualization instead of physical-based visually realistic visualization, which is discussed in the next section.



(a) (b)

Figure 7.6: (a) Smoke visualization using the LBM; (b) campfire visualization using the LBM [WZF*04].

7.1.1.3 Physically Based Modeling Methods

The physically based modeling method is based on the incompressible Navier-Stokes equations (equation 2.2 and 2.3) discussed in section 2.3.1.3. They are non-linear equations which conserve mass and momentum respectively and are broadly applied in the engineering areas. Therefore, the modeling process is indeed to solve the incompressible Navier-Stokes equations iteratively and update internal particles of smoke and flames according to the simulated velocity field at each time step before the rendering process.

To better understand the incompressible Navier-Stokes equations, they are rewritten again with velocity $\vec{U} = (u, v, w)$ in Cartesian coordinates:

$$\partial u/\partial x + \partial v/\partial y + \partial w/\partial z = 0 \tag{7.7}$$

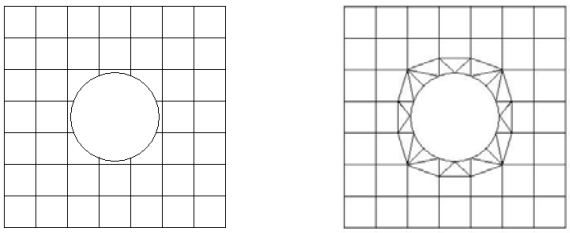
$$\rho \partial u/\partial t = -\rho(u\partial u/\partial x + v\partial u/\partial y + w\partial u/\partial z) - \partial P/\partial x + \rho\upsilon(\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 + \partial^2 u/\partial z^2) + f_x \tag{7.8}$$

$$\rho \partial v/\partial t = -\rho(u\partial v/\partial x + v\partial v/\partial y + w\partial v/\partial z) - \partial P/\partial y + \rho\upsilon(\partial^2 v/\partial x^2 + \partial^2 v/\partial y^2 + \partial^2 v/\partial z^2) + f_y \tag{7.9}$$

$$\rho \partial w/\partial t = -\rho(u\partial w/\partial x + v\partial w/\partial y + w\partial w/\partial z) - \partial P/\partial z + \rho\upsilon(\partial^2 w/\partial x^2 + \partial^2 w/\partial y^2 + \partial^2 w/\partial z^2) + f_z \tag{7.10}$$

Here, equation 7.7 is the mass conservation equation for the incompressible flow, corresponding to equation 2.2. Equations 7.8 to 7.10 ensure the momentum conservation, corresponding to equation 2.3, and the items on their right side are convection, pressure correction, diffusion, and external forces in *x, y, z* directions respectively. Accordingly, to calculate these items for each frame fast and efficiently in the modeling process, methods are developed to aim at the computation of individual items. Focusing on visualization, some representative work and methods developed in this thesis are presented in the following subsections. However, to have a better apprehension, before the introduction of individual modeling methods, let us first investigate the type of grids which contain the modeling volume and where modeling methods are carried out.

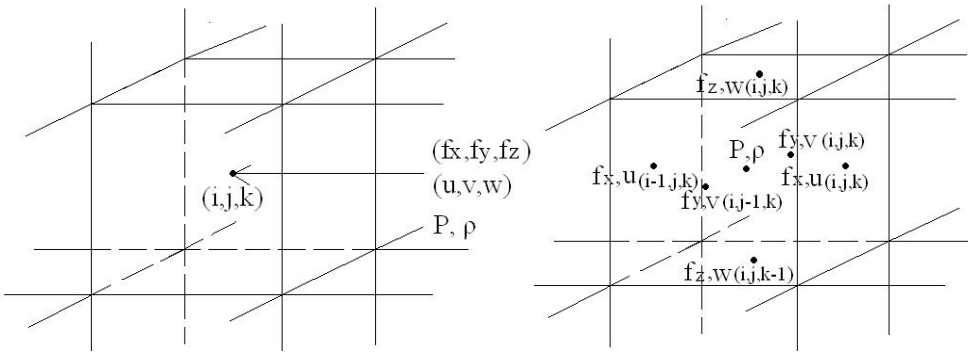
According to the shape of cells, there are two basic types of grids: **(a) structured grids** and **(b) unstructured grids**. Structured grids have regular cells and their cell vertices lie in the intersection of a family of lines. Unstructured grids are formed by a combination of triangular and quadrilateral cells. Structured grids have simpler data structure and require less programming efforts and computational time than unstructured grids; while the main advantages of unstructured grids are their great flexibility in adapting the mesh to complex boundaries and the accuracy in modeling flow properties based on finer meshes at the place close to the boundary. As an example, Fig. 7.7(a) shows a 2D structured grid having a circle boundary in the center and one of its corresponding 2D unstructured grids is shown in Fig. 7.7(b).



(a) (b)

Figure 7.7: (a) a 2D structured grid; (b) a 2D unstructured grid.

Since one primary concern of visualization is the computing time cost, the **(a.1) equidistant structured grid** is normally chosen for its simplicity. According to the location where quantities such as velocity and pressure are defined, structure grids can be further divided into **(a.1.1) collocated grids** and **(a.1.2) staggered grids**. Collocated grids define the quantities in the center of each cell, while staggered grids define the quantities on cell surfaces or at the cell vertices. Fig. 7.8 shows a 3D cell of a collocated grid and a 3D cell of a partially staggered grid. Both grids are commonly used for modeling flows for the visualization purpose.



(a) (b)

Figure 7.8: (a) a cell of a collocated grid; (b) a cell of a partially staggered grid.

The size of the grid plays an important role in the modeling process. On one side, it is directly related to the computing time. A fine grid requires more storage space and more computing time for all the cells than a coarse grid. On the other side, it is also closely related to the computational accuracy. To fast update the internal particles of a flow, the velocity at an arbitrary place in a grid is normally the linear interpolation of velocities in surrounding cells. It generates unavoidable errors since the actual velocity distribution in the grid is not linear. Moreover, as it will be discussed in the later sections, a coarse grid may generate large computation errors during the calculation of the velocity or pressure gradient. The accumulation of errors may lead to a divergent result and hence destroy the modeling process after several iterations.

Since the Navier-Stokes equations are non-linear, computational convergence is an important issue in the modeling process. The methods introduced in the following sections also take convergence as a

primary concern. A successful modeling method must generate a convergent solution no matter at which simulation time step.

(1) Semi-Lagrangian Scheme

Semi-Lagrangian Scheme is a stable and convergent scheme to compute the convection items in equations from 7.8 to 7.10. If we only consider the convection item in equation 7.8, we have:

$$\partial u / \partial t = -(u\partial u / \partial x + v\partial u / \partial y + w\partial u / \partial z) \tag{7.11}$$

To solve equation 7.11, a straightforward solution is to apply a finite differential method (FD). Equation 7.12 to 7.14 give the first order derivatives of the backward differences scheme (BDS), the forward differences scheme (FDS), and the central differences scheme (CDS) respectively for equidistant structured grids. Among them, CDS has the smallest truncated error and hence is commonly used to solve differential equations. The second order derivative of CDS is given in equation 7.15.

$$\partial u / \partial x = (u_{i,j,k} - u_{i-1,j,k}) / \Delta x + \Delta x(\partial^2 u / \partial x^2) / 2 \tag{7.12}$$

$$\partial u / \partial x = (u_{i+1,j,k} - u_{i,j,k}) / \Delta x - \Delta x(\partial^2 u / \partial x^2) / 2 \tag{7.13}$$

$$\partial u / \partial x = (u_{i+1,j,k} - u_{i-1,j,k}) / (2\Delta x) - \Delta x^2(\partial^2 u / \partial x^2) / 6 \tag{7.14}$$

$$\partial^2 u / \partial x^2 = (u_{i+1,j,k} - 2u_i + u_{i-1,j,k}) / \Delta x^2 + o(\Delta x^2) \tag{7.15}$$

Here, $\triangle x$ is the cell's edge length, $o(\Delta x^2)$ is the truncated error, which is proportionally to the square of $\triangle x$. By applying equation 7.14 to equation 7.11, we get:

$$\begin{aligned} &2(u^{t+\Delta t}_{i,j,k} - u^t_{i,j,k}) / \Delta t = -u^t_{i,j,k}(u^t{}_{i+1,j,k} - u^t{}_{i-1,j,k}) / \Delta x \\ &- v^t_{i,j,k}(u^t{}_{i,j+1,k} - u^t{}_{i,j-1,k}) / \Delta x - w^t_{i,j,k}(u^t{}_{i,j,k+1} - u^t{}_{i,j,k-1}) / \Delta x \end{aligned} \tag{7.16}$$

From equation 7.16 we can observe that if differences between velocities are too large or/and the time interval $\triangle t$ is too long, velocity at the cell *(i,j,k)* can easily grow up to an infinite value and result in a divergent computation. To solve this problem, Semi-Lagrangian Scheme is introduced by Stam [Sta99] to compute the convection item in a stable way and broadly used in later research works [FSJ01, NFJ02, Sta03a, Sta03b, SRF05, and TLP06].

The Semi-Lagrangian Scheme can be understood intuitively from the meaning of convection. It simply says that, to obtain the velocity at a point x at the time step $t+\triangle t$, the point x can be back traced through the velocity field over a time $\triangle t$; then, the new velocity at the point x can be set to the velocity that the particle, now at x, had at its previous location a time $\triangle t$ ago. This defines a path $p(x,s)$ corresponding to a partial streamline of the velocity field as shown in Fig. 7.9. Therefore, the Semi-Lagrangian Scheme can be defined as:

$$\vec{U}(x,t) = \vec{U}(p(x,t-\Delta t)) \tag{7.17}$$

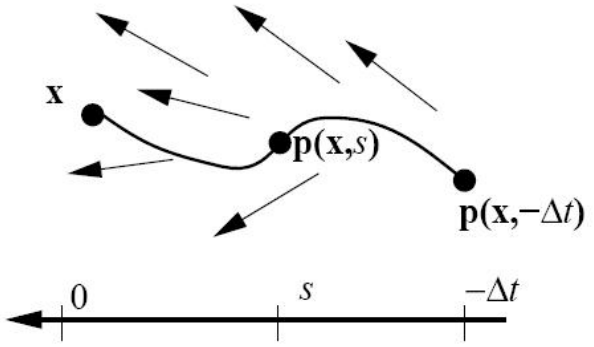


Figure 7.9: back-trace path in a velocity field [Sta99].

In implementation, to simplify the computation, the velocity at a point x at the time step $t+\triangle t$ is approximately set to the velocity, where the particle at x propagates to, at the time step t, by assuming "convection" is the exchange of two velocities after a time $\triangle t$. In other words, if the velocity at x at the time step t is $\vec{U}_t$, then its new velocity, $\vec{U}_{t+\Delta t}$, at the next time step $t+\triangle t$ is equal to the velocity at $x+\vec{U}_t \triangle t$ at the time step t. Fig. 7.10 illustrates this solution.

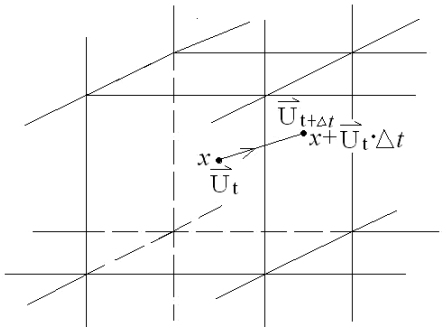
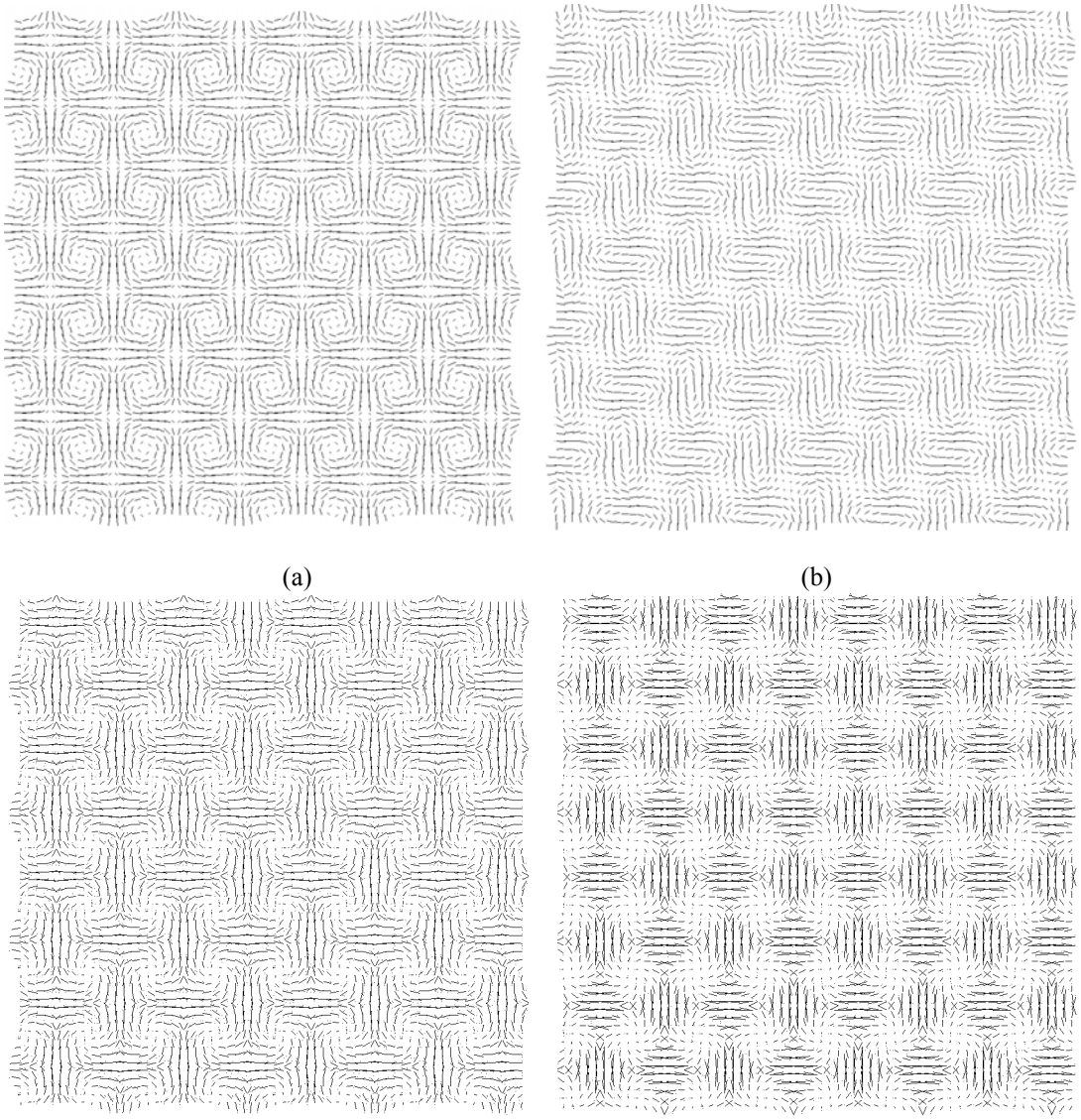


Figure 7.10: Approximate implementation of the Semi-Lagrangian Scheme.

From this approximation we can observe that the new velocity at any place in a grid at any time step is never larger than the maximum velocity at the former time step. Therefore, the Semi-Lagrangian Scheme ensures the convergence of the computation on the convection item, no matter how long the time step $\triangle t$ is. We can also observe from the approximation that the convection item gradually damps out the original velocity.

To compare the effects of the Semi-Lagrangian Scheme and the classic CDS when they are applied to the calculation of the convection item, we can consider an initially 2D repetitive velocity field, as shown in Fig. 7.11 (a), only under the convection condition. With the simulation implemented in this thesis, Fig. 7.11 (b) shows a convergent velocity field calculated by the Semi-Lagrangian Scheme after 0.2 second. With this scheme, the velocity field stably and gradually vanishes after 8 seconds. Fig. 7.11 (c) and (d) show the same velocity field under the convection condition after 0.1 second and 0.2 second respectively calculated by CDS. The velocity field becomes divergent after 0.25 second.

In general, the Semi-Lagrangian Scheme can approximately calculate the convection item fast and stably. However, its oversimplification wipes off vortical properties from the convention and hence results in too stable flows.



(a) (b)

(c) (d)

Figure 7.11: (a) Initial velocity field; (b) the velocity field calculated by the Semi-Lagrangian Scheme after 0.2 second; (c) The same velocity field originally shown in (a) calculated by the central differences scheme on the convection item after 0.1 second; (d) the velocity field after 0.2 second.

(2) Viscid and Inviscid Flows

The viscosity of a flow can be defined as a force which prevents the flow to move. It is analogous to the friction between solid bodies and it also serves as a mechanics for transforming kinetic energy into thermal energy [Tri89]. For incompressible flows, the influence of viscous force on the velocity field is expressed by viscid items. If we only consider the viscid item in equation 7.8, we have:

$$\partial u / \partial t = \upsilon(\partial^2 u / \partial x^2 + \partial^2 u / \partial y^2 + \partial^2 u / \partial z^2) \tag{7.18}$$

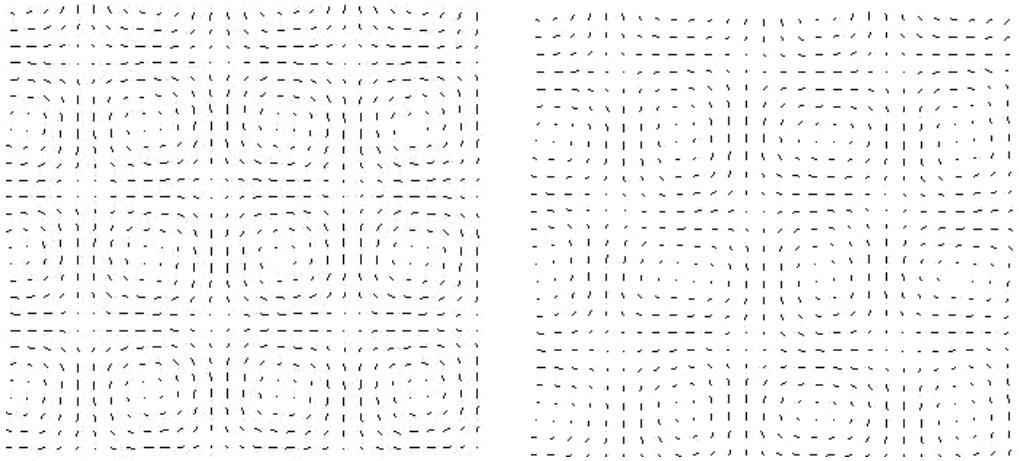
After applying equation 7.15 to equation 7.18, we get:

$$(u^{t+\Delta t}_{i,j,k} - u^{t}_{i,j,k})/\Delta t = \upsilon((u^{t}{}_{i+1,j,k} - 2u^{t}_{i,j,k} + u^{t}{}_{i-1,j,k})/\Delta x^2 + (u^{t}{}_{i,j+1,k} - 2u^{t}_{i,j,k} + u^{t}{}_{i,j-1,k})/\Delta x^2 + (u^{t}{}_{i,j,k+1} - 2u^{t}_{i,j,k} + u^{t}{}_{i,j,k-1})/\Delta x^2) \quad (7.19)$$

where υ is the kinematic viscosity, it is about 0.15×10^{-4} m^2/s[21] for the air around us. We can observe from equation 7.19 that if the differences of velocities between a cell and its neighborhoods especially in a fine grid are not too high, the viscosity item will not play an important role in a long term. However, it does retard the change of velocity fields and gradually attrite the velocity field to zero as hinted by its name. Therefore, in order to accelerate the modeling process, some applications, for example [Sta99], ignore the viscosity (diffusion) item and use the inviscid Euler equation (equation 2.4) instead.

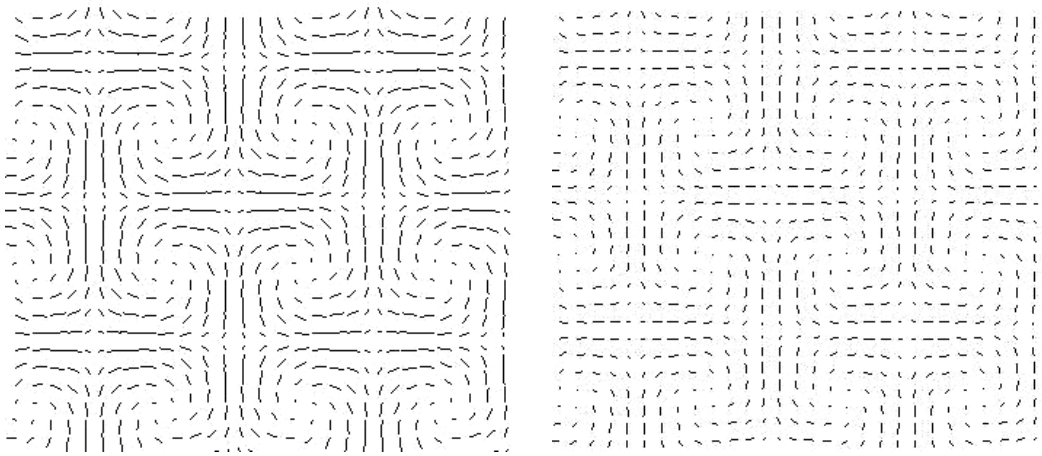
To make the effect of the diffusion intuitively to the reader, simulation, which is developed in this thesis, is carried out on the initially 2D repetitive velocity field and results are shown in Fig. 7.12. Fig. 7.12 (a) illustrates the velocity field initially shown in Fig. 7.11 (a) and calculated by CDS only on the diffusion item after 0.2 second with $\upsilon=0.05$ m^2/s (in order to see the effect of viscosity more apparently). Fig. 7.12 (b) shows the same initial velocity field under the influence of both convection and diffusion after 0.2 second. Here, the convection item is computed with the Semi-Lagrangian Scheme. Fig. 7.12 (c) and (d) show the same velocity field also under the influence of both convection and diffusion after 0.1 and 0.2 second respectively. Here, the convection item is computed by the CDS.

We can observe from Fig. 7.12 (c) and (d) that if the viscosity is large enough, it can make the velocity field convergent as comparing to Fig. 7.11 (c) and (d) and even decrease the velocity field to zero. This conclusion also stands when we compare the Fig. 7.12 (b) to Fig. 7.11 (b). Therefore, the diffusion item is only a 'friction' item, which prevents the fast change of velocity fields.



(a) (b)

[21] The value of υ for air is from http://scienceworld.wolfram.com/physics/KinematicViscosity.html



(c) (d)

Figure 7.12: The velocity field initially shown in Fig. 7.11 (a) under the influence of (a) only diffusion after 0.2 second; (b) both convection and diffusion after 0.2 second (the convection item is computed by the Semi-Lagrangian Scheme); (c) both convection and diffusion after 0.1 second (the convection item is computed by the CDS); (d) both convection and diffusion after 0.2 second (the convection is computed by the CDS).

(3) Pressure Field

The pressure distribution in the area of smoke and flames is very hard to measure. A general solution in Fluid Dynamics is to derive it from the incompressible Navier-Stokes equations. If we apply $\nabla\cdot$ to both sides of equation 2.3, we get:

$$\partial(\nabla \bullet \vec{U})/\partial t = -\nabla \bullet (\vec{U} \bullet \nabla \vec{U}) - \nabla^2 P/\rho + \upsilon \nabla^2 (\nabla \bullet \vec{U}) + \nabla \bullet \vec{f} \tag{7.20}$$

For an incompressible flow, we can apply equation 2.2 to equation 7.20, then

$$\nabla^2 P = -\rho \nabla \bullet (\vec{U} \bullet \nabla \vec{U}) + \rho \nabla \bullet \vec{f} \tag{7.21}$$

Equation 7.21 becomes a typical Poisson equation. After replacing the right side of equation 7.21 with b for easier explanation and understanding, and applying the 2^{nd} CDS (equation 7.15) on equation 7.21, we get:

$$(p_{i+1,j,k} - 2p_{i,j,k} + p_{i-1,j,k})/\Delta x^2 + (p_{i,j+1,k} - 2p_{i,j,k} + p_{i,j-1,k})/\Delta x^2 + (p_{i,j,k+1} - 2p_{i,j,k} + p_{i,j,k-1})/\Delta x^2 = b_{i,j,k} \tag{7.22}$$

As we can observe from equation 7.22, the computation of pressure for each cell in a grid is related to its six neighbors. Therefore, the pressure field needs to be computed together. If the grid has m^3 cells, we have m^3 equations. By writing all the pressure items in a vector, we get:

$$\vec{P} = (p_{0,0,0}, ..., p_{m,0,0}, p_{m,0,1}, ... p_{m,m,0,}, p_{m,m,1}, ... p_{m,m,m}) \tag{7.23}$$

Putting all the coefficients on the left side of equation 7.22 into a matrix A, we have a sparse matrix with dimension $m^3 \times m^3$. If we take $b_{i,j,k}$ in equation 7.22 as an element of a vector $\vec{b}$, the pressure field can be computed from the following linear equation:

$$A \cdot \vec{P} = \vec{b} \tag{7.24}$$

There are many numerical methods to solve the linear equation which has the style of equation 7.24. Generally speaking, they can be divided into three categories: **(a) direct methods**, **(b) iterative methods**, and **(c) multigrid methods**.

Direct methods like **(a.1) Gauss elimination** and **(a.2) LU-decomposition** are normally used to solve low dimension linear equations. For the large sparse matrix *A* in equation 7.24, direct methods usually fail to obtain the exact solution. Iterative methods are more flexible than direct methods. Under their particular preconditions, they can converge towards the exact solution through the iteration until the required accuracy is reached. Iterative methods can be further divided into two categories: **(b.1) stationary methods**, such as **(b.1.1) Jacobi**, **(b.1.2) Gauss-Seidel**, and **(b.1.3) Successive Overrelaxation**, and **(b.2) non-stationary methods**, such as **(b.2.1) Conjugate Gradient Method (CG)**, **(b.2.2) Generalized Minimal residual**, **(b.2.3) BiConjugate Gradient**, and **(b.2.4) Quasi-Minimal Residual**.

As hinted by its name, stationary iterative methods perform the same operation in each iteration. For a close view of the stationary method, we can first decompose the matrix *A* in equation 7.24 as $A=N-Q$, where *N* is an easily invertible matrix. Accordingly, equation 7.24 is changed to:

$$\vec{P} = N^{-1}Q\vec{P} + N^{-1}\vec{b} \tag{7.25}$$

An iteration procedure can be constructed as

$$\vec{P}^{m+1} = N^{-1}Q\vec{P}^{m} + N^{-1}\vec{b} \tag{7.26}$$

where, *m* is the iteration counter. Comparing to its exact solution, $\vec{P}$, the error of the solution $\vec{P}^{m}$ at the m^{th} step, ε^{m}, is

$$\varepsilon^{m} = \vec{P}^{m} - \vec{P} \tag{7.27}$$

Replacing $\vec{P}^{m}$ in equation 7.27 according to equation 7.26, we get:

$$\varepsilon^{m} = N^{-1}Q\vec{P}^{m-1} + N^{-1}\vec{b} - \vec{P} \tag{7.28}$$

According to equation 7.25, equation 7.28 can be further written as:

$$\varepsilon^{m} = N^{-1}Q\vec{P}^{m-1} + N^{-1}\vec{b} - (N^{-1}Q\vec{P} + N^{-1}\vec{b}) = N^{-1}Q\vec{P}^{m-1} - N^{-1}Q\vec{P} = N^{-1}Q\varepsilon^{m-1} \tag{7.29}$$

Then, we have:

$$\| \varepsilon^{m} \| \leq \| N^{-1}Q \| \cdot \| \varepsilon^{m-1} \| = \| N^{-1}Q \|^{m} \varepsilon^{0} \tag{7.30}$$

Therefore, if the norm of the matrix $N^{-1}Q$ is smaller than 1.0, ε^{m} is smaller than ε^{m-1} and ε^{0}. In this case, the error of the computation on the pressure field decreases exponentially as *m* increases.

For all the iteration methods, there must be a proper condition for their stop. Otherwise, the computation process will never end. Since ε^{m} is related to the exact solution, which we intend to

compute, the residual after m^{th} iterations, R_m, is normally used to be the criteria for ending the iteratively computation process. As hinted by its name, the residual is defined as:

$$R_m = A \cdot \vec{P}^m - b \tag{7.31}$$

There are different methods to decompose the matrix A into N and Q. Nearly all the stationary methods make use of the following decomposition.

$$A = L + D + U \tag{7.32}$$

where L is the lower triangular matrix, D is the matrix with only the diagonal elements, and U is the upper triangular matrix of A.

The Jacobi method (b.1.1) sets $N=D$ and $Q=-U-L$. According to equation 7.26, the pressure $p_{i,j,k}$, namely p_l in the vector $\vec{P}$ where $l=i\times m2+j\times m+k$, can be iteratively calculated by the following equation with initial $\vec{P}^0 = (1,1.....1)$.

$$p_l^{m+1} = (b_l - \sum_{k \neq l} a_{l,k} p_k^m) / a_{l,l} \tag{7.33}$$

where, $a_{l,k}$ is the element *(l,k)* in the matrix A. Therefore, the Jacobi method is based on solving every variable, p_l, with respect to the other variables, p_k. One iteration of this method corresponds to solving every variable once. This method is easy to understand and implement. As long as the norm of the matrix $N^{-1}Q$ is smaller than 1.0, this method is convergent, but the convergence is relatively slow.

The Gauss-Seidel method (b.1.2) is similar as the Jacobi method. It takes $N=D+L$ and $Q=-U$. Then, according to equation 7.26, p_l can be computed as:

$$p_l^{m+1} = (b_l - \sum_{k=0}^{l-1} a_{l,k} p_k^{m+1} - \sum_{k=l+1}^{m^3} a_{l,k} p_k^m) / a_{l,l} \tag{7.34}$$

Therefore, the Gauss-Seidel method uses updated values as soon as they are available in each iteration. In general, if the Jacobi method converges, the Gauss-Seidel method will converge faster than the Jacobi method, though it is still slow.

The Successive Overrelaxation method (b.1.3) is derived from the Gauss-Seidel method. It introduces an additional parameter ω to divide the matrix D into two parts. Then it has $N=D/\omega+L$, and $Q=(1-\omega)D/\omega-U$. Consequently, p_l can be computed as:

$$p_l^{m+1} = \omega p_{l,Gauss-Seidel}^{m+1} + (1-\omega) p_l^m \tag{7.35}$$

where, $p_{l,Gauss-Seidel}^{m+1}$ is the m^{th} step result generated from the Gauss-Seidel method. The idea of the successive Overrelaxation method is to choose a value for ω that will accelerate the rate of convergence. Normally a value from 1.7 to 1.9 is chosen for equidistant orthogonal grids [Tri89]. For an optimal choice of ω, this method can converge much faster than the Gauss-Seidel method. For $\omega=1$, this method reduces back to the Gauss-Seidel method.

Unlike the stationary iterative method, the nonstationary iterative method has iteration dependent coefficients and is more difficulty to understand. The most well-known nonstationary method is the

CG method (b.2.1). CG is an extremely effective method for symmetric positive definite systems. It uses conjugate directions instead of the local gradient for going downhill to find the nearest local minimum of a function. Based on CG, other algorithms, such as the Conjugate Gradient on the Normal Equations method, BiConjugate Gradient method, and Conjugate Gradient Squared method, are developed to solve the linear equation 7.24 under different conditions.

CG ends the iteration as the residual at a step is smaller than the user's requirement. Though CG is a very efficient method to solve equation 7.24, it requires the coefficient matrix A being symmetry and positive definite, which also means that the modeling area must have symmetry boundary conditions. This prerequisite largely limited its usage. Later on, the Biconjugate Gradient method was developed to make up this shortcoming [FM84]. It allows A to be a nonsymmetric and nonsingular matrix. However, its convergence is not as stable as CG. The Conjugate Gradient Squared method has a faster convergence speed than CG under ideal conditions; but it may sometimes lead to unreliable results [FM84].

The Multigrid method (c) can be viewed as an accelerator for iterative methods. It can not solve the linear equation 7.24 alone. The Multigrid method is based on the observation of the computation process of iterative methods. One can observe that the error, ε^m, decays very fast during the iterative computation and iteration methods converge much faster on a coarse grid than on a fine grid [FM84]. Meanwhile, the computing time cost for one iteration is also much less on a coarse grid than on a fine grid. Utilizing these specialties, the multigrid method accelerates the computation by minimizing the error on all possible coarser grids other than on the original fine grid. Because it is extremely efficient when solving equation 7.24 with iteration methods, it is simply illustrated below. For easier illustration and understanding, we start with 1D problem. 2D and 3D problems can be solved similarly by directly extending the 1D solution.

When solving equation 7.24 in 1D with an iterative method, we have residual R_i^m for cell i at the m^{th} step as:

$$R_i^m = (p_{i+1}^m - 2p_i^m + p_{i-1}^m)/\Delta x^2 - b_i \tag{7.36}$$

where $\triangle x$ is the cell's edge length. Because of equation 7.27, we have a relation of the residual and the error as:

$$R_i^m = (\varepsilon_{i+1}^m - 2\varepsilon_i^m + \varepsilon_{i-1}^m)/\Delta x^2 + (p_{i+1} - 2p_i + p_{i-1})/\Delta x^2 - b_i = (\varepsilon_{i+1}^m - 2\varepsilon_i^m + \varepsilon_{i-1}^m)/\Delta x^2 \tag{7.37}$$

where p_{i+1}, p_i, and p_{i-1} are the exact solution in cell $i+1$, i, and $i-1$. In order to perform the iteration on a coarser grid, a restriction of the residual to the coarser grid with a cell spacing $\triangle X=2\triangle x$ has to be done according to the Fig. 7.13. The residual R_I^m on the coarse grid can be computed from the residual on the original fine grid by

$$R_I^m = R_{i+1}^m/4 + R_i^m/2 + R_{i-1}^m/4 = (\varepsilon_{i+2}^m - 2\varepsilon_i^m + \varepsilon_{i-2}^m)/(2\Delta x)^2 = (\varepsilon_{I+1}^m - 2\varepsilon_I^m - \varepsilon_{I-1}^m)/\Delta X^2 \tag{7.38}$$

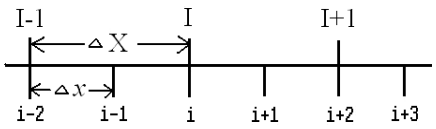


Figure 7.13: Cell spacing for a fine grid and a coarse grid in 1D.

From equation 7.38, we can observe that it has the same style as equation 7.22 in 1D. Therefore, new iterations can be executed on the coarse grid by taking R_I^m as b_i and ε_I^m as x_i. Until the residual of ε_I^* on the coarse grid reaches a required resolution, iterative computation stops and ε_I^* can be used to correct p_i so that the residual on the fine grid becomes smaller than before.

The correction of p_i on the fine grid requires ε_I^* to be prolonged onto the fine grid. As we can observe from Fig. 7.13, using linear interpolation, we get:

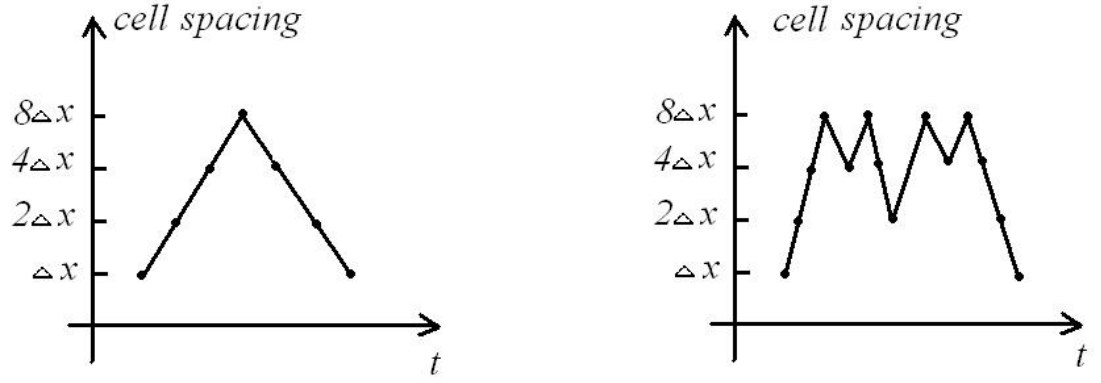
$$\varepsilon_i^* = \varepsilon_I^* \text{ and } \varepsilon_{i+1}^* = (\varepsilon_I^* + \varepsilon_{I+1}^*)/2 \tag{7.39}$$

Then applying the correction to the pressure field, we get:

$$p_i^* = p_i^m - \varepsilon_i^* \tag{7.40}$$

Similarly, it is possible to use even coarser grids with cell size like $4\triangle x$, $8\triangle x$, etc. The computation procedure of the multigrid method is very flexible. It can go back to a coarser grid again and again until the desired residual on the original grid is achieved. Fig. 7.14 shows two common computational possibilities of the multigrid method. In Fig. 7.14 (a), computation is carried out on gradually coarser grids till on a grid 8 times coarser than the original grid and then on gradually finer grids till on the original grid. In Fig. 7.14 (b), restriction and prolongation are repeated on coarser grids before the calculation goes back to the original grid.

In general, the multigrid method can accelerate the computation process when using iterative methods to solve the linear equation 7.24. The iteration on the coarser grid can provide a good correction to the computing field on the finer grid within much less computing time. Therefore, the pressure field can be computed more efficiently by solving equation 7.24 with a combination of an iteration method and the multigrid method.



(a) (b)

Figure 7.14: (a) "V"-cycle; (b) "W"-cycle of the computation grid change in the multigrid method.

(4) Turbulent Wind Field

Smoke and flames in nature are not like in a vacuum place, where their motions are determined by their own convections and diffusions. Other external disturbances, such as natural or man-made wind, influence their motions as well. Because the wind is also one kind of air flows, it has turbulent characteristics like smoke and flames. The influence of the turbulent wind on the motion of smoke and flames is described as the external force $\vec{f}$ in the incompressible Navier-Stokes equations. Then how

to describe the wind field becomes a problem, since modeling another flow is time-consuming for the visualization purpose and it is also very difficult to obtain real data through measurements.

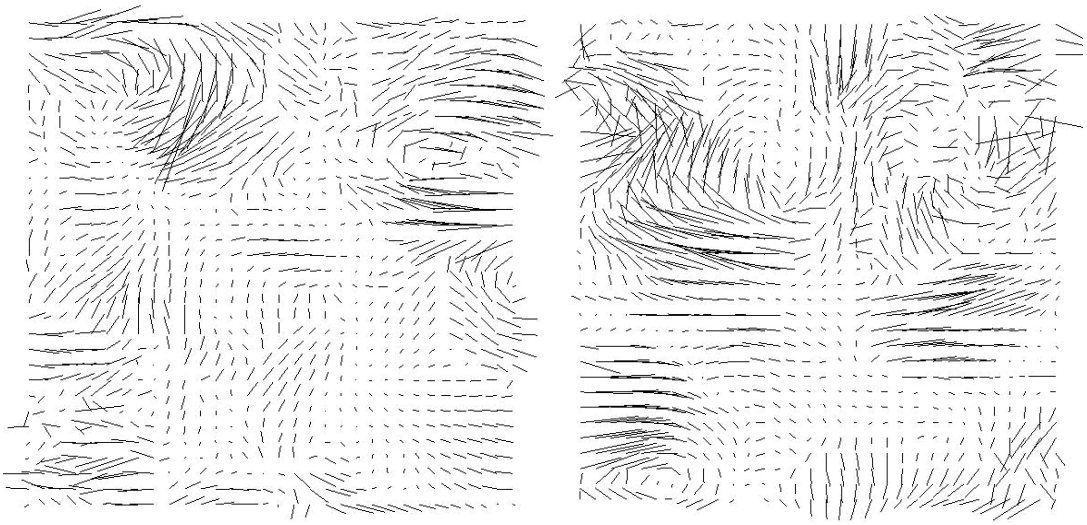
In order to save computing time and also to introduce varying and turbulent external forces to the motion of smoke and flames, this PhD work simply describes the wind field as a flowing vortical field. Initially, vortices are randomly distributed in the modeling area with zero strength. Then they are moved together with other internal particles according to the simulated velocity field; meanwhile, the norm of their strengths starts to change and they influence the flow as external forces. Accordingly, the following equation describes the way to compute the external force caused by the turbulent wind field.

$$\vec{f}_{i,j,k} = \sum_{m=1}^{n} s_m e^{-\lambda \| \vec{x}_{i,j,k} - \vec{c}_m \|} \frac{(\vec{x}_{i,j,k} - \vec{c}_m)}{\| \vec{x}_{i,j,k} - \vec{c}_m \|} \times \vec{w}_m \tag{7.41}$$

where, $\vec{f}_{i,j,k}$ is the external force in cell (*i, j, k*), *n* is the number of vortices in the wind field, s_m is the parameter to control the value of vortex *m*, λ is the parameter to control the speed of force decaying from the center of the vortex, $\vec{x}_{i,j,k}$ is the place where the external force is located in cell (*i, j, k*), $\vec{c}_m$ is the place where the center of the vortex *m* locates, and $\vec{w}_m = (wx_m, wy_m, wz_m)$ is the vortex *m*. To add more turbulence, s_m is set as a time-dependent function:

$$s_m = c_1 \sin(c_2 t) \tag{7.42}$$

where c_1 is the parameter to control the maximum size, and c_2 is the parameter to control the frequency of value change.



(a) (b)

Figure 7.15: (a) A layer of external force field resulting from a turbulent wind; (b) the same layer after 1.0 second.

According to equation 7.41, for a wind field having *n* vortices, the computation of external forces for *m* cells in a grid has the complexity of *O(mn)*. Therefore, if *m* or/and *n* is very big, the computing time cost is not ignorable. Moreover, according to equation 7.41, places that are farther away from the center of a vortex are slightly influenced by the vortex because of the exponential item. To avoid extra computation for cells far away from a vortex, a valid radius is assigned to each vortex at the beginning.

As an example, Fig. 7.15 shows a layer of an external 3D force field caused by a turbulent wind at two different times. The force field is calculated with the method described in this section. As we can observe from Fig. 7.15, though this method is very simple, it can efficiently introduce a turbulent wind field into the modeling area.

(5) Boundary Confinement

The boundary confinement deals with the motion of flows at the edge of the modeling area and the interaction between the flow and other objects. When we review the incompressible Navier-Stokes equations and their solutions discussed above, we can find that boundary confinement is a difficult and important issue since the computation of the flow's properties in one cell is always related to its other neighbors. The boundary conditions are responsible to tell us how we can deal with those neighboring cells which are not occupied by the flow.

For the edge of the modeling area, if the flow has not any interaction with other objects and the modeling area is large enough to cover the main activity of the flow, repetitive boundary condition is usually used. As hinted by its name, repetitive boundary condition defines that the flow's properties are repetitive from left to right, bottom to top, front to back, and vice verse; as shown in Fig. 7.16 in 2D.

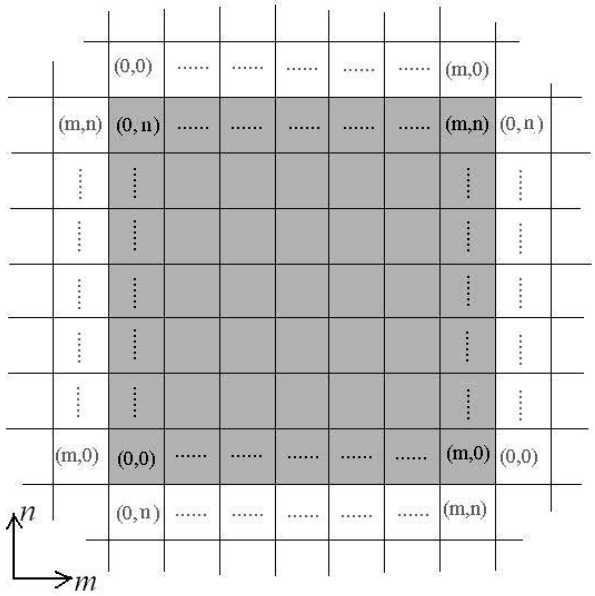


Figure 7.16: 2D repetitive boundary condition.

For the boundaries where the flow interacts with objects, Dirichlet boundary conditions and Neumann boundary conditions are commonly used. Foster and Metaxas used a Dirichlet boundary condition to model the motion of turbulent gas [FM97]. They directly set the value of flow's velocity and temperature on the boundary by assuming that the normal velocity at the non-penetrable boundary faces equals to zero, the tangential velocity at the boundary faces equals to the tangential velocity of its adjacent gas-occupied cell. Because the velocity at any place in the modeling field is the linear interpolation of velocities in its neighboring cells, internal particles close to the boundary hence have small velocities in the normal direction. Therefore, this boundary condition may produce artificial motion at the place which is close to the boundary. To make up this flaw, the method of 'ghost cells' is used as a remedy. Ghost cells are the cells which are in the object-occupied side and adjacent to flow-occupied cells. The name 'ghost' rises from that they are assumed to have the same flow properties as their adjacent flow-occupied cells during the modeling process. This assumption

postpones the decrease of velocity in the normal direction and hence eliminates the artificial motion resulting from Dirichlet boundary conditions.

Fedikw et al. utilized a Neumann boundary condition by setting the normal pressure gradient to zero at the boundary faces [FSJ01]. Meanwhile, they also utilized ghost cells to ensure that the density and velocity will not suddenly change at the boundary surfaces. Though the boundary condition they used, even without considering the assumption of ghost cells, cannot ensure the normal velocity at the boundary faces to be zero and hence makes the mass no longer conservation, it can produce visually realistic results.

In general, no matter which boundary condition is used in the modeling process, internal particles of smoke or flames need to be bounced back from the non-penetrable boundary. Mirror reflection is normally used for its simplicity and capability of conserving momentum.

(6) Vorticity Confinement

The goal of vortex confinement is to add vortical and turbulent motions back to the simulated steady flow. If the incompressible Navier-Stokes equations are numerically solved in a continuous modeling area, the turbulent property of fluid flow can be preserved. However, it is impractical since computational resources are limited and computing time is one of our primary concerns in visualization [FSJ01]. Therefore, numerical modeling is performed discretely on coarse equidistant orthogonal grids in computer graphics. Consequently, rolling and swirling motions of the simulated flow are damped out gradually if we ignore external interferences such as turbulent wind fields

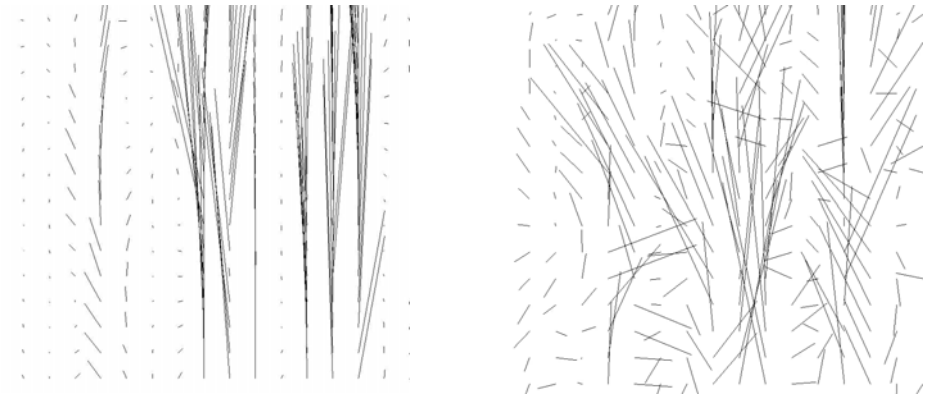
Thanks to the work of Steinhoff and Underhill, a vorticity confinement method was developed to add physically plausible small-scale turbulent motions back to the artificially steady flow, which one cannot computationally afford to add enough cells for accurate simulation [SU94]. Their original idea is to exaggerate the existing vorticity, so that the vorticity confinement method creates external forces according to simulated vorticities at each time step. The computation of external forces is described by the following equation.

$$f_{confinement} = c_1 c_2 \left(\frac{\nabla |\vec{w}|}{\left\| \nabla |\vec{w}| \right\|} \times \vec{w} \right) \tag{7.43}$$

where, c_1 is a parameter to control the amount of external forces added back into the simulated flow field, c_2 is a parameter related to the cell size of a grid and ensures that the application of this method is valid when the grid is dynamically adjustable, and $\vec{w}$ is the vortex of the velocity field on the grid, defined by equation 2.5.

The vorticity confinement method can efficiently introduce turbulent motions to the flow without too much computational costs. Fig. 7.17 shows the comparison of a flow's velocity field with and without using this method. We can observe more turbulence in the velocity field simulated with the vorticity confinement method as shown in Fig. 7.17 (b). Because this method adds additional external forces to all the cells in the grid at each time step, the turbulent motion will not be damped out by the numerical simulation on any coarse grids. Hence, parameter ε must be carefully chosen in order to avoid the accumulation of external force leading the modeling process to a divergent end.

Since the vorticity confinement method is based on the originally simulated vortex, it can not improve the motion of a flow which is too steady to have any vortices. Therefore, we can not choose a too coarse grid for the modeling because the computation may damp out all the vortices from the velocity field.



(a) (b)

Figure 7.17: (a) A velocity field without vorticity confinement; (b) the same velocity field after added vorticity confinement.

(7) Runge-Kutta methods

Runge-Kutta methods are an important family of iteration methods for solving ordinary differential equations[22] by using several trial steps in an interval to cancel out lower-order error terms[23]. When we solve the incompressible Navier-Stokes equations, we discretize them not only in space (on grids) but also in time, as shown in equation 7.16 and 7.19. Consequently, the simulated velocity field at one time step is resulted from the velocity field at the last time step, namely

$$\vec{U}^{t+\Delta t} = \vec{U}^{t} + \Delta \vec{U}_{t}^{t+\Delta t} \tag{7.44}$$

$$\Delta \vec{U}_{t}^{t+\Delta t} = \Delta t(-\vec{U}^{t} \bullet \nabla \vec{U}^{t} - \nabla P^{t} / \rho + \upsilon \nabla^{2} \vec{U}^{t} + \vec{f}^{t}) \tag{7.45}$$

Accordingly, the direct computation takes $\vec{U}^{t}$ as a constant velocity field during the time from t to $\triangle t$. This assumption introduces computational errors at each time step and their accumulation may lead to a divergent computational result in a few steps.

Runge-Kutta methods were developed by the German Mathematicians C. Runge and M.W. Kutta to get rid of this kind of computational error. According to the time sections, n, chosen to divide the interval $\triangle t$, it is called n^{th} order Runge-Kutta method. Generally speaking, the third order or fourth order Runge-Kutta method is usually used because of the amount of error they can eliminate in a reasonable time. As an example, the fourth order Runge-Kutta method is introduced in this section to solve equation 7.44 and 7.45 more accurately. The other Runge-Kutta methods are similar and can be selected according to the user's requirement based on the computing time and accuracy.

Using the fourth order Runge-Kutta method to solve equation 7.44 and 7.45, we have the following equation:

$$\vec{U}^{t+\Delta t} = \vec{U}^{t} + \frac{\Delta t}{6}(k1 + 2k2 + 2k3 + k4) \tag{7.46}$$

[22] From http://en.wikipedia.org/wiki/Runge-Kutta

[23] From http://mathworld.wolfram.com/Runge-KuttaMethod.html

where
$$k1 = -\vec{U}^t \bullet \nabla \vec{U}^t - \nabla P^t / \rho + \upsilon \nabla^2 \vec{U}^t + \vec{f}^t \tag{7.47}$$

$$k2 = -(\vec{U}^t + \frac{\Delta t}{2} k1) \bullet \nabla (\vec{U}^t + \frac{\Delta t}{2} k1) - \nabla P^{t+\Delta t/2} / \rho + \upsilon \nabla^2 (\vec{U}^t + \frac{\Delta t}{2} k1) + \vec{f}^{t+\Delta t/2} \tag{7.48}$$

$$k3 = -(\vec{U}^t + \frac{\Delta t}{2} k2) \bullet \nabla (\vec{U}^t + \frac{\Delta t}{2} k2) - \nabla P^{t+\Delta t/2} / \rho + \upsilon \nabla^2 (\vec{U}^t + \frac{\Delta t}{2} k2) + \vec{f}^{t+\Delta t/2} \tag{7.49}$$

$$k4 = -(\vec{U}^t + \Delta t \cdot k3) \bullet \nabla (\vec{U}^t + \Delta t \cdot k3) - \nabla P^{t+\Delta t} / \rho + \upsilon \nabla^2 (\vec{U}^t + \Delta t \cdot k3) + \vec{f}^{t+\Delta t} \tag{7.50}$$

The Runge-Kutta methods generate values based on several slopes (tangential directions) instead of one slope. Therefore, they can produce the computational result more close to the exact solution. As an example, the basic idea of the fourth order Runge-Kutta method is illustrated in Fig. 7.18 in 2D. *y(t1+△t)* is the exact solution of *y* after time *△t* from *y(t1)*. *y*(t1+△t)* is the value calculated with the fourth order Runge-Kutta method and *y^(t1+△t)* is the value calculated with the normal gradient method. We can easily find from Fig. 7.18 that the value calculated with the fourth order Runge-Kutta method is closer to the exact solution.

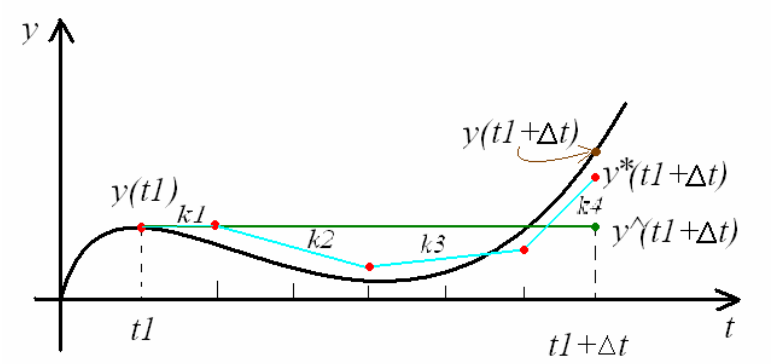


Figure 7.18: 2D illustration of solving an ordinary differential equation with the fourth order Runge-Kutta method and the normal gradient method.

(8) Spectral Method

Comparing to the finite differential method (FD), the primary advantage of a spectral method is that spatial derivatives can be more accurately evaluated with the aid of Fourier transform. Hence, the velocity field of the flow can be more easily computed in the Fourier domain. By applying the Fourier transform on the mass conservation equation in the incompressible Navier-Stokes equations, we get:

$$ix\hat{u} + iy\hat{v} + iz\hat{w} = 0 \tag{7.51}$$

where $i^2 = -1$, and $(\hat{u}, \hat{v}, \hat{w})$ is the Fourier transform of the velocity field $\vec{U} = (u, v, w)$. Equation 7.51 simply says that in the Fourier domain, the wavenumber is perpendicular to the velocity of an incompressible and mass conservation flow. The computation of equation 7.51 is quite simple when using the fast Fourier transform on the velocity field. Therefore, instead of computing the time-consuming pressure correction item to ensure the mass conservation, Stam utilized this property to correct the velocity field in his work [Sta99, Sta01, and Sta03a].

Similarly, the convection item in *x* direction (equation 7.11) can be computed by:

$$-(u\partial u/\partial x + v\partial u/\partial y + w\partial u/\partial z) = -(uF^{-1}(ix\hat{u}) + vF^{-1}(iy\hat{u}) + wF^{-1}(iz\hat{u})) \quad (7.52)$$

where $F^{-1}(x)$ is the inverse Fourier transform; the diffusion item in x direction (equation 7.18) can be computed by:

$$\upsilon(\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 + \partial^2 u/\partial z^2) = \upsilon F^{-1}(-(x^2+y^2+z^2)\hat{u}) \quad (7.53)$$

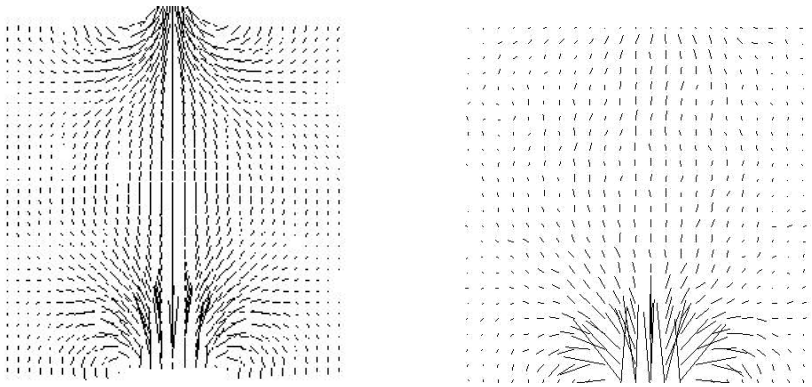
The convection and diffusion item in y and z directions can be computed similarly. According to equation 7.21, the pressure can be computed by:

$$P = -\rho F^{-1}((ix\hat{\psi}_x + iy\hat{\psi}_y + iz\hat{\psi}_z + ix\hat{f}_x + iy\hat{f}_y + iz\hat{f}_z)/(x^2+y^2+z^2)) \quad (7.54)$$

where $(\hat{\psi}_x, \hat{\psi}_y, \hat{\psi}_z)$ is the Fourier transform of convection item and $(\hat{f}_x, \hat{f}_y, \hat{f}_z)$ is the Fourier transform of the external force.

We can observe from equation 7.51 to 7.54 that the incompressible Navier-Stokes equations can be simply solved in the Fourier domain without the time-consuming and easily divergent computation of non-linear items. However, a vital limitation of spectral methods is that Fourier transforms only work on periodic boundary conditions. For example, if we simulate smoke rising from the bottom of a grid in an open boundary condition directly with a spectral method, the velocities on the top will be automatically distorted to fit the bottom.

A simple solution we can apply is to double the grid (modeling area) in these heavily distorted directions. After the computation on velocity field at time step, only some part of the values are used as the final velocity field, while the values in the remaining area are set to zero to mimic the external infinite space at the end of every cycle. Though this method causes unavoidable errors, it efficiently removes the periodic artifact from velocity fields and keeps the solution of the Navier-Stokes equations convergent. Fig. 7.19 shows a velocity field with and without doubling the modeling area in 2D. We can observe that the periodic artifacts at the top of the grid, shown in Fig. 7.19 (a), have been eliminated by doubling the modeling area in y direction. The final usable velocity field is shown in Fig. 7.19 (b).



(a) (b)

Figure 7.19: (a) A velocity field computed by the spectral method; (b) the same velocity field after doubling the modeling area in the y direction and omitting the upper part in use.

In most cases, spectral methods cannot be applied directly to modeling a flow which interacts with other objects because the additional boundaries introduced by those objects do not fit the periodic boundary requirements of the Fourier transform. Zero Dirichlet boundary conditions can be used to solve this problem approximately. Because we use linear interpolation to obtain a velocity in velocity fields, two inverse velocity vectors at the centers of two adjacent cells will generate a zero velocity at their middle point. Therefore, in each cycle we can first update a velocity field, *A*, without considering inner objects. Then a new velocity field, *B*, is constructed by setting inverse velocities to the object-occupied cells and zero velocities to the other cells. When we update *B* through the same time interval and add it to *A*. After projecting the resulted velocity field to a plane perpendicularly to its wavenumber for mass conservation, described in equation 7.51, we can obtain the final velocity field in this cycle. Similar as the Dirichlet boundary used by Foster and Metaxas, this boundary condition may result in slow movements along boundary's normal directions though it can better conserve mass and momentum.

In general, the spectral method avoids using time-consuming and easily divergent iterative solvers in other numerical computation methods. It is extremely suitable to calculate the velocity field of a flow with open boundary conditions.

7.1.2 Large Eddy Simulation

In the computational fluid dynamics (CFD) for engineering applications, turbulent flows, characterized by highly unsteady behaviors in 3D, are investigated numerically with various methods, which can be divided into three categories: Direct Numerical Simulation (DNS), Reynolds Averaged Navier-Stokes (RANS) and Large Eddy Simulation (LES).

As discussed in the section 7.1.3.1 to 7.1.3.7, DNS is a straightforward method and its results are more close to exact solutions. With various degrees of simplification, it is commonly used to visualize dynamic smoke in the computer graphics area and already obtained promising results [FM97, FSJ01, and Sta99]. However, visually realistic smoke visualization need to apply DNS on a fine grid with very short interval to ensure the computational convergence. It is time-consuming for the purpose of visualization. As hinted by its name, RANS simulation only calculates statistical quantities such as mean flow properties. It neglects the small scale turbulent properties of smoke and results in artificially stable flows. The time cost and accuracy of LES lies between the above two methods. Comparing to RANS, LES provides rather accurate results, which can maintain vortices in velocity fields simulated on coarse grids. Therefore, a modified LES method is introduced by this PhD work to model and simulate dynamic smoke for the visualization purpose.

The idea of LES is to split the velocity field of a flow into two parts, a spatially local mean, $\bar{U}_{mean}$, and a fluctuation about that mean, $\bar{U}'$, [Lay02]. The mean velocity field dominates the motion of the flow to a large extent and hence needs to be calculated precisely. To display the swirling and rolling properties, the fluctuation also needs to be calculated in each step to stir the internal particles of the flow. The main advantage of LES is that the turbulent motion of the flow can be conserved when modeling is carried out on a coarse grid. To fast and efficiently simulate mean velocity fields, the spectral method, discussed in the section 7.1.3.8, is used to solve the incompressible Navier-Stokes equations.

According to the definition of LES, since $\bar{U}'$ is fluctuating and has a random character, its average effect is also reflected by the mean velocity. $\bar{U}_{mean}$ is generally much more energetic than $\bar{U}'$; and its size and strength make it to be the most effective transporter of the conserved properties. $\bar{U}_{mean}$ can be obtained by filtering the velocity field $\bar{U}$ with a kernel $g(\vec{x})$, namely

$$\vec{U}_{mean} = g(\vec{x}) \otimes \vec{U} \tag{7.55}$$

By applying the filter kernel to the Navier-Stokes equations, we have:

$$\nabla \bullet \vec{U}_{mean} = 0 \tag{7.56}$$

$$\partial \vec{U}_{mean} / \partial t + \nabla \bullet (\vec{U}_{mean} \vec{U}_{mean}) + \nabla P / \rho = \upsilon \nabla^2 \vec{U}_{mean} - \nabla \bullet ((\vec{U}\vec{U})_{mean} - \vec{U}_{mean} \vec{U}_{mean}) + \vec{f} \tag{7.57}$$

It is important to note that $(\vec{U}\vec{U})_{mean} \neq \vec{U}_{mean} \vec{U}_{mean}$. The item $(\vec{U}\vec{U})_{mean} - \vec{U}_{mean} \vec{U}_{mean}$ is called Reynolds stress tensor, which is in fact the large scale momentum flux caused by the action of the small or unresolved scales and not easy to simulate accurately. There exist many approximate models of Reynolds stress tensor [FP02, Lay02]. Since the main purpose of our simulation is visualization instead of engineering application, and smoke and flames naturally undergo complicated disturbances from the atmosphere, for example smoke ejected from a chimney interacting with the surrounding flowing air, we collect all the unresolved influential elements together with the Reynolds stress tensor to form a new item $\vec{\Re}$, which is simply modeled as a white noise to save computing time. Then, equation 7.57 becomes:

$$\partial \vec{U}_{mean} / \partial t + \nabla \bullet (\vec{U}_{mean} \vec{U}_{mean}) + \nabla P / \rho = \upsilon \nabla^2 \vec{U}_{mean} + \vec{f} - \vec{\Re} \tag{7.58}$$

and the pressure *P* can be computed by:

$$\nabla^2 P = -\rho \nabla \bullet (\nabla \bullet (\vec{U}_{mean} \vec{U}_{mean})) + \rho \nabla \bullet (\vec{f} + \vec{\Re}) \tag{7.59}$$

Besides the thermal buoyancy and gravity of the flow, which interact with the velocity field, other external forces such as turbulent wind field should be added to $\vec{f}$ as well. However, if the external force is too strong or too turbulent, the numerical solution of the velocity is instable and easily divergent because under this condition it is the external force which dominates the velocity field of the flow.

Based on the mean velocity field $\vec{U}_{mean}$ defined in equation 7.56 and 7.57, we can simulate the small-scale swirling and rolling properties of the flow. According to the definition of LES, the fluctuation item is $\vec{U}' = \vec{U} - \vec{U}_{mean}$. Applying Fourier transform, *F(x)*, to both sides of the equation, we have:

$$F(\vec{U}') = F(\vec{U}) - F(g) F(\vec{U}) \tag{7.60}$$

Then,

$$F(\vec{U}') = (1/F(g) - 1) F(\vec{U}_{mean}) \tag{7.61}$$

For the filter $g(\vec{x})$, a low pass Gaussian kernel is selected for its smooth properties:

$$g(\vec{x}) = e^{-\vec{x}^2 / (2\sigma_g{}^2)} \tag{7.62}$$

where σ_g is standard deviation. Then the Fourier transform of $g(\vec{x})$ is:

$$F(g) = e^{-2\pi^2 \sigma_g{}^2 |\vec{w}_q|^2} \tag{7.63}$$

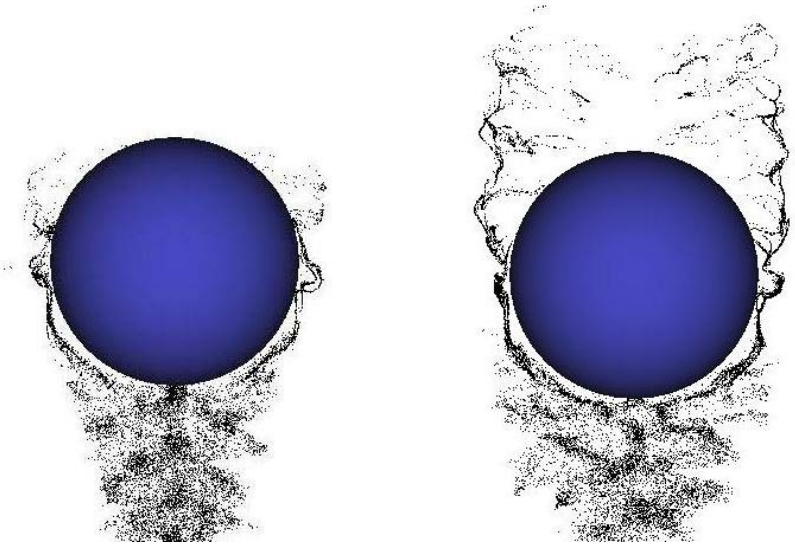
where $\vec{w}_q$ is the wavenumber in the Fourier domain. Because the wavenumber is limited to the grid size when using the fast Fourier transform, we can apply [0/1] Padé approximation on *F(g)* to accelerate the simulation. Then,

$$F(g) = 1/(1 + 2\pi^2 \sigma_g{}^2 |\vec{w}_q|^2) \tag{7.64}$$

According to equation 7.61, we have:

$$F(\vec{U}') = 2c\pi^2 \sigma_g{}^2 |\vec{w}_q|^2 F(\vec{U}_{mean}) \tag{7.65}$$

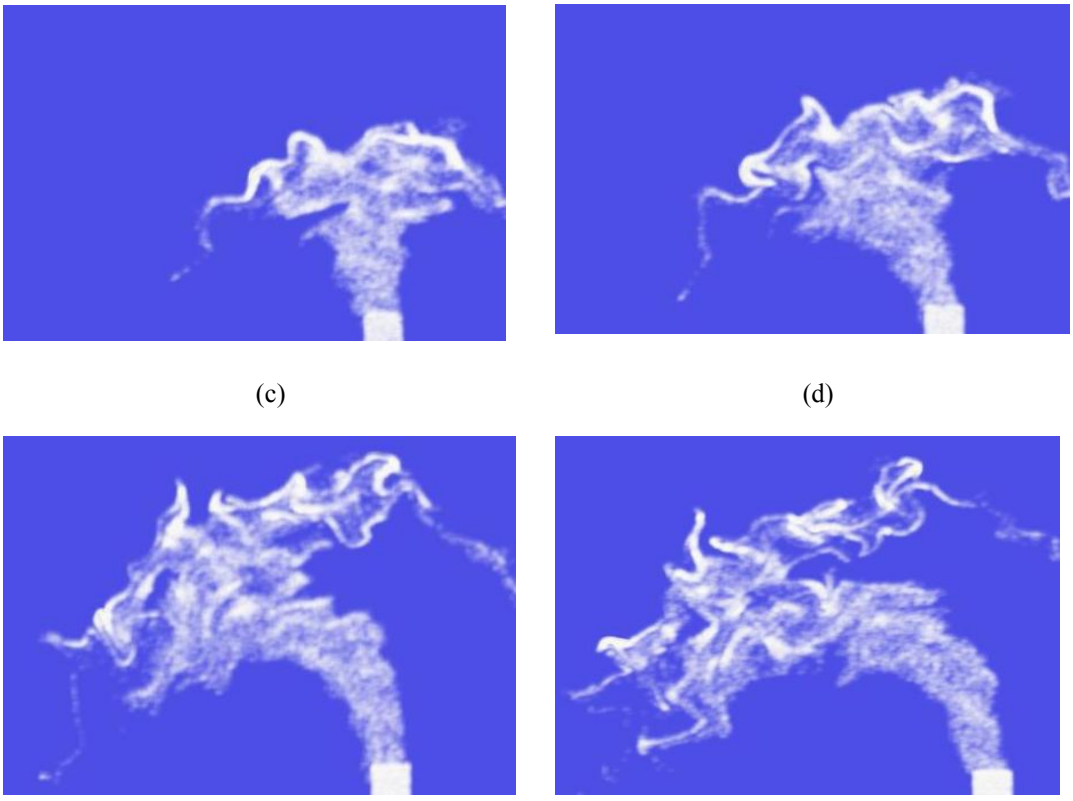
where c is an adjustable parameter proportionally to the cell spacing, Δx, of a grid. Therefore, the velocity field can be obtained by solving equation 7.58, 7.59, and 7.65 at each time step. Modeled with this large eddy simulation method, Fig. 7.20 shows the internal particles of smoke interact with a sphere in time sequence in 2D. Particles are simply bounced back from the surface of the sphere. Fig. 7.21 shows smoke under a sudden wind in time sequence simulated by this method in 2D. The time for modeling the velocity field is only 0.015 second per frame with a Pentium4-2.4GHz CPU. We can observe from these two figures that the modeling process can still be kept convergent when the velocity field is disturbed by sudden external forces. Moreover, the swirling and rolling properties of smoke are conserved though the applied grid only has a size of 32×32.



(a) (b)

Figure 7.20: The particles of smoke interact with a sphere in time sequence simulated by the LES method.

(a) (b)



(c) (d)

(e) (f)

Figure 7.21: Smoke influenced by sudden external forces in time sequence simulated by the LES method in 2D.

Numerical stability is an important issue in validating a model and its simulation algorithm. With the LES method described above, instability occurs when the simulation time-interval Δt or fluctuation factor c is too big, aside from the external forces. From equation 7.58, we can find that if Δt is too big, the computational error will be quickly accumulated and cause velocity fields diverging to infinite values. Moreover, it is meaningless to simulate a velocity field with a long interval, since the changes of velocity fields are too coarse to reflect the exact motion of the flow. Hence, to make a modeling process stable and conserve the turbulent motion of the flow, Δt must be set according to:

$$\Delta t \le \min(\Delta x/(m\upsilon), \Delta x/|\vec{U}|) \tag{7.66}$$

where m^2 is the cell number of the applied grid. The factor c controls the magnitude of a fluctuant velocity field. If c is too big, the mean velocity fields will be distorted unreasonably heavy by the fluctuation. A noticeable negative result is the final velocity field has obvious oscillation. Therefore, according to our experiments, c should be set according to:

$$c \le 0.02/\left(\pi^2 \sigma_g{}^2\right) \tag{7.67}$$

Because of the numerical accuracy of the spectral method in the computation of derivatives, the kinematic viscosity has no lower bound, which is crucial for the system to maintain stability in the work of [FM97].

Though this method obtains promising results in 2D in real-time, its 3D results are not so attractive. The computing time for each frame in 3D is still satisfying, around 0.172 second with a Pentium4-2.4GHz CPU. However, the velocity are damped down so much that the motion of the flow becomes unreasonable slow. Fig. 7.22 shows a layer of the simulated 3D velocity field. Comparing to the 2D simulation result in Fig. 7.19 (b), the velocity decreases too fast from the flow's inlet which is located at the bottom center part of the grid.

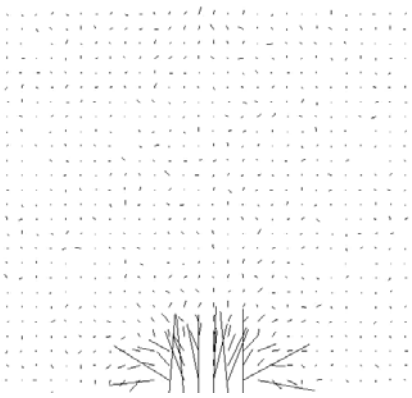


Figure 7.22: A layer of the simulated 3D velocity field with the LES method.

Based on the former discussion about the physically based modeling methods, we can find that they can hardly result in a turbulent flow unless other external forces are intentionally added. Further detailed discussion on this topic can be found in section 7.4. Consequently, this PhD thesis develops another efficient method for 4D real-time turbulent flow simulation. This method treats internal particles as objects and can generate visually convincing vortical flows with small computation costs. It belongs to the compromised modeling method. However, since it is more flexible and efficient than the other compromised modeling methods and it is developed alone in this thesis, it is introduced separately in the next section.

7.1.3 Particle-based Modeling Method

As we discussed in the section 7.1.1.3, numerically solving the incompressible Navier-Stokes equations on grids to model the velocity field of flows is normally time-consuming and easily divergent. The worst case is that it is hardly capable of creating a turbulent flow without a turbulent wind field, which is used to add vortically disturbing forces to the velocity field, and the vorticity confinement method, which is also developed to add more vortices to the velocity field through the external force. The idea of this particle-based modeling method is to take the motion and physical properties of individual particle into consideration, accompanied by their average velocity field on grids.

As we discussed in section 5.1, particle systems are commonly used to model amorphous phenomena because they can generate continuously and smoothly photorealistic images through blending. Internal particles of smoke and flames have different sizes, velocities, and other physical properties such as temperatures and densities since each particle represents a group of real particles in the modeling flow. The formal discussed physically based modeling methods only take their average properties on grids into consideration and their velocities are obtained from the simulated velocity field by linear interpolation. Therefore, for an individual particle, its unique motion and other properties are swept off and replaced by interpolated values, which decreases the turbulence of flow since neighboring particles now have similar properties. Unless a grid is divided fine enough to contain only one particle

in one cell, the simulation on the grid cannot model the motion of flow based on particle systems exactly.

Furthermore, the computing time cost of the formal physically based modeling methods equals to the time cost of updating each particle to a new position with a complexity of $O(n)$ plus the simulation time cost on grids, which normally takes more time than updating particle positions and depends on environmental conditions and grid size. They are not capable of modeling a turbulent flow in real-time with a current common computational resource.

However, if we purely consider the interactions of particles in a microscopic state, we can imagine it is too time-consuming to afford in visualization. For example, if n particles are used to model a flow, the computational complexity is $O(n^n)$. Even with some simplified methods like the Barnes-Hut algorithm [BH86] and the Fast Multipole Method [CGR88, And92], the computational complexity is no less than $O(nlog(n))$ without considering the time to construct a tree data structure for the n internal particles at each time step.

Therefore, the particle-based modeling method developed in this PhD thesis takes the advantages of both sides, computing some average properties on grids and keeping unique properties for individuals. The computation complexity of this method is $O(m^3+n)$, where m^3 is the size of grids in 3D and its value is much smaller than n. We can observe from the complexities that this particle-based modeling method is much faster than the others and its results, which will be shown later in this section, also prove that it is fast enough for modeling a turbulent flow in real-time. Moreover, this method takes a special property, which has never been treated by the former methods, into consideration. That is, every individual internal particle is expansive and has a lifetime because it represents a group of real particles. When it expands to a certain degree or cool down to a temperature the same as its surrounding air, it will disappear from the view.

To better understand the particle-based method, let us first investigate the initial condition of smoke and flames. The internal particles, no matter from smoke or flames, are ejected from somewhere or changed from other kinds of particles with initial positions, velocities, temperatures, lifetimes, and sizes. Though they may have similar properties, such as the same lifetime, their initial velocities are diversities. For example, smoke from a chimney is already influenced by the boundary of the chimney and contains vortices initially. Therefore, some particles are treated as vortex sources and given vortical strengths initially. Moreover, the most remarkable motion of the smoke and flames is to rise under the effect of the thermal buoyancy. Hence, the velocity in the upward direction is saved separately. Because of the close relationship between the lifetime and temperature, we only need to store one of them to represent the change from a particle of the modeling flow to a particle of the air. Accordingly, we can initialize internal particles of the modeling flow as:

$$p_i = \{(x_i, y_i, z_i), life_i, \sigma_i, up_i\}$$

where, p_i is the i^{th} particle, (x_i,y_i,z_i) is the position of p_i, $life_i$ is the lifetime of p_i, σ_i is the radius of p_i, and up_i is the upward velocity of p_i. The vortices which are attached to some internal particles can be initialized as:

$$W_j = \{p_{(j+r_j)\cdot vorInterval}, (w_x, w_y, w_z)\}$$

where, W_j is the j^{th} vortex source and is attached to the $((j+r_j)vorInterval)^{th}$ particle, r_j is a random float number between 0 and 1, $vorInterval$ is the average number of particles which contain one vortex source, and (w_x, w_y, w_z) is the vortex vector of V_j.

After all the internal particles of the modeling flow are released from their resources, the differences between their temperatures and the temperature of their surrounding air generate thermal buoyancies and unbalanced pressure field in the flow according to the ideal gas law. The thermal buoyancies force particles to rise in the air and the unbalanced pressure field influences the motion of the modeling flow and its interacting air. According to the initial setting of individual particle, the thermal buoyancy can be obtained individually. However, it is time-consuming to calculate the pressure field only based on particles. Therefore, grids are used to divide the modeling area into m^3 cells. The pressure field and its effect on the motion of the flow are simulated based on a 3D grid. Because the ideal gas law states that

$$P = \rho kT / m_c \tag{7.68}$$

where P is the pressure, ρ is the density of gas, k is the Boltzmann's constant, T is the gas temperature, and m_c is the molecular mass, we can obtain $P_{i,j,k}$ in the cell *(i,j,k)* from:

$$P_{i,j,k} = \eta \cdot \rho_{i,j,k} \cdot T_{i,j,k} \tag{7.69}$$

where η is a constant related to the material of the flow, $\rho_{i,j,k}$ and $T_{i,j,k}$ are the average density and temperature of the flow in the cell *(i,j,k)*. Here, $\rho_{i,j,k}$ can be obtained from the number of internal particles and $T_{i,j,k}$ can be obtained by the average temperature (the lifetime) of internal particles in the cell. Then we can calculate the effect of the pressure on the motion of the flow approximately by assuming the pressure differences as the externally forces to cells, namely

$$\partial \vec{U} / \partial t = \eta \cdot c_1 \cdot \nabla P \tag{7.70}$$

where $\vec{U}$ is the velocity field caused by the unbalanced pressure field, and c_1 is a constant to control the velocity changes. Therefore, the velocity of any internal particle is determined by the velocity field $\vec{U}$, diffusion and the velocity caused by its thermal buoyancy and gravity, which can be computed as the following equation.

$$\partial up_i / \partial t = c_2(life_i - g) \tag{7.71}$$

where c_2 is a constant to control the amount of buoyancy and gravity, g is a constant related to the gravity of a particle, and particles are assumed to have identical gravity as $c_2 \bullet g$ to simplify the computation.

Because of the thermal conduction, convection, and radiation, the temperatures of flow's internal particles fall down gradually to the temperature of their surrounding air by transferring thermal energy to neighboring air continuously. Accordingly, the lifetimes of internal particles decrease together with their temperatures and the time they exist in the modeling flow. Here, the decrease of lifetime approximates thermal lose during thermal exchanges and expansions of particles. After the lifetime of a particle decreases to zero, it turns into a particle of the air and disappears from the view. Because the particle's lifetime is related to its surrounding temperatures, the lifetime change can be described as:

$$\partial life_i / \partial t = \max(\xi(T_{new} - T_{old}) - loss, 0.0) \tag{7.72}$$

where ξ is the thermal exchange parameter, T_{new} is the average temperature of the cell where particle i is at the new time step, T_{old} is the temperature of the cell where particle i is at the current time step, and *loss* is the ratio of thermal dissipation. Accordingly, $T_{i,j,k}$ in equation 7.69 can be updated by:

$$\partial T_{i,j,k} / \partial t = \xi(\sum_{h=1}^{f} life_h / m_p - T_{i,j,k}) \tag{7.73}$$

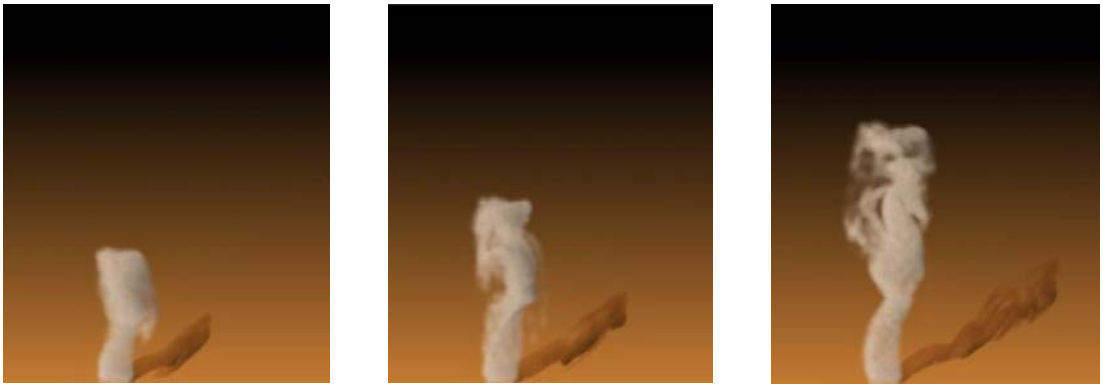
where m_p is the number of particles, which are moved to the cell *(i,j,k)* according to the calculated velocity at the current time step. Assuming the density of the air is equal to 1.0 for easy computation, $\rho_{i,j,k}$ can be updated by:

$$\rho_{i,j,k} = (m_p + n_t) / n_t; \tag{7.74}$$

where n_t is the number of the air's internal particles in a cell without considering the modeling flow. Please note that, because of the thermal exchange between the modeling flow and the air and the collision between particles of the flow and particles of the air, some of the flow's momentum is transferred to the air. In this condition the momentum is still conserved except that its carrier is the mixture of the modeling flow and the air.

The vortices, which are initially attached to particles, update their positions with particles. Their influence on the motion of the flow can be calculated according to equation 7.41 and 7.42 by changing *t* in equation 7.42 to the lifetime. In this method, direct bounce-back boundary condition is used for its simplicity. However, to simulate the interaction between the modeling flow and the non-penetrable boundary, when the particles, which are carrying vortices, bounce back from the boundary, their lifetimes are reset to initial values. In other words, the vortices have sudden changes at the boundary surfaces and their lifetimes are prolonged by the interactions with boundaries. Then, the finial velocity field can be corrected by the diffusion item and in the spectral field according to equation 7.51.

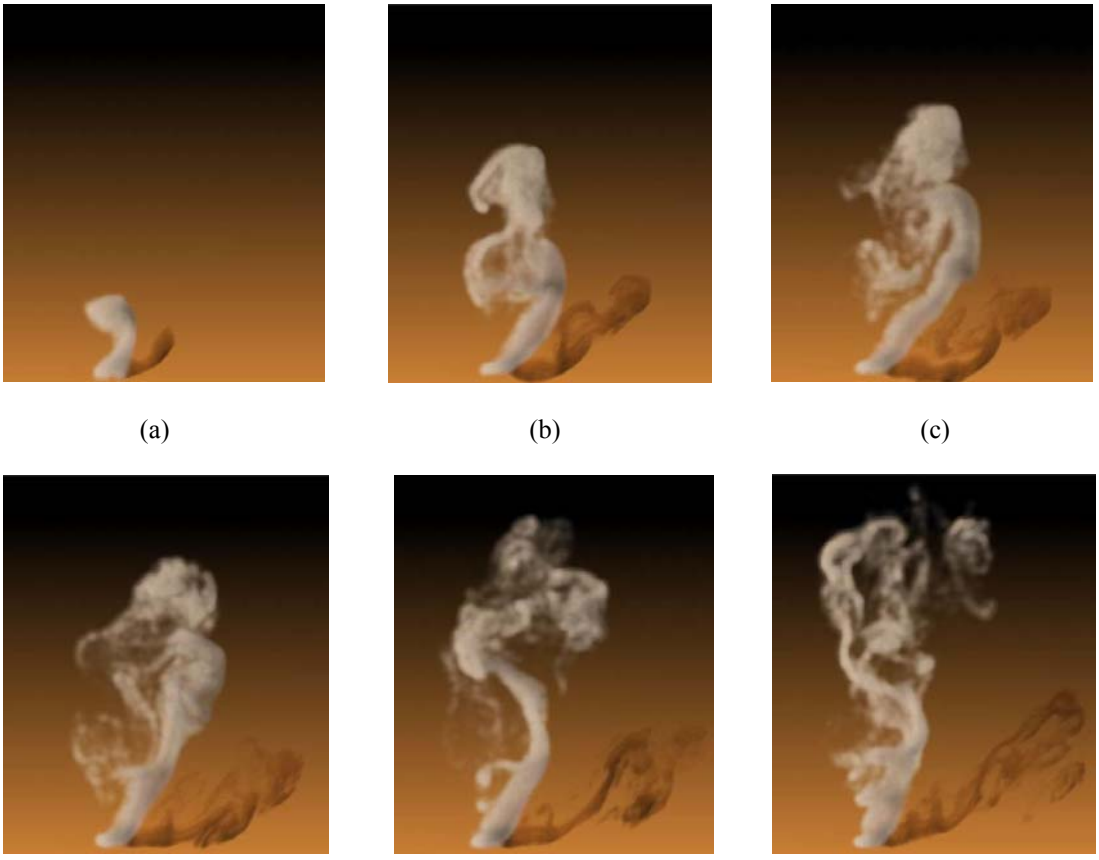
In general, this particle-based modeling method is stable and can produce turbulent flows in real-time, no matter whether there are other external forces or non-penetrable objects. As an example, Fig. 7.23 shows a turbulent smoke without any other interference in time sequence; Fig. 7.24 shows a turbulent smoke under the influence of a sudden wind; and Fig. 7.25 shows a turbulent smoke interacting with a solid sphere. They are all visualized in real-time.



(a) (b) (c)

(d) (e) (f)

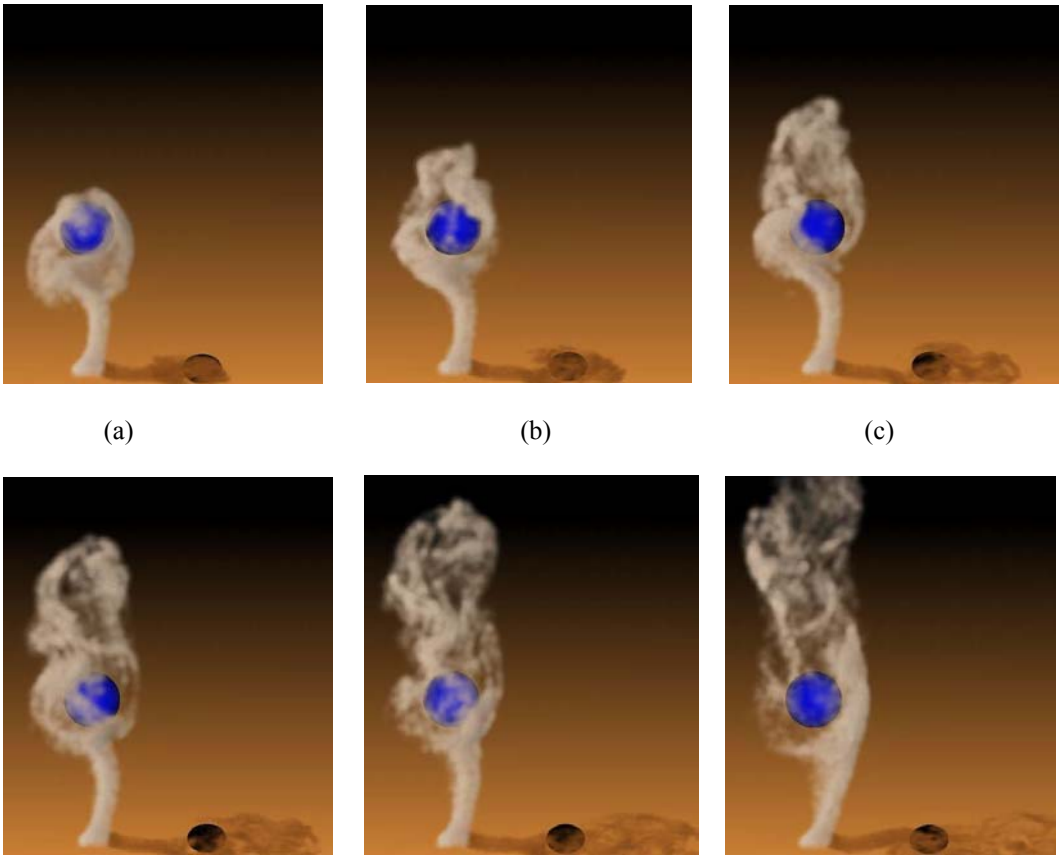
Figure 7.23: Rising smoke in time sequence.



(a) (b) (c)

(d) (e) (f)

Figure 7.24: Rising smoke under the influence of a sudden wind in time sequence.



(a) (b) (c)

(d) (e) (f)

Figure 7.25: Rising smoke interacting with a solid sphere.

7.2 Smoke Rendering

Smoke is one kind of semi-transparent amorphous phenomena with various densities and colors. To display visually convincing smoke with self-shading properties in an efficient way, the recording matrix method, which is developed in this thesis and presented in section 5.2.2, is applied to rendering smoke. The primary advantages of this method are that it only requires small temporary saving space and can perform rendering processes much faster than other methods. Smokes illuminated by the direct light sources in Fig. 7.23 to 7.25 are rendered directly with this method.

Unlike clouds, which are normally illuminated by the direct light source with parallel constant-intensity rays in the daytime, smoke may be illuminated by a spot light source at night. When smoke is illuminated by a spot light, rays traverse smoke along divergent directions and have gradually decreasing intensities within the volume of its lightened cone. Tracing rays along different directions is time-consuming and also does not match the basic idea of the recording matrix method, where the matrix is supposed to be perpendicular to the incident light direction. Inspired by the shear-warp method developed by Lacroute and Levoy [LL94], we can transform smoke models illuminated by spot light sources to models illuminated by corresponding direct light sources before the calculation of average intensities on metaballs, namely the calculation of colors of internal particles.

Unlike volume slices which require to be sheared in non-orthogonal projection [LL94], smoke is constructed by metaballs and hence can be thought as always in orthogonal projection. So we only

need to scale our models to turn perspective rays into parallel rays. In addition, since the sizes of metaballs in a model is very small comparing to the size of the whole model, the error of average intensity on each metaball results from the differences of metaball sizes before and after scaling can be ignored. Fig. 7.26 illustrates this transformation in a simple 2D manner. Since our metaballs are spheres, they are round from all view directions in 2D.

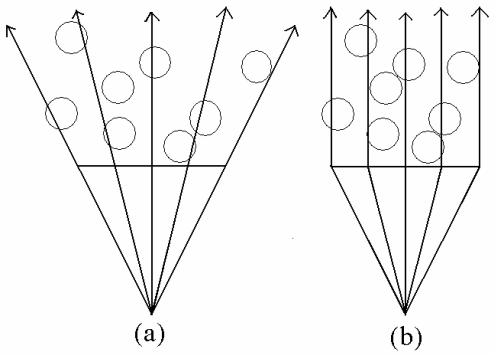
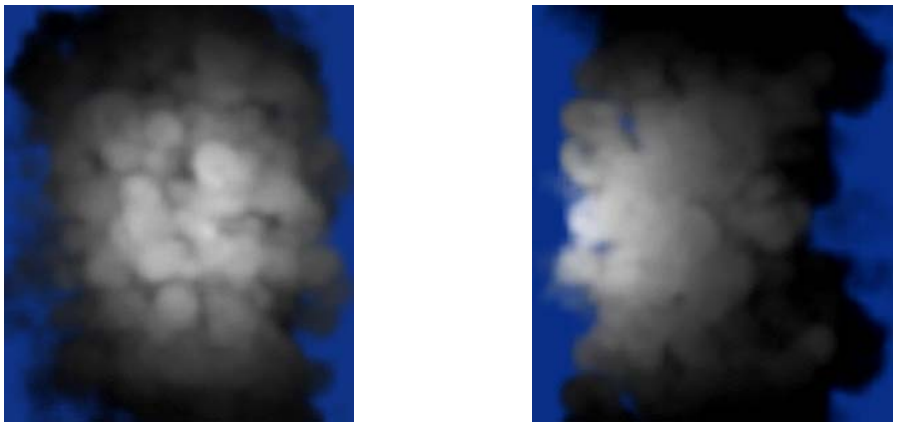


Figure 7.26: (a) Metaballs illuminated by a spot light in 2D; (b) metaballs illuminated by a corresponding direct light after the transformation.

After all the metaballs are transformed to new places, the average intensity on every metaball can be calculated by the recording matrix method. Herein, the scaling factor, introduced in section 5.2.7, plays an important role in the calculation of average intensities on metaballs. This is because light sources are normally far away from gaseous phenomena. Then after scaling, their models are usually too small to obtain reasonable illumination results. In this case, the scaling factor is used to enlarge models for higher resolution and accurate calculation results. In addition, since the spot light has decreasing intensities in its lightened cone, the elements in the matrix are initialized with a Gaussian equation. To save the computation along rays with very low intensities, we can set an additional cut-off angle for the spot light. Metaballs out of the cut-off angle can easily be detected in the former transformation process and intensities on them are directly assumed to be zero.

After the transformation, smoke illuminated by the spot light source can be rendered with the recording matrix method. As an example, Fig. 7.27 displays the results of a dense smoke illuminated by a white spot light source and viewed from two different directions.



(a) (b)

Figure 7.27: a dense smoke illuminated by a white spot light source and viewed from two different directions.

7.3 Flame Rendering

As discussed in section 3.4.2, rendering flames visually realistically is a challenging problem due to the complicated light interactions among soot and other various chemical compounds found in the fire. Depending on its temperatures, soot absorbs all the light it encountering and emits electromagnetic light in various wavelengths with various amplitudes. At the same time, other chemical compounds absorb parts of incident light on them and scatter light to all the other possible directions.

Fig. 7.28 illustrates the process as light travels through a flame. When the incident light (in orange) encounters a non-soot particle, part of it is scattered to other directions and the left part continues transferring in the forward direction. When the forward part encounters a soot particle, it is absorbed totally by the soot particle, while the soot particle also emits light (in blue) to various directions.

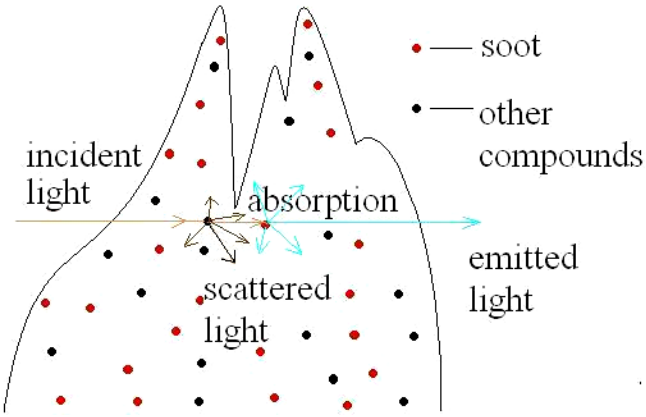


Figure 7.28: Light absorption, scattering, and emission while it traverses the flame.

The process of light absorbing and multiple scattering on other chemical compounds of flames is exact the same as the process of the light traversing the smoke and clouds. Therefore, the recording matrix method can be directly applied on those particles. The remaining issue is the light interacting with soot. Because soot has blackbody radiation under high temperatures, it absorbs all electromagnetic radiation without reflection and scattering. At the same time, it also radiates every possible wavelength of energy and the wavelength is determined by its temperature. The blackbody radiation at position x, $L_{\varphi}(x)$, can be described by the Planck's formula:

$$L_{\varphi}(x) = \frac{2c_1}{\varphi^5 (e^{c_2/(\varphi T)} - 1)} \tag{7.75}$$

where T is the temperature, φ is the wavelength, c_1 and c_2 are two positive constants according to Siegel and Howell [SH81].

For the convenience of rendering, the radiation can simply be mapped to colors [NFJ02]. Basically, accompanied with the rise of temperature, soot appears red, orange, and bluish. Therefore, to accelerate the flame rendering process, this thesis divides the light emitted by soot into three phases according to its simulated temperature. In the first phase, the temperature is not so high, the red value raises faster than the green and blue values till it reaches 90% of its saturation and the other two reach 60% and 30% of their saturations respectively. In the second phase, the green value raises fastest till it reaches 100% of its saturation, and the blue value raises faster than the red value till they reach 70% and 100% of their saturations respectively. In the third phase, only the blue value raises to 100% of its saturation while the other two drop to 90% of their saturation. Because soot emits light, the flame may

appear very bright and even white at some places. Fortunately, taking the advantage of the cut-off capability of the color buffer, we need not extra calculations when we blend the color of all the internal particles together. Namely, the value in each color channel beyond the 100% saturation will be cut off automatically in the final image rendering process.

As an example, Fig. 7.29 shows a flame rendered by this method in time sequence. The flame is modeled by the particle-based method, developed in this thesis and described in section 7.1.3. To emphasize the light emitted from soot, no incident light is used in this model.

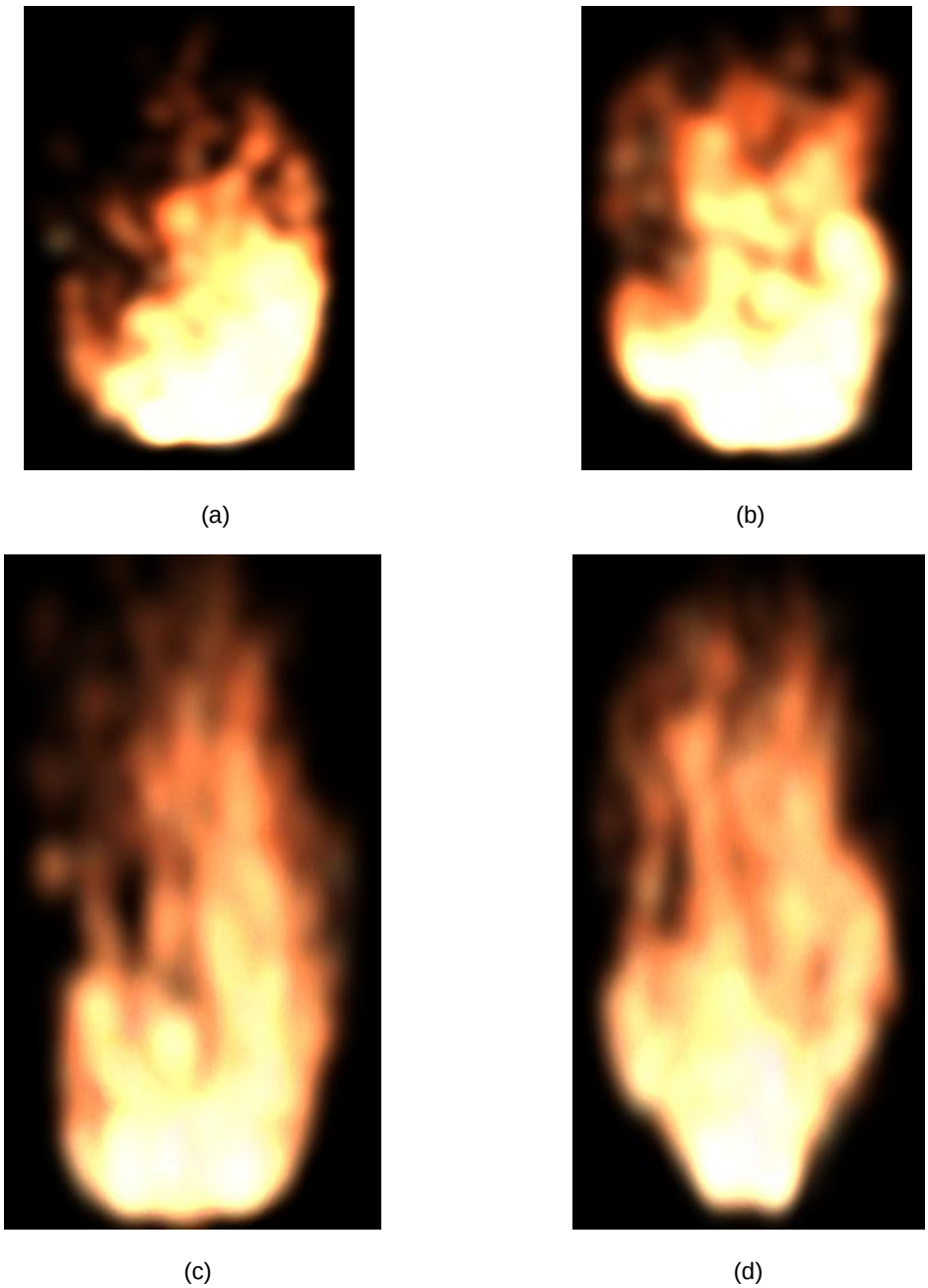


Figure 7.29: A flame visualized with the particle-based modeling method, developed in this thesis in time sequence.

7.4 Discussion

The heuristic modeling methods are not based on the physical properties of smoke and flames. Instead, they simulate the dominant motion of the flow through observation and introduce turbulence to the flow through random functions. Because they are totally based on experiences, it is hard for them to simulate gaseous phenomena in large-scales.

The compromised methods are methods between the heuristic modeling methods and the physically based methods. They are based on the physical properties of the flow and are largely simplified for fast modeling speeds. In other words, the compromised methods can achieve real-time modeling by compromising the physical accuracy. Comparing to the LBM method, the particle-based method developed in this thesis contains more visible turbulence with similar computing time cost.

The physically based methods are all based on the incompressible Navier-Stokes equations and attempt to use efficient methods to solve equations as fast as possible for the visualization purpose. Because the convection item in the Navier-Stokes equations is non-linear, computation may become divergent when using CDS to calculate it directly. The semi-Lagrangian scheme [Sta99] is useful to keep the computation convergent and ensures stable simulation of the flow's advection. However, as discussed in [PK05, SRF05], its approximate manner introduces artifacts and also sweeps off small scale details from the velocity field.

Another difficulty existing in the physically based methods is the calculation of the pressure field through a Poisson equation. It is important because the pressure field can be used to correct the velocity field and ensures the mass and momentum conservation. The combination of iterative methods and the multigrid method is usually used to solve the Poisson equation on grids. The solution is not always convergent when the boundary is not symmetric. When the boundary is symmetric, spectral methods can be used to solve the pressure and velocity fields. Based on the Fast Fourier Transform, the spectral methods cost less computing time than iterative methods and produce more stable results.

Because the physically based methods discretize the velocity field on equidistant orthogonal grids for easy and fast computation, small-scale turbulence is swept off from the velocity field during computation. The vorticity confinement method is introduced to enlarge the existing vortices in the velocity field. Parameters should be selected carefully in case the velocity field will become divergent under too strong disturbance. However, the vorticity confinement method can not generate small-scale details when the simulated velocity field is too stable. Moreover, it can not ensure the momentum conservation since its strength depends on user selectable parameters. Turbulent wind fields are also capable to add vortices to the velocity field. Similarly, their strengths should be small enough to keep the computation convergent and they also cannot ensure the momentum conservation.

The large eddy simulation method developed in this thesis can generate turbulent flow even at the initial phase. It unitizes the spectral method to calculate the pressure and velocity fields. Therefore, its computation process is more stable and faster than those physically based methods. However, because the spectral method damps down the velocity too much on 3D coarse grids, it is more suitable for the flow on 2D layers.

In general, the physically based modeling methods developed for the visualization purpose are simplified methods comparing to the methods applied in engineering applications. They cannot conserve the mass and momentum of the flow exactly, especially when there are non-penetrable objects in the flow. Though their results are more close to exact solutions, they are slower than the heuristic methods and compromised methods and can not ensure convergent simulation under any condition.

Chapter 8 Real-time Visualization of Fire Propagation

Wildland fire occurs over a wide range of spatial and temporal scales in the landscape. An accurate simulation of fire propagation can serve as an educational or training tool for the fireman and the public to find the best way to control the fire propagation and escape from dangerous places. However, without a proper user interface, the simulation output may not be so easily understood by the user as to the researcher who also understands schemes of scientific simulations quite well. Therefore, there is a great need for an easily understandable interface which also allows the user to choose and set different parameters for the scientific simulations in the background as well.

The 4D (spatial and temporal) visualization subsystem in IPODLAS is such a desirable user interface, because it can display the fire spreading in terrain through space and time visually realistically and allow the user to navigate inside the virtual 3D scene and communicate with a fire propagation simulation subsystem, the Geographic Resources Analysis Support System (GRASS) or the fire area simulator, FARSITE[24], at anytime freely. Comparing to the 2D spatial plus temporal visualization, the 4D visualization can also display the flame length and the danger of the smoke to the user though it requires more computational resources and time. Accompanying with the fast development of computer hardware and visualization techniques, we can foresee that 4D visualization, as a complete illustration mode of scientific simulation, is the trend in the near future.

With the current computation resource, it is a big challenge to visualize the fire propagation in terrain in real-time. As discussed in the last chapter, the motion of fire flames and smoke obeys the motional rule of the incompressible flow, which is computationally time-consuming to simulate in physically accurate way. Moreover, the visually realistic rendering of flames and smoke also requires considerable computing time. At the same time, interactions between flames and other objects, such as tree branches, are too complicated to calculate accurately. Furthermore, it is also difficult to measure shapes of smoke and flames in practice at any time interval. Therefore, real-time realistic visualization of fire propagation can hardly be accomplished within a standard computer at present.

This chapter presents a method we applied to the knowledge-based 4D real-time visualization of fire propagation in terrain. Some simplifications are made in order to accelerate the visualization process to achieve real-time effects. The visualization also does not take wind, air pressure, and interactions between flames and objects into consideration. The visualization is part of the integrated system, IPODLAS. Therefore, the user can easily interrupt the visualization and change simulation parameters through the visualization interface to view the new simulated output of fire propagation in terrain. Furthermore, because migratory insects influence vegetation in their resident places, the visualization of fire propagation and the visualization of migratory insects can be integrated to display the property of fire propagating in the place after vegetation has been changed by migratory insects.

8.1 Materials

In this chapter, the 4D visualization of fire propagation in terrain is based on scientific simulation outputs. VTP is used again for its terrain rendering ability and the ability of extending existing visualization capabilities with OpenGL API. Hence, 3D animated flames and smoke can be visualized as an embedded module to the virtual terrain of the study area.

To enhance interactivities, an additional function is developed within VTP to allow the user to select an ignition point in terrain with the mouse. After the selection, the geographical information of the selected ignition point is transferred through sockets to a fire simulation subsystem, for example the

[24] Farsite is a fire behavior and growth simulator. http://www.fire.org/index.php?

GRASS. Then the fire propagation behavior in terrain is simulated and the output is transferred back to VTP for display.

8.1.1 Study Area

All the areas which have digital elevation models and matched uptodate fuel data are suitable to be the study area. The average wind data of the investigated area would assist more realistic simulation. At present, the best data we have are in the Swiss National Park (SNP), Switzerland, where we have a digital terrain model and local fuel data at a ground resolution of 5 meters, and an aerial photograph with a resolution of 2 meters. Based on LIDAR data, we even have the height and crown diameter of every single tree there [MKI*05]. Consequently, we can obtain the 2D fire propagation behavior in terrain and the maximum height of the flame from the simulation run of the FARSITE. Therefore, the fire propagation in SNP can serve as an ideal study case or test bed for the knowledge-based 4D real-time visualization of wildland fire. Fig. 8.1 displays this study area in the SNP.

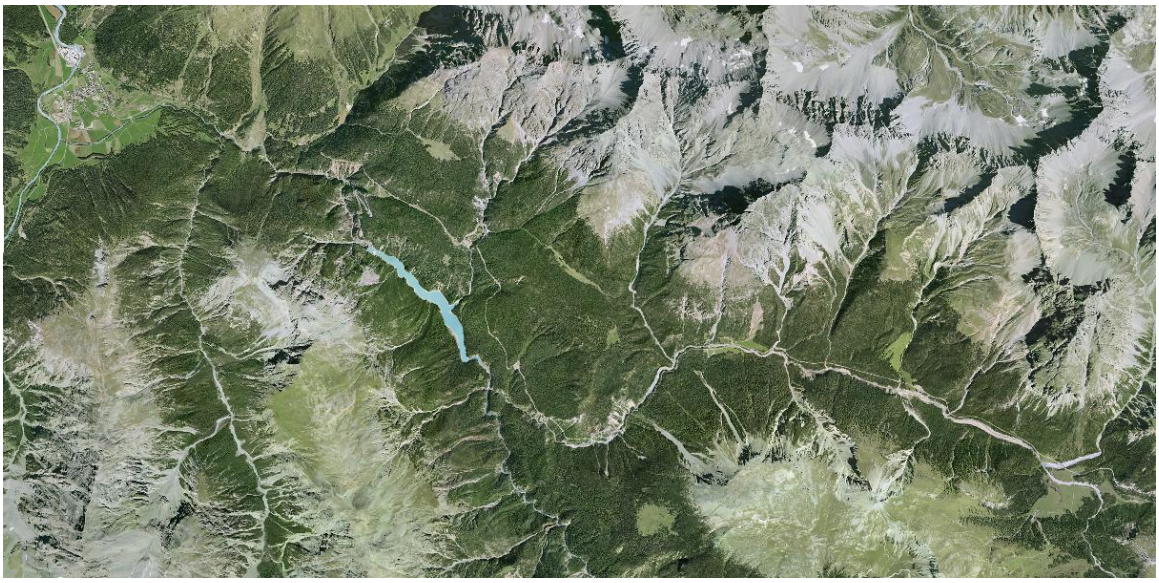


Figure 8.1: An aerial photography of the study area in the Swiss National Park (SNP).

8.1.2 Data Preparation

The following data sets are necessary for the knowledge-based spatiotemporal visualization of wildfire propagation in terrain:

- A Digital Elevation Model (DEM) to describe the geometry of the ground in the study area.
- An image to describe the appearance of terrain in a photo-realistic manner.
- The ignition time of each cell. Based on the local DEM and fuel data, it is simulated by a fire simulation system, such as the GRASS or the FARSITE.
- The maximum flame length in each cell.

However, since we lack the information of fire extinguishing time for each cell, an assumed value is assigned to all the cells. This value depends on the available computation resources. In another word, it should be a proper value to ensure real-time visualization.

More measured or scientifically simulated information, such as the realistic motion of chemical compounds of flames in the air and the temperature distribution in the burning area, could largely

increase the visualization quality. However, limited to the existing knowledge, they are not available at present; but we can expect that their emergence will make the visualization easier and more visually realistic in the future.

8.2 Modeling Method

Modeling the motion of flames and smoke in the air realistically is extremely time-consuming especially when there are many interactions between them and other objects, such as vegetations, in the combustion area. Therefore, to reduce the computing time for the purpose of real-time visualization, only flames above the ground are visualized in the scene. Namely, the interactions between flames and vegetations or other objects are ignored. Then, to indicate the resulting vegetation change after combustion, the color of the burned area is changed to black right after the extinguishment of fire.

In order to further reduce the modeling time, the motion of flames and smoke should also be simplified. Considering the methods discussed in the last chapter, we can find that only heuristic methods are able to visualize large-scale gaseous phenomena in real-time at present. All the other methods need a large number of particles to calculate their motion properties. Taken the rendering process into consideration, the heuristic modeling method of moving small textures inside the volumes of smoke and flames is applied to the real-time visualization of fire propagation in this thesis.

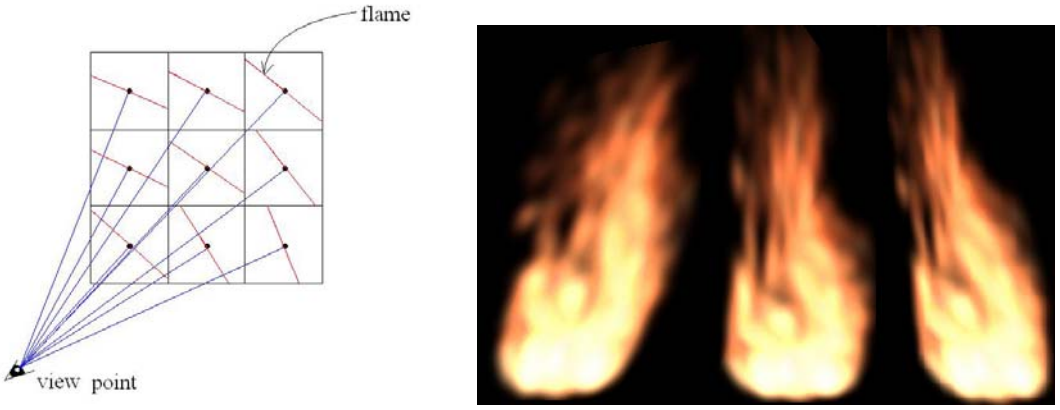
This method is introduced in section 7.1.1. The primary advantages of this method are that the turbulent properties of flames and smoke are directly created by the changes of small textures, and only a small amount of particles are needed in the later rendering process. Because the ground is static to flames and smoke and we ignore the interactions among flames, smoke, and other objects, the object dynamics in this method need not to be taken into consideration. The global dynamics can be accomplished by propagating small textures upwards to mimic the rising behavior of flames and smoke. The local dynamics, swirling and flickering movements, then can be realized by the random replacement of some textures used in the previous frame and texture animation, namely continuously changing texture coordinates.

8.3 Rendering Method

Since the above modeling method requires small textures to represent groups of particles, proper images should be used in order to avoid artifacts in the final rendering result. For example, if small textures have regular geometries and obvious edges, the final image will look more solid than fuzzy. Therefore, the selection of those textures needs special attentions. Generally speaking, the selection should follow two rules: first, an image selected to be a small texture should contain turbulent details; second, the edge of the image should look as fuzzy as amorphous phenomena.

For flames, which do not rise as high above the ground as smoke does, a single texture is used in this thesis for a burning cell in terrain to accelerate the rendering speed. Flame images in Fig. 7.29 fit the above selection requirements and hence can be used as textures. From the time of a cell being ignited, a random selected flame texture is used and blended with background images to represent the combustion of the cell. The height of the texture starts from zero and grows up gradually to the flame length of the cell; and then it gradually reduces to zero again till it distinguishes. Since textures are 2D images, the dynamic impostor technique is used to rotate flame images to always face the viewer. However, it makes no sense to rotate the bottom of flames to the air. Therefore, the bottoms of the flame's textures are always located on the surface of the ground and rotate around cells' centers as shown in Fig. 8.2 (a). To introduce more turbulence, we can continuously change the positions of a texture's two top vertices randomly to create animation effects. As an example, Fig. 8.2(b) shows three flames generated by the texture animation method.

Smoke can rise to higher places in the air than flames. Therefore, more small textures are needed for the rendering of smoke rising from every burning cell. These small textures can be parts of smoke images or images already rendered by many metaballs as shown in Fig. 8.3. To ensure fuzzy edges, a Gaussian filter can be applied on images with distinguished solid edges. Moreover, the dynamic impostor technique is also applied to the smoke texture to create 3D visual effects. Unlike flame textures whose bottoms must stay on the ground, a texture of smoke can be rotated to face any directions around its center.



(a) (b)

Figure 8.2: (a) A 2D projection of flame textures in cells; (b) flame images generated by texture animation.

Figure 8.3: Small textures for smoke rendering.

8.4 Cross-scale Modeling

The final goal of the real-time visualization of fire propagation is to visualize wildfire in a visually realistic way no matter where the viewer stands. The physically based methods or compromised methods discussed in chapter 7 described how to model fire flames and smoke in various visually convincing ways; and the photorealistic rendering methods discussed in chapter 5 can render flames and smoke lively. However, using these methods can not generate a real-time visualization in such a large scale as the one which is achieved with the methods presented in section 8.2 and 8.3. In other words, because of the limited computation resources of common personal computers, the real-time visualization of wildfire propagation is accomplished by compromising visual quality.

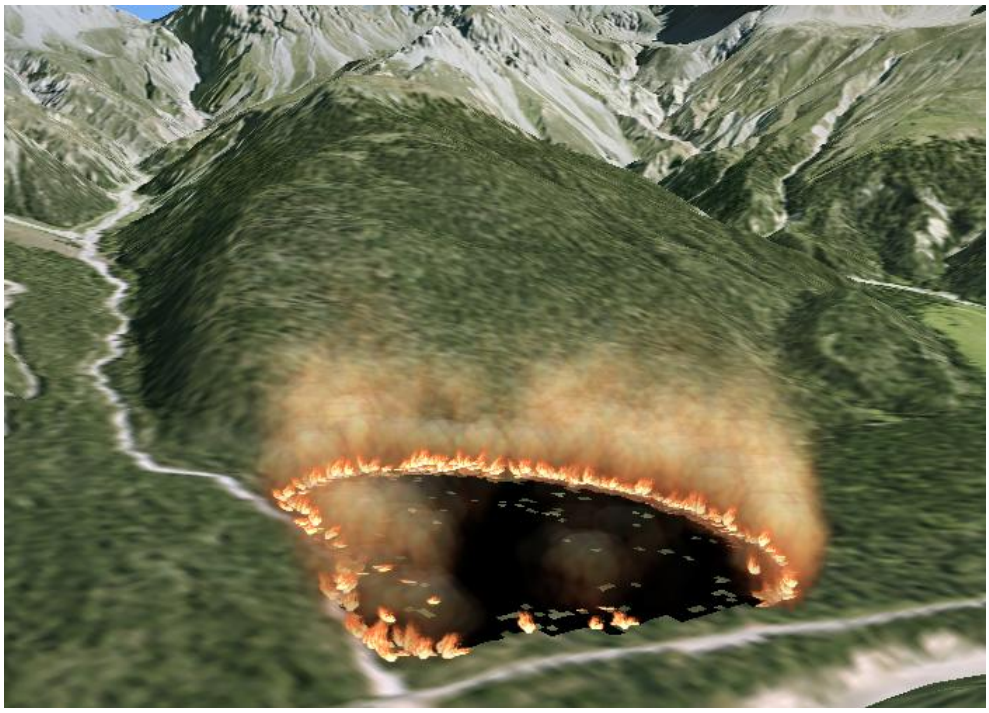
Since flames and smoke can be visualized in real-time in a very small scale as discussed in chapter 7, an efficient cross-scale modeling method can enable the user to continuously observe flames and smoke from different distances. In other words, when his view point is close enough to a small flame he could observe the flame visually realistically; and when his view point gradually moves away from the wildfire he could observe the overall wildfire propagation behavior.

Accordingly, depending on the position of the viewer, an efficient Level of Detail (LOD) method is required to change the position and the number of internal particles which display the motion of flames and smoke, and furthermore change textures for these particles to make the appearance of flames and smoke visually continuously while the user navigates in the virtual landscape. This method should be based on the visualization methods presented in chapter 7 and this chapter. It should also ensure the real-time performance. Because of the time limitation, the research of cross-scale modeling could be a challenge of the future work.

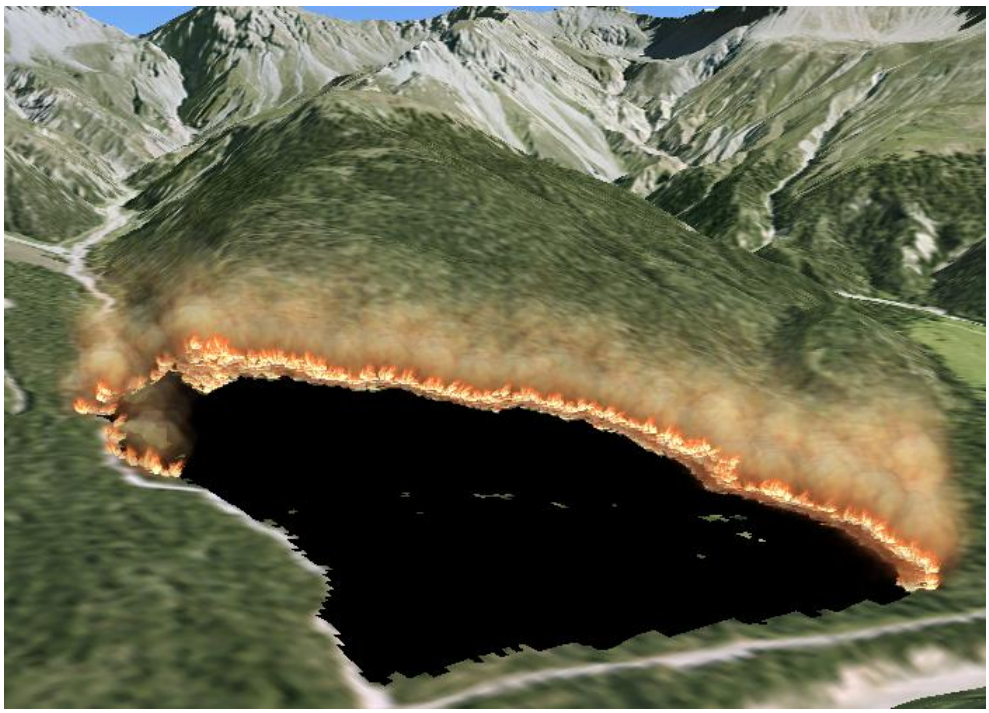
8.5 Results

This chapter presents a simple real-time visualization method to display wildfire propagation in 4D. The fire propagation information is obtained from a scientific simulation run in the FARSITE. The flame in each cell is represented by a random flame texture, whose bottom is always located on the ground and the two vertices on the top of the texture randomly change positions to generate the effect of flame animation. The flame texture is rotated around the center of its cell while the user changes his view point. The smoke of each burning cell is visualized by blending small composite smoke textures which always randomly move upwards. Burned areas are changed to black to simulate vegetation residues. As an example, Fig. 8.4 displays five images in time sequence obtained from a real-time visualization of wildland fire propagation in SNP.

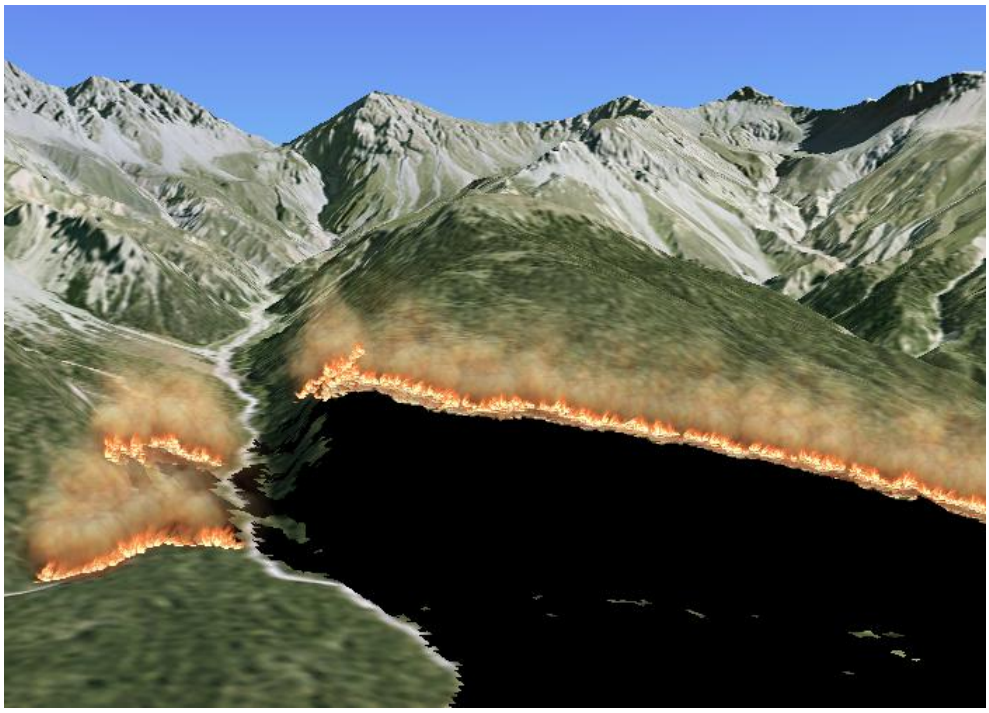
(a)



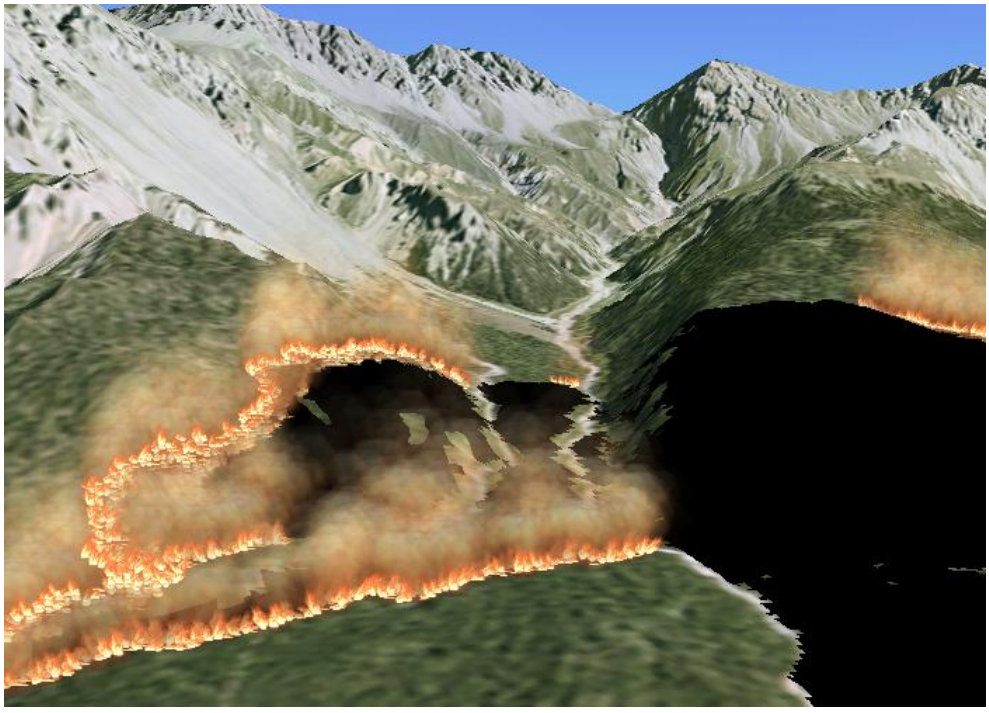
(b)



(c)



(d)



(e)

Figure 8.4: Real-time visualization of wildfire propagation from an ignition point in SNP.

Chapter 9 Results

This thesis developed efficient and fast modeling and rendering methods for the visualization of natural amorphous phenomena such as clouds, smoke, and flames in small-scale. In order to visualize these phenomena in large scale in real-time based on scientific simulation data, corresponding simplified modeling and rendering methods have also been developed. Therefore, the contributions of this thesis can be divided into two categories: contributions to the integrated spatiotemporal system, IPODLAS, and contributions to the photorealistic visualization of natural amorphous phenomena. Though the methods developed for the photorealistic visualization are not implemented in terrain visualization yet because of their non real-time property, we can believe that they are soon going to become real-time and suitable for the terrain visualization with the fast development of the computation hardware and new terrain visualization methods. The result of each method has been presented with screenshots in its corresponding section. This chapter is to give the reader a general overview.

9.1 Contributions to The IPODLAS

This thesis developed visualization tools for the knowledge-based real-time 4D visualization of migratory insects and wildland fire propagation in terrain. These visualization tools are developed as embedded modules of VTP, which is the 4D interface of IPODLAS. The aim of these visualization modules is to provide the user, especially the layman, an easily understandable interface other than data tables which are detached from spatial locations. Therefore, simplified modeling and rendering methods are developed to generate visualizations in real-time.

9.1.1 Real-time 4D Visualization of Migratory Insects

This thesis developed a real-time 4D visualization approach for the migratory insect dynamics and resultant vegetation changes in terrain. In order to present scientific spatiotemporal simulation results in a visually intuitive manner, insects in groups are visualized as animated clouds and migrate to their destinations in the virtual landscape. Simplified methods are used to accelerate the visualization of dynamic clouds with the compromise of visual quality. Due to the redistribution of migratory insects after their migration in each year, the influenced vegetation has different extents of defoliation. Hence, the appearance of the landscape is also changed in the visualization, depending on new local defoliation extents.

Though this thesis takes LBM dynamics in the Upper Engadine valley as a study case, other migratory insect dynamics in other places can be visualized with proper input data as well. The user can observe his interested migratory species in the selected time interval and location through the visualization. Commands and data are transferred between the visualization subsystem and other subsystems in IPODLAS. After a new scientific simulation result is obtained and transferred to the visualization subsystem, it can display the result in 4D in virtual terrain to the user.

9.1.2 Real-time Visualization of Wildland Fire Propagation

This thesis developed a 4D real-time visualization approach for the fire propagation in the landscape. In order to display large-scale fire flames and smoke in real-time, simplifications are made with the compromise of visual quality. Dynamic flames are visualized by applying texture animation techniques on photorealistic flame images, while dynamic turbulent smoke is visualized with small smoke images which move upwards from fire flames. The appearance of smoke adds reality to the virtual scene. To indicate places after combustion, the ground changes colors to black to mimic residues after combustion.

The visualization of wildland fire propagation is based on scientific simulation results. The user can freely select an ignition point on the ground in the visualization subsystem. The geographical information of the selected point is then transferred through a socket to the IPODLAS kernel and then to a fire simulation subsystem. After this subsystem provides the simulated fire propagation data, the visualization subsystem starts to display fire propagation in the virtual landscape from the ignition point.

The real-time wildfire visualization displays fire propagation behaviors to the user in an easily understandable way. With the 4D visualization subsystem, simulation results are displayed in 3D terrain. Thus, even the layman can observe fire propagation properties related to the change of the landscape. In general, this work provides a 4D interface to the scientific simulation of wildland fire propagation.

9.2 Contributions to The Visually Realistic Visualization of Natural Amorphous Phenomena

In addition to simplified methods for real-time visualization in IPODLAS, this thesis also developed sophisticated methods for visually realistic visualization of natural amorphous phenomena in small-scale. Though these methods take more computing time, resultant natural amorphous phenomena have much better visual effects. They can be applied to cross-scale visualization in IPODLAS when the common computer hardware is fast enough in the future.

9.2.1 Photorealistic Cloud Visualization

The two-level marching cube method was developed in this thesis to easily generate visually realistic cumulus cloud models with small particles. This new modeling method can generate various models with several primitive geometries such as spheres and ellipsoids. The outer surface of applied primitive geometries defines the macrostructure for cloud models. With the change of cube sizes at the two different levels in the modeling process, various models can be created automatically.

The recording matrix method is developed in this thesis to efficiently render amorphous phenomena, such as clouds, smoke, and flames. Taking multiple light sources and light multiple scattering among internal particles of amorphous phenomena into account, this method can display visually convincing gaseous phenomena with self-shading properties in various colors no matter whether they are illuminated by spot or direction light sources. In addition, this method only requires a small temporary saving space and can accomplish rendering processes faster than the hardware acceleration method without losing visual quality. Moreover, with an adjustable scaling factor, it can further accelerate rendering processes efficiently to various extents or improve illumination effects according to the user's requirements. With this rendering method, amorphous phenomena can be displayed in a fast and photorealistic way.

9.2.2 Physically Based Smoke and Flame Visualization

This thesis developed two physically based modeling methods to simulate the motion of smoke and flames as dynamic incompressible flows. The LES method can be used to simulate turbulent flows in a 2D laminar layer with open boundary conditions efficiently. This method conserves the small scale rolling characteristics of the flow, which are omitted in the computation on coarse grids. Doubling the computation domain in the non-repetitive direction allows the use of spectral methods to compute flows with non-repetitive boundary conditions. With the fast Fourier transform, the numerical solver for the incompressible Navier-Stokes equations is largely accelerated and the convergent scope of

smoke's velocity fields is enlarged. This method also allows the user to interactively add other influential factors, such as a turbulent wind field to the modeling flow in process.

The particle-based modeling method developed in this thesis is a compromised modeling method, which takes both physical accuracy and computing time cost into consideration. This method keeps the motion and physical properties of individual particle and calculates some average properties on grids to accelerate the modeling process. The turbulent motion of the modeling flow is generated by vortices which are attached on particles and move together with particles in the velocity field. This method can model 3D turbulent flows, including smoke and flames, in real-time.

Smoke and flames can be rendered by the recording matrix method with slight changes according to individual conditions. Consequently, visually realistic 3D turbulent smoke and flames in small scale can be visualized in real-time.

Chapter 10 Discussions and Future Work

This chapter discusses the particular simplifications made in this thesis to obtain a fast visualization speed without losing too much visual reality. Based on the developed methods, some further applications and related future research directions are presented. These applications and research directions can help the interested reader to better utilize or/and refine this PhD work.

10.1 Simplifications for The Purpose of Fast Visualization

This thesis adopted various simplified methods to accelerate the visualization speed. In the visualization of atmospheric clouds, the multiple scattering property of light is simplified as multiple forward scattering to reduce the computation complexity. This simplification avoids the tangling computation of light multiple scattering among a large number of internal particles. Consequently, the illumination result of incident light on all the internal particles of a cloud can be calculated orderly in one process. The rendered results shown in the chapter 5 and 7 prove that this simplification does not really reduce visual quality. In this regard, clouds, smoke, and flames still can be rendered in a visually realistic way.

To visualize the dynamics of migratory insects in real-time, groups of real creatures are represented by clouds in an abstract way and those clouds are dynamically constructed by randomly distributed particles and rendered with a simplified method to accelerate the modeling process. As we can observe from the images of visually realistic atmospheric clouds in chapter 5, the simplified modeling method allows a real-time visualization with a compromise of visual reality. However, visualizing insect clouds as photorealistic atmospheric clouds could not enhance the user's understanding to the dynamics of migratory insects. Therefore, the simplified visualization method does not reduce the potential of this visualization subsystem to display temporal simulation results.

In the visualization of fire flames and smoke in small-scale, physically based modeling process is an extremely time-consuming process. To save computing time, coarse equidistant orthogonal grids are normally used for their simple structures and the computing time which can be saved without the use of finer grids. This simplification results in too stable flows since small-scale turbulences are damped down by the computation on coarse grids. To remedy this effect, turbulences are introduced in the velocity field by external forces. For example, the vorticity confinement method adds external forces to enhance the existing vortices, and in the particle-based method vortices are directly introduced and move with the other internal particles of the flow in its velocity field. Though these remedy methods do not keep the physical accuracy of the simulation, they result in swirling and rolling movements and hence make the visualization of flames and smoke visually more realistic.

One difficulty in flames and smoke modeling is to simulate interactions between the flow and non-penetrable objects. If we directly set normal velocities at boundary surfaces to zero, the simulated velocity field will have artificially slow velocities in the area near the boundary. Therefore, different methods, such as the ghost value method and the method of setting the normal pressure gradient to zero, are used to solve this boundary problem. However, an unavoidable step for those methods is to bounce the particle back from the boundaries, and those methods also can not keep the physical accuracy of the simulation. This thesis does not develop any new efficient method to deal with this problem. The direct bouncing back method is applied for its simplicity. A further idea about the boundary problem will be discussed in the section of future work.

The particle-based method developed in this thesis is an approximate physically based method. The simulated velocity field does not conserve the mass and momentum. In fact, some former modeling methods [FM97, FSJ01, and NFJ02] which concentrate on the physical accuracy prove that physically based modeling is time-consuming for the visualization purpose and its results are not visually

attractive as they are expected unless turbulent external forces are introduced in on purpose. However, with artificial external forces, the mass and momentum can not be conserved anymore. Moreover, the calculation process of these methods is not always sound because of the easily divergent property in the calculation of incompressible Navier-Stokes equations. Therefore, some approximate methods [LF02, SRF05, and TLP06] are developed recently to pursue both visual quality and fast modeling speed instead of physical accuracy. Different from other approximate methods, the particle-based method first takes both individual and average properties of the flow into consideration. Moreover, it achieves real-time simulation of the flow.

In fact, it is difficult to simulate turbulent flows with limited information about their surrounding air flows. Internal particles of smoke and flames are so sensitive that their movements can be influenced by tiny disturbances. Without external disturbances, the flow simulated on a coarse grid will be quite stable. Consequently, simulated flows do not have attractively swirling and rolling properties without the consideration of their interactions with the environmental air flow. Therefore, when current modeling methods are used to simulate the motion of flows, external forces are introduced in the modeling field deliberately to generate more vortices and hence make the flow appear more turbulent.

In the wildfire visualization, fire propagation properties are modeled by a scientific fire simulation system. Therefore, the accuracy of fire propagations in terrain depends on the simulation system. The visual reality of the visualization depends on the computation resource and the available environmental information. For instance, with current standard computation resource, we can not visualize large-scale fire flames and smoke visually realistically in real-time with the methods discussed in chapter 7. Moreover, the environmental information, such as wind and air pressure, can influence the appearance of flames and smoke so much that the visual reality can be largely enhanced by taking them into account. Therefore, as a 4D interface of scientific simulation system, the visual effect of the visualization subsystem is limited to available knowledge and techniques. However, the visualization for the purpose of entertainment, which only takes the visual reality as the primary concern, has more freedom to generate visually realistic results with imagination.

10.2 Further Applications

This thesis developed a visualization module to display the dynamics of migratory insects and resultant vegetation changes, and another module to display the wildfire propagation in terrain. These two modules have no limitations on simulation systems and study areas. Therefore, the visualization system can display various migratory insect dynamics and wildfire propagations at various places for different simulation systems.

The atmospheric cloud visualization can be applied to flight simulation, games, and weather forecast, where the appearance of photorealistic clouds in the air can largely enhance visual quality. Though the overall atmospheric cloud visualization needs a long computing time, rendering clouds again according to the movement of the view point does not take a long time with current displaying resources as long as clouds and their light sources are static, because in this case intensities on internal particles of clouds will not change and clouds can be rendered by only blending all the internal particles in distance descending order to the view point. The dynamic impostor technique is used to accelerate the visualization of atmospheric clouds in terrain. In addition, when the change of the view point is very small and the viewer cannot distinguish the difference of a cloud's appearances at two positions, the cloud need not to be rendered again.

Driven by the air flow, atmospheric clouds change shapes irregularly and move together with the flow. This phenomenon can be visualized by combining the method of cloud visualization with the method of flow modeling. Nowadays, satellite images can display the motion of clouds in 2D. Applying these

images to confine the motion of clouds, we can reconstruct 3D clouds from 2D images and the resultant visualization will display the motion of clouds in a realistic and visually impressive way.

The modeling methods for flames and smoke can be applied to the simulation of other turbulent flows in movies and games. For example, when a car passes by a place with a high speed, the nearby objects, such as leaves on the street, feathers, clothes, and hairs, are affected by a sudden change of the air flow. Ignoring the interaction between these objects and the air flow, we can visualize their dynamics driven by the turbulent air and their own constraints, for example their gravities. Another example of a flow may be the lava flood from a volcano. Based on scientific simulations, their visualizations can be educational tools or 4D interfaces of simulation systems as well.

10.3 Future Work

The simplification of light multiple forward scattering is based on scientific measurements on non-icy particles [HWL*00]. However, soot in flames is different from those particles and it emits light equally to different directions. This simplification ignores the illumination effect of soot on some particles which are in front of it in the computing queue. Therefore, the visual effect of flame visualization can be enhanced by taking the backward illumination effect of soot into consideration.

The terrain visualization algorithm implemented inside VTP needs to be updated in the future work. The algorithm does not take the potential of present GPU. Though it is a real-time dynamic LOD algorithm, it still can be improved and have much faster rendering speed. Since we need to visualize other complicated phenomena in terrain, it is important to have a faster terrain rendering algorithm.

In the visualization of migratory insect dynamics, the resultant vegetation change is represented by changing the color of corresponding places in terrain. With local vegetation information in the future, we can visualize the vegetation in 3D as well, and directly change their colors to enhance the visual reality.

Dealing with the boundary condition efficiently and accurately is a big challenge in the flow visualization. Current modeling methods calculate the velocity field of the flow with the finite differential method. Because boundary conditions are hardly to be defined in a proper way, the velocity field cannot ensure the mass and momentum conservation of the flow. Finite volume method may solve this problem easier than the finite differential method. As hinted by its name, finite volume method can ensure that the flux flows out of a finite volume equals to the flux flows into the volume. Therefore, applying the finite volume methods for cells near the boundary may enhance the accuracy.

The wildfire propagation is visualized in terrain without considering interactions between flames and local vegetation. It would be more visually realistic to visualize the combustion process of every individual tree in the forest. Then, complicated computation is required to calculate the interaction between flames and trees with complicated geometries. Moreover, we also need to concern the process of objects turning into combustible fuels. Therefore, the visually realistic visualization of the combustion process on individual trees could be a future research direction.

Based on the visualization of wildfire propagation in large-scale and the visualization of the combustion process on individual trees in small-scale, cross-scale modeling could be a challenge for the future work. The cross-scale modeling should allow the user to observe both combustion processes on individual trees and fire propagation properties in terrain. When the user changes his view point, he should not observe gaps in the visualization. Namely, accompanying with the user's navigation in the virtual landscape, the visualization should be still in real-time and display the user continuously visually realistic images.

Chapter 11 Conclusions

This thesis developed new methods for the visually realistic visualization of natural amorphous phenomena, as well as the real-time visualization of those phenomena in terrain based on scientific simulation results. With the work of this thesis, clouds, smoke, and fire flames can be visualized in a fast and visually realistic way. Moreover, the dynamics of migratory insects and wildfire propagation can be displayed in real-time in the virtual landscape.

A new rendering method for amorphous natural phenomena, called recording matrix, is developed in this paper. With this method, the illumination effects of incident light transmitting through gaseous phenomena can be calculated fast and efficiently. An efficient transformation changes models illuminated by the spot light source to models illuminated by the direct light source. Hence, illumination effects of the spot light can be calculated by the recording matrix method as well. An extra adjustable scaling factor is used in this method to change the size of model. Our experiences prove that shrinking or enlarging models by this factor can further accelerate the calculation of the illumination phase or improve its computational precision under different conditions. Results produced by the recording matrix method are analyzed and compared with the hardware acceleration method. The comparison proves that visually convincing gaseous phenomena can be rendered with this method in a much faster way.

A real-time 4D spatiotemporal visualization is developed to display scientific simulation results of migratory insect dynamics and resultant vegetation changes in terrain. The visualization displays simulated ecological phenomena in an intuitive way, which allows research results to be easily understood by a wide range of users. In this visualization, dynamic insect clouds are used to represent migrating insect groups in real-time. The resultant vegetation change is represented by changing the color of the corresponding place in terrain according to the simulated defoliation percentage.

A new large eddy simulation (LES) method is developed to model 2D turbulent flows in a physically accurate way. In this model, a velocity field is decomposed into two parts, a mean velocity field and a fluctuation. The mean velocity field, which is a more accurate numerical solution according to the Navier-Stokes equations, dominates the large scale motion of smoke; while the fluctuation based on the mean velocity field reflects the small scale details. The main advantage of this method is to simulate turbulent smoke efficiently in a coarse grid and the turbulence can be easily controlled by adjusting either external forces or the fluctuation factor. With a revised spectral method, efficient and stable simulations can be achieved by LES. The revised spectral method can work on non-periodic boundary conditions and compute the differential equations accurately. Consequently, the Navier-Stokes equations and the derived Poisson equation for the calculation of air pressure can be solved directly and simply in the Fourier domain. Moreover, the spectral method avoids the instability and risks of divergent computation in other iterative methods.

A particle-based modeling method is developed to model 3D turbulent flows in real-time. This method takes individual movements of particles into concern as well as their average properties on grids. A lifetime is given to every particle to control its size and temperature. Based on their lifetimes, particles can change from fuel to soot in the flame visualization. With vortex particles which move together with other particles in the simulated domain, swirling and rolling properties are introduced in the velocity field of the flow. In general, the motion of flames and smoke can be modeled in a visually realistic way with the particle-based method.

A real-time 4D spatiotemporal visualization is developed to display scientific simulation results of wildfire propagation in terrain. Each fire flame is represented by a flame image with the dynamic impostor and texture animation techniques to generate visually dynamic effects. Smoke is visualized by a few small smoke images with simple movements to enhance the visual reality. The place after

combustion is indicated by changing it to black. The visualization displays the user the wildfire propagation properties in a visually perceivable way.

In general, this thesis contributes to the visually realistic visualization of natural amorphous phenomena in both small-scale and large-scale. The 4D visualization of these phenomena can be applied to display scientific simulation outputs in terrain in an intuitive way.

Appendix

A Pseudo Code

In order to make this PhD work further understandable and easily reproducible, this section presents the pseudo code of methods developed in this thesis. The real code is in C++ with OpenGL API. The pseudo code described below is to give the reader a more clear view of the methods developed in this thesis. Accordingly, the reader can reproduce them in other programming languages and graphics APIs. Moreover, it is also convenient for the reader who wants to extend these methods to have an easier start.

A.1 Insect Cloud Visualization

Insect cloud visualization is a real-time visualization implemented as a module of VTP. The processes of loading a DEM as geometry and a satellite image as texture are accomplished by existing VTP functions. Then dynamic terrain is visualized automatically. The new module is implemented in an inherited dynamic geometry class, which has a callback function to update virtual scenes. This function is called 25 times per second (if the computation and rendering processes can be finished in time) to ensure the real-time visualization. The left work is done according to the following pseudo code:

1. Read insect migration data from socket or a file.
2. Read vegetation change data from socket or a file.
3. Read geographic data to locate the insect dynamics in terrain from a file.
4. Calculate the migration road between two ‘sites’.
5. Calculate the particle number to represent the number of migrating insects.
6. Assign a color to each insect cloud according to the user’s choice or randomly.
7. Visualize one insect cloud rising from one site at the beginning of each migration. Each insect cloud has the similar shape as its original site.
8. Separate each insect cloud into smaller clouds, when it rises to a certain height above the ground. These small clouds, originally from one site, have different migration targets.
    a) The macrostructure of insect cloud is determined by several ellipsoids, whose sizes depend on the number of individual insect cloud.
    b) The microstructure of insect cloud is shaped by randomly distributed small particles, which are textured with a Gaussian function to ensure ‘fuzzy’ edges.
    c) Particles are blended on a dynamic impostor.
9. Move insect clouds according to the migration roads and the shape of the ground.
    a) Insect clouds are always above the ground.
    b) Particles are redistributed for each cloud.
10. Disperse particles of an insect cloud when any part of the insect cloud is above the center of its target site.
    a) The landing path of a particle is determined by its current position and a random position on its target site.
    b) Particle disappears from the view when it is below the surface of the ground.
11. Change the color of sites according to the corresponding vegetation data after all the insect clouds land in their target sites.
12. Start to visualization the migration in the next year. Namely, start again from step 4.

A.2 Atmospheric Cloud Visualization

Because of the considerable visualization time which is needed to visualize atmospheric clouds with current computation resources, the atmospheric cloud visualization is not implemented inside VTP, but in an independent application instead. Models of atmospheric clouds are also constructed by particles. This section describes the static atmospheric cloud visualization. Namely, the positions of particles do not change according to the time. The atmospheric cloud visualization can be divided into two parts: modeling and rendering. Both pseudo codes are given below.

Atmospheric cloud modeling

1. Generate several big ellipsoids for the macrostructure of a cloud.
2. Utilize the marching cube algorithm to locate particles (small spheres) on the outer surface of the ellipsoids with a proper cube size.
   a) Generate a grid with the cube size.
   b) Mark grid nodes which are inside the ellipsoids.
   c) Randomly locate points on the red triangles in Fig. 5.2.
   d) Initialize every particle's radius with a proper random number.
3. Utilize the marching cube algorithm again to locate particles according to the former located particles with a much smaller cube size.
4. Repeat step 3 until all the particles are located. The particle number is pre-selected according to the required size of clouds.
5. More complicated cloud models can be generated by the combination of several simple models.

Atmospheric cloud rendering

1. Sort particles according to their distances to the light source.
2. Build a recording matrix according to the physical size of clouds on the screen. This can be done by only considering the size of big ellipsoids which construct the macrostructure of clouds.
3. Initialize all the elements in the recording matrix to be the intensity of the incident light.
4. For each particle, calculate its projective area in the recording matrix from near to far against the light source.
5. For each particle, change its corresponding element values in the recording matrix according to the extinction parameter and albedo.
6. For each particle, average its corresponding element values in the recording matrix and save the average as its intensity.
7. After all the particles obtain their intensities, sort them again according to their distances to the view point.
8. Render each particle from far to near against the view point.
   a) Multiple the intensity of every particle with albedo and set the result as the red, green, and blue channels of the particle's color.
   b) The alpha channel of the particle's color is set to be the extinction parameter.
   c) All the particles are textured with an image calculated by a Gaussian function with "GL_MODULATE" mode.
   d) Set the OpenGL blending function as: glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA).
   e) Blend every particle on an impostor.
9. Rotate the impostor to face the view point.

Clouds need not to be rendered again if positions of the light source and the view point change to a negligible extent, within which the user cannot distinguish the difference between a cloud's appearances. However, if the position of the light source changes to a non-negligible extent, no matter

whether the position of the view point changes, rendering process needs to be started again from step 1. Otherwise, clouds have incorrectly self-shading properties. But if only the position of the view point changes to a non-negligible extent, the rendering step can start from step 7.

A.3 Smoke Visualization

The smoke visualization also includes two parts: smoke modeling and smoke rendering. As discussed in chapter 7, this thesis develops two smoke modeling methods. The pseudo code of LES method, which is suitable for the simulation of flows in 2D laminar layer, is presented first and followed by the pseudo code of particle-based modeling method.

Smoke modeling

I LES

1. Build a grid and initialize velocity, temperature, and density for each cell. These properties are all located in the center of each cell.
2. Initialize the position, size, and temperature of internal particles.
3. Find a proper time interval, $\triangle t$, to update the velocity field in the grid. A too large $\triangle t$ will lead to a divergent and artificial simulation. Generally speaking, the result of $\triangle t$ multiplying the velocity of a cell should be less than the length of the cell edge.
4. Add external forces to a new empty flux field.
5. Utilize the library of fast fourier transform to transform a temporary composition field according to the advection item.
6. Calculate the advection item in the Fourier domain.
7. Transform the temporary field back and scale it according to the size of the grid. The fast Fourier transform makes the length of every vector in the field be multiplied by the size of the grid.
8. Add the temporary field to the flux field.
9. Calculate the diffusion item with CDS.
10. Transform both the flux field and a copy of the velocity field to the Fourier domain.
11. Calculate the pressure item.
12. Transform the flux field back and scale it.
13. Update the velocity field according to the flux field. Herein, the flux field multiplied by $\triangle t$ is the increment of the velocity field.
14. Calculate the fluctuation velocity field based on the new velocity field.
15. Update the position of particles according to both the fluctuation velocity field and the new velocity field. Linear interpolation is performed to calculate the velocity for every particle in the grid.
16. Update the temperature and density for the grid with the Semi-Lagrangian Scheme.
17. Start from step 3 for the next frame.

II Particle-based modeling

1. Build a grid and initialize velocity, temperature, and density for each cell.
2. Initialize the position, size, lifetime, and velocity of internal particles.
3. Initialize the vortices strength and select random particles as their carriers.
4. Update the density and temperature fields in the grid according to the lifetime and position of the particles.
5. Update the velocity field in the grid according to the differences of temperatures and densities in different cells.
6. Calculate the influence of vortices on the velocity field.
7. Update the velocity of particles according to their lifetimes and viscosity of the air or other intermediate.

8. Update the position of particles according both the velocity field and their own velocity. This is like a relative motion.
9. Update the density of cells according to the new position of particles.
10. Update the lifetime of particles according to the density of their current cells and the difference of temperatures between their old cells and current cells.
11. Update the size of particles according to their new lifetimes.
12. Update the temperature of cells according to the new lifetime of particles and the density of cells.
13. Start from the step 5 for the next frame.

Smoke rendering
Smoke illuminated by the direct light source can be rendered in the same way as the atmospheric cloud rendering pseudo code. If smoke is illuminated by the spot light source, the step 2 and 3 of the pseudo code should be replaced by the following:

3. Rearrange all the particles.
   a) Calculate the general size of particles on a projective surface according to the position of the light source.
   b) Adjust this projective surface to make the projection of particles have a proper size. Too large projection will increase the computation cost and too small projection will make the illumination result artificial.
   c) Calculate the new position of particles according to the position and illumination angle of the spot light source. If some particles are located out of the illumination angle, their intensities are directly set to 0.
4. Initialize all the elements in the recording matrix according to a Gaussian function, which mimics the effect of the gradually decreasing intensity of the spot light source.

A.4 Flame Visualization

The flame visualization also contains two parts: flame modeling and flame rendering. The flame modeling is similar as the smoke modeling. For 3D real-time flame visualization, the particle-based modeling method is used. However, because of the special property of soot in flames, particles of flames are divided into two categories: soot and non-soot. The initial statuses of particles are set to gaseous fuel with different lifetimes. Accompanying with their movements, some particles are ignited, turn into soot, and release heat. When the lifetime (temperature) of a soot particle decreases to a value below a threshold, it does not emit light anymore. Instead, it allows some light to go through it and also scatters some light to other directions with different extents.

Accordingly, the flame modeling only needs to replace the step 10 in the pseudo code of the particle-based modeling with:

10. Update the lifetime of particles according to the density of their current cells, the difference of temperatures between their old cells and current cells, and the status of particles. If the lifetime of a particle is long enough and the particle is non-soot, it turns into soot with an increase to its lifetime.

The flame rendering is similar to the atmospheric cloud rendering and the smoke rendering. Depending on the type of light source, it has a different step 2 and step 3 as shown in the pseudo code of the smoke rendering. Moreover, because soot absorbs all the light and emits light according to its temperature, step 5 and 6 in the pseudo code of the atmospheric cloud modeling need to be changed for soot particles as shown below:

5. For each soot particle, change its corresponding element values in the recording matrix according to the intensity of its emitting light, which depends on its temperature.
6. For each soot particle, save the intensity of its emitting light.

References:

[AFF01] Allgöwer, B., Fischlin, A. and Frei, U.: Knowledge based dynamic landscape analysis and simulation for alpine environments. Full Proposal for SNSF Project Nr.4048-064432, 2001, Zürich.

[And92] Anderson, C. R.: An Implementation of The Fast Multipole Method without Multipoles. *SIAM J. Stat. Compu*tation, 1992, 13(4), pp.923-947.

[BF88] Baltensweiler, W., Fischlin A.: The Larch Bud Moth in the Alps. *In: Berryman, A.A.(ed.), Dynamics of forest insect populations: patterns, causes, implications,* Plenum Publishing Corporation, New York a.o., 1988, pp. 331-351.

[BH86] Barner, J., Hut, P.: A Hierarchical O(N log N) Force-calculation algorithm. *Nature*, 1986, 324(4), pp. 446-449.

[Bie04] Biegger, S.: A Visual System for The Interactive Study and Experimental Simulation of Climate-induced 3D Mountain Glacier Fluctuations. *Remote Sensing Series*, 2004, Vol. 43, University of Zurich. 150 pages.

[Boh87] Bohren, C.F.: Multiple Scattering of Light and Some of its Observable consequences. *Am. J. Phys.* 1987, 55(6), pp.524-533.

[BR99] Baltensweiler, W., Rubli, D.: Dispersal: An important driving force of the cyclic population dynamics of the larch bud moth. *Zeiraphera diniana Gn. For. Snow Landscape Res.*. 1999, 74(1): 3-153.

[CCF94] Cabral, B., Cam, N., Foran, J.: Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware. *Symposium on volume Visualization 1994*, pp. 91-98.

[CH06] Clasen, M., Hege, H.C.: Terrain Rendering using Spherical Clipmaps. *Eurographics /IEEE VGTC Symposium on Visualization 2006,* pp. 91-98.

[CGR88] Carrier, J., Greengard, L., Rokhlin, V.: A Fast Adaptive Multipole Algorithm for Particle Simulation. *SIAM J. Sci. Computation,* 1988, 9(4), pp.669-686.

[CM99] Chopard, B., Masselot, A.: Cellular automata and Lattice Boltzmann Methods: A new approach to computational fluid dynamics and particle transport. *Future Gener Comp. SY,* 1999,16 (2-3): 249-257.

[CRW*03] Chuvievo, E., Riano, D., Wagtendonk, J.V., Morsdorf, F: Wildland Fire Danger; Estimation and Mapping : The Role of Remote Sensing Data. *Chapter Fuel Loads and Fuel Type Mapping, World Scientific*, 2003, pp. 119-142.

[DK00] Doellner, J., Kersting, O.: Dynamic 3D Maps as Visual Interfaces for Spatiotemporal Data. *ACMGIS 2000*, 2000, pp. 115-120.

[DKY*00] Dobashi, Y.,Kaneda, K., Yamashita, H., Okita, T., Nishita, T.: A Simple, Efficient Method for Realistic Animation of Clouds. *In Proc. SIGGRAPH'00,* 2000, pp.19-28.

[DWS97]Duchaineau M., Wolinsky M., Sigeti D.E.: ROAMing Terrain: real-time Optiamall Adapting Meshes. *In Proc. of IEEE Visualization'97*, 1997, pp. 81-88.

[Dun79] Dungan, W.: A Terrain and Cloud Computer Image Generation Model. *In Proc. Of SIGGRAPH'79,* 1979, pp.143-150.

[Ebe97] Ebert, D.S.: Volumetric Modelling with Implicit Functions: A Cloud is Born. *In Proc. of SIGGRAPH'97,* 1997, pp. 147-156.

[EMP*03] Ebert, D.S, Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: Texturing and Modeling. *Morgan Kaufmann Pulishers*, 2003.

[Fis82] Fischlin, A.: Analyse eines Wald-Insekten-Systems: Der subalpine Lärchen-Arvenwald und der graue Lärchenwickler Zeiraphera diniana Gn. (Lep., Tortricidae). *Diss. ETH No. 6977*, Swiss Federal Institute of Technology: Zürich, Switzerland, 1982, pp. 294-318.

[Fis83] Fischlin, A.: Modelling of Alpine valleys, defoliated forests, and larch bud moth cycles: the rôle of migration. *In: Lamberson, R.H. (ed.), Mathematical models of renewable resources,* Humboldt State University, Mathematical Modelling Group, University of Victoria, Victoria, B.C., Canada, 1983, pp. 102-104.

[Fis91] Fischlin, A.: Interactive modeling and simulation of environmental systems on workstations. *In: Möller, D.P.F. & Richter, O. (eds.), Analysis of Dynamic Systems in Medicine, Biology, and Ecology*, Springer, Berlin a.o., 1991, pp. 131-145.

[FB79] Fischlin. A., Baltensweiler. W.: Systems analysis of the larch bud moth system. *Part I: the larch-larch bud moth relationship*, Mitt. Schweiz. Ent. Ges. , 1979, pp. 273-289.

[FHP86] Frisch, U., Hasslacher, B., Pomeau, Yves: Lattice-Gas Automata for the Navier-Stokes Equation, *Physical Review Letters 56*, 1986, pp. 1505-1508.

[FM84] Faber, V., Manteuffel, T.: Necessary and Sufficient Conditions for the Existence of Conjugate Gradient Method. *SIAM J. Numer. Anal. 21,* 1984, pp. 315-339.

[FM96] Foster, N., Metaxas, D.: Realistic animation of liquids. *Graphical Models and Image Processing*, 1996, 58(5), pp. 471–483.

[FM97] Foster, N., Metaxas, D.: Modeling the Motion of a Hot Turbulent Gas. *In Proc. of SIGGRAPH'97*, 1997, pp. 181-188.

[FP02] Ferziger J.H., Peric M., Computational Methods for Fluid Dynamics. Third edition, Springer print, 2002.

[FSJ01] Fedkiw, R., Stam, J., Jensen, H.w.: Visual Simulation of Smoke. *In Proc. of SIGGRAPH'01*, 2001, pp. 15-22.

[Gar85] Gardener, G.Y.: Visual Simulation of Clouds. *Computer Graphics*, 1985, 19(3), pp.297-304.

[HWL*00] Hu, Y.X., Wielicki, B., Lin, B., Gibson, G., Tsay, S.C., Stamnes, K., Wang, T.: δ-Fit: A Fast and Accurate Treatment of Particle Scattering Phase Functions with Weighted Singular-value Decomposition Least-squares Fitting. *Journal of Quantitative Spectroscopy & Radiative Transfer,* 2000, Vol. 65, pp. 681-690.

[Hof00] Hoffman H.: Realistische Darstellung atmosphärischer Effekte in interaktiven 3-D-Landschaftsvisualisierungen. *Remote Sensing Series*, 2000, Vol. 35, University of Zurich. 160 pages.

[HPP49] Hardy, J., Pazzis, de O., Pomeau, Y.: Molecular Dynamics of a Classical Lattice Gas: Transport Properties and Time Correlation Functions. *Physical Review*, 1949, 13(5), 1949-1961.

[HL01] Harris, J.M., Lastra, A.: Real-Time Cloud Rendering. In *Proc. Eurographics2001*, 20(3), pp.76-84.

[IPW*06] Isenegger, D., Price, B., Wu, Y., Fischlin, A., Frei, U., Weibel, R., Allgöwer, B.: A software architecture for coupling temporal simulation systems, VR, and GIS. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2006, 60(1): 34-47.

[KCR00] King, S.A., Crawfis, R. A., Reid W.: Fast Volume Rendering and Animation of Amorphous Phenomena. *Volume Graphics*, 2000, pp. 229-242.

[KH84] Kajiya, J.T., Herzen, B.V.: Ray tracing Volume Densities. *Computer Graphics*, 1984, 18(3), pp.165-174.

[Klu02] Kluyskens, T.: Making Good Clouds. MAYA based Queen Maya magazine tutorial, http://reality-sgi.com/tkluyskens /txt/tutor6.html.

[KPH*02] Kniss, J., Premoze, S., Hansen, C., Ebert, D.: Interactive Translucent Volume Rendering and Procedual Modeling. *In Proceedings of the conference on Visualization*, 2002, 19-27.

[Lay02] Layton W. J.: A Mathematical Introduction to Large Eddy Simulation. *Computational Fluid Dynamics-Multiscale Methods*, Von Karman Institute for Fluid Dynamics, Rhode-Saint-Gen\\`ese, Belgium, 2002.

[LBY96] Leclercq, E., Benslimane, D., Yetongnon, K.: A Distributed Object Architecture for Interoperable GIS. *IEEE Knowledge and Data Engineering Exchange Workshop*, 1996, pp. 73-80.

[LC87] Lorensen, E. W., Cline, E. H.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *In Proc. SIGGRAPH'87*, 1987, pp. 163-169.

[LF02] Lamorlette, A., Foster, N.: Structural Modeling of Flames for a Production Environment. *In Proc. of SIGGRAPH'02*, 2002, pp. 729-734.

[Lev88] Levoy, M.: Display of Surfaces from Volume Data. *IEEE Computer Graph*. 1988, 8(5), pp. 29-37.

[Lev90] Levoy, M.: Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 1990, pp. 245-261.

[LL94] Lacroute, P., Levoy, M: Fast Volume Rendering Using a Shear-warp Factorization for the Viewing Transformation. *In Proc.SIGGRAPH'94*, 1994, pp.451-458.

[LH04] Losasso, F., Hoppe, H.: Geometry Clipmaps: Terrain rendering using nested regular grids. *In Proc. of SIGGRAPH'04*, 2004, pp. 769-776.

[LWK03]Li, W., Wei, X. M., Kaufman, A.: Implementing Lattice Boltzmann Computation on Graphics Hardware. *The Visual Computer*, 2003, pp. 444-456.

[MKI*05] Morsdorf, F., Kötz, B., Itten, K., Isenegger, D., Allgöwer, B. : Sensitivity of a Forest Fire Behavior Model to High Resolution Remote Sensing Data. *In EARSeL 5th International Workshop on Remote Sensing and GIS Applications to Forest Fire*. 2005, Zaragoza, Spain.

[MSH*99] Mueller, K., Schareef, N., Huang, J., Crawfis, R.: High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Trans. Visualization and Computer Graphics*, 1999, 5(2), pp. 116-134.

[MYD*01] Miyazaki, R., Yoshida, S., Dobashi, Y., Nishita, T.: A Method for Modeling Clouds based on Atmospheric Fluid Dynamics. *In Proceeding Pacific Graphics*, 2001, pp. 363-372.

[NDN96] Nishita, T., Dobashi, Y., Nakamae, E.: Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light. *In Proc. SIGGRAPH'96,*1996, pp. 379-386.

[NFJ02] Nguyen, D.Q., Fedkiw, R., Jensen, H.W.: Physically Based Modeling and Animation of Fire. *In Proc. SIGGRAPH'02*, 2002, pp.721-728.

[NMM*06] Neophytou, N., Mueller, K., McDonnell, K.T., Hong, W., Guan, X., Qin, H., Kaufman, A.: GPU-Accelerated Volume Splatting With Elliptical RBFs. *In Eurographics /IEEE VGTC Symposium on Visualization,* 2006, pp.13-20.

[PD04] Pajarola, R., DeCoro, C.: Efficient Implementation of Real-time View-dependent Multiresolution Meshing. *IEEE Transactions on Visualization and Computer Graphics*, 2004, 10(3), pp. 353-368.

[Per85] Perlin, K.: An Image Synthesizer. *Computer Graphics*, 1985, 19(3), pp. 362-378.

[PIA*] Price, B., Isenegger, D., Allgöwer, B., Fischlin, A.: Spatio-temporal modelling of Larch bud moth in the European Alps: The importance of data resolution. Landscape Ecology (submitted).

[Ree83] Reeves, W.: Particle Systems- A Technique for Modeling a Class of Fuzzy Objects. *ACM Transcations on Graphics 1983*, 2(2), pp 91-108.

[RE03] Roettger, S., Ertl, T.:Fast Volumetric Display of Natural Gaseous Phenomena. *Computer graphics international*, 2003, pp. 74-86.

[RH98] Roettger, S., Heidreich, W., Slusallek, P.H., Seidel, H.: Real-time Generation of Continuous Levels of Detail for Height Fields. *In Proc. WSCG'98*, 1998, pp. 315-322.

[PK05] Park, S. I., KIM, M. J.: Vortex fluid for gaseous phenomena. *In ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2005, 261–270.

[Rot72] Rothermel, R. C: A mathematical model for predicting fire spread in wildland fuels. *Research Paper INT-115*, Ogden, 1972, UT: US Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station.

[Sch95] Schaufler, G.: Dynamically Generated Impostors. *GI Workshop Modeling,* 1995, pp. 129-136.

[SDY02] Sato, T., Dobashi, Y., Yamamoto, T.: A Method for Real-Time Rendering of Water Droplets Taking into Account Interactive Depth of Field Effects. *In Proc. of IWEC*, 2002, pp. 110-117.

[SF93] Stam, J., Fiume E.: Turbulent Wind Fields for Gaseous Phenomena. *In Proc. of SIGGRAPH'93*, 1993, pp. 369-376.

[SH81] Siegel, R., Howell, J.: Thermal Radiation Heat Transfer. Hemisphere Publishing Corp., Washington, DC, 1981.

[SLS*96] Shade, J., Lischinski, S., Salesin, D., DeRoseT., Snyder J., Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. *In Proc. of SIGGRAPH'96,* 1996, pp. 75-82.

[SRF05] Selle, A., Rasmussen, N., Fedkiw, R.: A Vortex Particle Method for Smoke, Water and Explosions. *In Proc. SIGGRAPH'05*, 2005, pp.910-914.

[SSE*03] Schpok, J., Simons, J., Ebert, D. S., Hansen, C.: A Real-Time Cloud Modeling, Rendering, and Animation System. *Symposium on Computer Animation,* 2003, pp. 31-39.

[Sta94] Stam, J.: Stochastic Rendering of Density Fields. *In Proc. of Graphics Interface '94*, 1994, pp. 51-58.

[Sta99] Stam, J.: Stable Fluids. *In Proc. of SIGGRAPH'99*, 1999, pp. 121-128.

[Sta01] Stam, J.: A Simple Fluid Solver based on the FFT. *Journal of Graphics Tools*, 2001, Vol. 6, pp. 43-52.

[Sta03a] Stam, J.: Real-Time Fluid Dynamics for Games. *In Proceedings of the Game Developer Conference*, 2003, pp. 110-118.

[Sta03b] Stam, J.: Flows on Surfaces of Arbitrary Topology. *In Proceedings of SIGGRAPH'03*, 2003, pp.724-731

[Sto63] Stommel, H.: Varieties of oceanographic experiments. Science, 1963, Vol. 139, pp. 572-576.

[SU94] Steinhoff, J., Underhill, D.: Modification of the Euler Equations for Vorticity Confinement: Application to the Computation of Interacting Vortex Rings. *Physics of Fluids*, 1994, 6(8), pp. 2738-2744.

[SW06] Schneider, J., Westermann, R.: GPU-Friendly High-Quality Terrain Rendering. *Journal of WSCG* **,** 2006, Vol. 14, pp. 49-56.

[TK05] Signar-olaf Tergan, Tanja Keller: Knowledge and Information Visualization: Searching for Synergies. Book published by Springer on Aug. 15, 2005.

[TLP06] Treuille, A., Lewis, A., Popovic, Z.: Model Reduction for Real-time Fluids. *In Proc. of SIGGRAPH'06*, 2006, pp. 826-834.

[Tri89] Tritton, D.J.: Physical Fluid Dynamics, 2nd ed. Oxford, England: Clarendon Press, 1989.

[WAN06] Wu, Y., Allgöwer, B., Nüesch, D.: Fast and Realistic Display of Clouds Using a Recording Matrix. *In proceeding of Eurographics'2006*, 2006, short paper, pp.21-24.

[Wan04] Wang, N.: Realistic and Fast Cloud Rendering. *Journal of graphics tool*, 2004, 9(3), pp. 21-44.

[Wes90] Westover, L.: Footprint Evaluation for Volume Rendering. *In Proc. of SIGGRAPH'90*, 1990, pp. 367-376.

[WLM*03] Wei, X.M., Li, W., Mueller, K. Kaufman A.: The Lattice Boltzmann Method for Gaseous Phenomena. *IEEE Transaction on Visualization and Computer Graphics*. 2003, pp. 164-176.

[WPI*06] Wu, Y., Price, B., Isenegger, D., Fischlin, A., Allgöwer, B., Nüesch, D.: Real-time 4D visualization of migratory insect dynamics within an integrated spatiotemporal system. *Ecological Informatics,* 2006, Vol.1, pp. 179-187.

[WZF*04] Wei, X.M., Zhao, Y., Fan, Z., Li, W., Qiu, F., Stover, Y.S., Kaufman, A.E.: Lattice-based Flow Field Modeling. *IEEE transactions on Visualization and Computer Graphics,* 2004, pp. 719-729.