Visualizing Cumulus Clouds in Virtual Reality

About the cover
The front cover shows, from left to right, images from Chapters 2, 3, 4, 5 and 7 of this thesis. The back cover shows a picture of cumulus clouds over the Markt in Delft and an illustration of the Cloud Explorer virtual environment.

Visualizing Cumulus Clouds in Virtual Reality

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.Ch.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 15 juni 2010 om 14:00 uur

door

Eric James GRIFFITH

Master of Science in Computer Science
Rensselaer Polytechnic Institute, Troy, New York, USA
geboren te Long Beach, California, USA.

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. F.W. Jansen

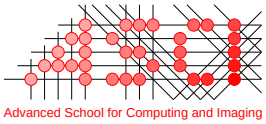
Copromotor: Ir. F.H. Post

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter
Prof.dr.ir. F.W. Jansen, Technische Universiteit Delft, promotor
Ir. F.H. Post, Technische Universiteit Delft, copromotor
Prof.dr.ir. H.W.J. Russchenberg, Technische Universiteit Delft
Prof.dr. H.J.J. Jonker, Technische Universiteit Delft
Prof.dr. J.B.T.M. Roerdink, Rijksuniversiteit Groningen
Prof.Dr. T. Kuhlen, Rheinisch-Westfälische Technische Hochschule Aachen
Prof.Dr.-Ing. W. Straßer, Eberhard Karls Universität Tübingen

Nederlandse Organisatie voor Wetenschappelijk Onderzoek

This research was sponsored by the Netherlands Organisation for Scientific Research (NWO).



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 204.

`http://graphics.tudelft.nl/~eric/`

Printed by Wöhrmann Print Service

Contents

Preface

The work presented in this thesis arose out of a research project entitled "Visualization of Cumulus Clouds in Virtual Reality." The project was a collaboration between the Computer Graphics and CAD/CAM group and the Multi-Scale Physics department at Delft University of Technology. Project funding was provided by the Netherlands Organisation for Scientific Research (NWO).

One of the central ideas of the project was using simulation techniques to study fair-weather cumulus clouds. The research goals of the project were twofold. On the data visualization side, the aim was to develop visualization tools and techniques for interactively visualizing the complex simulation data. On the atmospheric research side, the goal was to advance the basic understanding of cumulus clouds through the use of simulation and advanced data visualization techniques. Another key idea in the project was that, due to the complex, 3D, multifield, time-varying nature of the simulation data, more insight could be gained by visualizing the data in virtual reality (VR) than by visualizing the data in a standard, desktop environment.

This project has been very enjoyable for me personally to work on, and this thesis brings the cumulus cloud project to a successful close. On the way to reaching this point, I have been helped and inspired by a number of people. Without their contributions, both tangible and intangible, I would not have reached the end.

First, I'd like to thank the people from RPI that were instrumental in my decision to pursue a Ph.D. and to do so at Delft University of Technology. Srinivas Akella, Bruce Piper and Vera Kettnaker all took the time to work with me, challenge me, and encourage me to take the next step. Thanks to Harry McLoughlin for making the mechanics of my Master's degree possible, and a special thanks to Buğra Çaşkurlu for pointing me towards TU Delft.

In Delft, I have had the opportunity to work with many great colleagues. I would like to thank Frits Post for being my daily supervisor. His guidance, insight and affable manner have played a crucial role in making my time at TU Delft enjoyable and fruitful. I'd also

like to thank Erik Jansen for being my promotor and for helping me through the final stages of preparing this thesis. A special thanks to Gerwin de Haan for going through the PhD process with me and keeping it fun and interesting. Thanks also to Michal Koutek, whose dedication to VR made much of this work possible. I would like to thank Harm Jonker and especially Thijs Heus for working with us and taking the time to explain the basics of atmospheric physics. Thanks to the rest of the visualization group, in particular Charl, Jorik, Peter, Lingxiao, Stef, François and René, and thanks to my students Dylan Dussel and Torsten Stöter for our enjoyable collaboration. Thanks also to my committee members for taking the time to read and evaluate my work.

I would like to thank my family, and especially my mother and father as well as James and Beverly for their love and willingness to support me in whatever I chose to do. I'd also like to thank my friends here in the Netherlands and abroad for giving me a world outside of work so that my work could continue. Thanks to Peter, Kris, Mike Y., Mike V., and Carlos for our travels and adventures. Thanks to Fatemeh for our discussions about thesis writing. Thanks to many Delftians, Rotterdammers and Amsterdammers for navigating the Netherlands with me. Thanks again to the visualization guys for our evenings in 't Klooster. Thanks in particular to Marzena for inviting me to a wedding.

Finally, I would like to thank the NWO for providing project funding. Withouth this, there would have been no end to reach.

Eric J. Griffith
Rotterdam, 2010

CHAPTER 1

Introduction

This chapter introduces and motivates the topics covered in this thesis. First, scientific visualization and other visualization related topics are introduced. Next, a high level overview of cumulus clouds and research into their physical behavior is provided. This is followed by a discussion of visualization research challenges posed by cumulus cloud data and the approaches taken in this thesis to address these challenges. The chapter closes with an overview of the rest of the thesis.

1.1 Scientific Visualization

Visualization, in the sense of the research field, deals with converting raw data, typically numbers, into meaningful, visual representations. Some familiar examples are vital signs monitors or stock market trend graphs. By converting the data to a visual form, we can use our highly-developed visual system to make sense of the data. See Figure 1.1. This can be with the goal of gaining insight into the data but also with the goal of communicating our understanding of the data to others.

Within the research field of visualization there are several sub-fields that deal with visualizing specific types of data. Scientific visualization focuses on visualizing data from measurements obtained from simulation or experimentation. In particular, these measurements usually reflect physical quantities, such as temperature, and the spatial location where the measurements were taken is usually physically meaningful. A weather map is an everyday example of this type of visualization.

Much of current scientific visualization research attempts to provide insight into complex data. This data is often time dependent. That is, it represents a series of measurements or

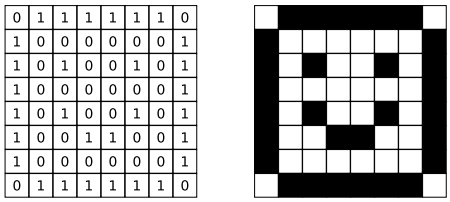


Figure 1.1: Visualization converts data to a visual form, which can be easier to understand.

calculated data in time. The data is frequently multi-field: there are multiple types of measurements in the data. The data is also commonly three-dimensional, representing measurements taken at several points in space. Television weather forecasts often use visualizations of these types of data. They show the past and expected future movement of weather patterns, which include indicators of temperature, precipitation, and so on. While the visualizations used on television are often 2D, they are based on, among other things, 3D simulations of the atmosphere.

Visualization research also deals with supporting different aspects of data analysis. Visualization can be used for browsing through data to get an overview of what's in the data. It can be used to automatically, or semi-automatically, find and visualize interesting features in the data as well as measure their properties. These modes allow users to develop and test hypotheses about the phenomena recorded in the data. When dealing with simulations, visualization can also aid in interactively adjusting simulation parameters to understand their effects or achieve a desired result. Visualization is also an effective means to share any discoveries made during data analysis with others. Analyzing MRI scan results from cancer patients is an example that involves several of these aspects. Doctors must study the scan results to determine the nature and seriousness of any tumors in the patients. Their findings are shared with the patients and, if necessary, are used for planning surgery or other treatment.

1.1.1 Visualization Pipeline

Visualization, scientific or otherwise, employs a common pipeline. See, for example, Haber and McNabb [41]. This pipeline gradually transforms raw input data into images that users can see and, ideally, interact with. See Figure 1.2. If done well, the visualization will serve as a tool for users to develop insight into and understanding of the data.

For scientific visualization, the input to the pipeline is the experimental or simulation data. In its original form, the data is usually not suitable for visualization. Instead, it must first be processed. This processing can consist of several things. If the data is noisy, it may be smoothed. If the data is too large, it may be down sampled or otherwise compressed. If the data is not in a suitable format, it may be converted. In addition to this type of processing, more advanced filtering may be applied to the data. This can reduce the amount of data that needs to be processed by identifying the interesting features and regions in the data. Further processing may also include preparing the data for use with special purpose data structures

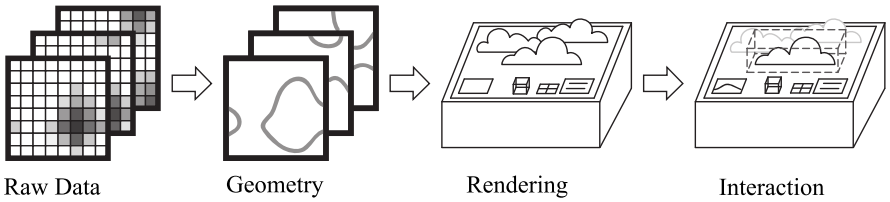


Figure 1.2: The visualization pipeline.

like kd-trees.

Once the data has been suitably prepared for visualization, it must be mapped to visualization data such as geometry and color information. This forms the basis of what the user will see. Some common techniques that are used in this process are generating isosurfaces using marching cubes [63] and using color maps to map properties onto a cross section such as a cutting plane. All or part of this step may be performed off-line, prior to visualization, but for interactive visualization applications, much of the mapping will be performed dynamically in response to the user's actions.

Traditionally, the data mapping step has been distinct from the rendering step. However, with the increasing power and programmability of graphics processing units (GPUs), there has been a trend to perform both steps at once on the GPU. Examples of this include GPU-based ray casting for volume rendering, fast line integral convolution for flow visualization, and, more recently, generating isosurfaces dynamically by executing marching cubes on the GPU. Several of these techniques are detailed by Weiskopf [106].

The rendering step takes the geometry and color information generated from the visualization data and generates the images that the user actually sees. For rendering techniques like ray tracing and ray casting, the image will be generated by directly mapping the data to color information without using geometry. For other techniques, the color information will be applied to the geometry during rasterization on the GPU.

It is also important to integrate user interaction into the system. Without interaction, the visualization result is either a static image or a pre-rendered movie. For some applications, this may be sufficient, but frequently the user will be interested in viewing the data from multiple angles or seeing the effects of changing the visualization parameters. The results of user interaction can feedback into all stages of the pipeline. The user can identify new, interesting regions of the data to study. He can generate isosurfaces for new isovalues. He can also simply view the current data from a new angle. An effective visualization platform should give the user freedom to interact with his data in meaningful ways so that he can gain insight into his data.

In some cases, it is also possible to take user interaction one step further and interactively couple it with a simulation that generates the data for visualization. This is known as computational steering and can allow the user to directly influence the simulation results and the related visualization. See Section 1.1.5 for more about this.

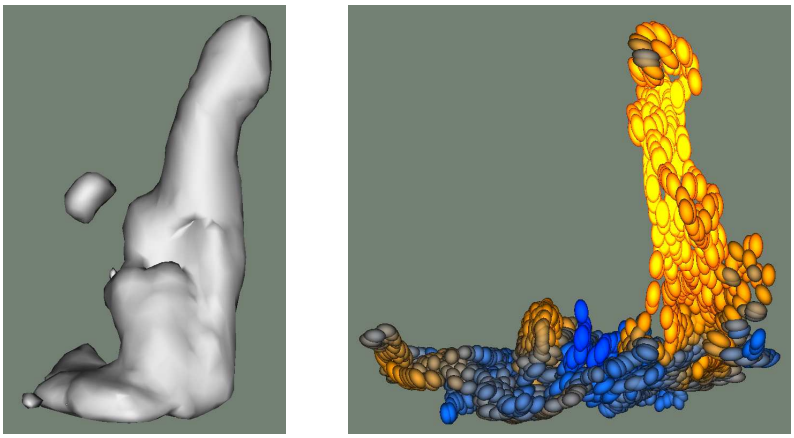


Figure 1.3: Left: isosurface of a cloud generated using the techniques described in Chapter 2. Right: ellipsoid particles in a cloud velocity field generated using the particle tracing techniques described in Chapter 5.

1.1.2 Flow Visualization

There are several specialized application areas under the umbrella of scientific visualization, such as medical visualization [77,81] and flow visualization. Since the dynamics of the atmosphere can be seen as fluid flow, flow visualization is the most relevant research area for this thesis. Flow visualization deals with both effective ways of visualization flow and techniques for identifying features of interest in the flow, like shock waves or vortices. Post et al. [76], Laramee et al. [61], McLoughlin et al. [67], and Laramee et al. [62] provide overviews of many flow visualization techniques. Mallinson [65] also discusses flow visualization techniques with a focus on how derived data can be used to improve the understanding.

Feature detection is used to find the interesting features in the data. If the data is time dependent, then feature detection is performed on each time step in the data. To identify correspondences in time between the features, feature tracking is used [76]. Once this process is complete, individual features can be followed through time in the data set.

In flow visualization, these features are typically visualized using isosurfaces or glyphs. Isosurfaces are used when the features are clearly defined in the data and their shape gives insight into their behavior. See Figure 1.3, left. Glyphs, such as ellipsoids, are used for more abstract features or to convey additional information. These techniques can also be combined using techniques like line integral convolution [108], to display information on the feature isosurfaces.

To directly visualize flow, two commonly used techniques are cutting planes and particle tracing. Other techniques that can be used (see [44]) include arrow plots or volume visualization of the vector field, isosurfaces of vector field components or derived values, and texture-based approaches such as line-integral convolution. Cutting planes, also called slicing planes, are used to visualize a particular scalar or vector quantity in the flow. The plane

is placed in the data and is colored or decorated with glyphs according to the flow properties along the plane. Particle tracing gives deeper insight into the nature of the flow by tracking massless, virtual particles through the flow. The particles can be visualized in a variety of ways to show their movement through the flow or to emphasize certain flow characteristics, such as vorticity or velocity in a particular direction. See Figure 1.3, right. They can also be combined together to form flow curves, such as pathlines, streamlines and streaklines [75], which can highlight flow patterns in time or space.

1.1.3 3D, Multivariate, Time-Dependent Visualization

Much scientific data is three dimensional, multivariate and time dependent, representing the state of some 3D object or volume at different points in time. Effectively visualizing such data presents challenges throughout the visualization pipeline due to the size and complexity of the data. Visualization research dealing with this kind of data often focuses on techniques to overcome these challenges. Bürger and Hauser [14] discuss a wide variety of visualization approaches used for complex data.

The size of the data itself is challenging to deal with where a data set of several gigabytes can be considered a small data set. The raw data is often unsuitable for efficient processing and must be first converted or reordered on disk in a more optimal way. Efficient processing can also be facilitated by storing the data on media with high bandwidth and low access times such as disk arrays or solid-state memory. This size of the data sets also means that they often greatly exceed the memory capacity of the computers used to visualize them, which is a further complication.

Aside from problems of data size, there are problems of data complexity. The interesting features in the data tend to be complex and dynamic. This can make it difficult to find and track the features in the data. The 3D, dynamic features can also be awkward for the user to interact with inside of a visualization application. Additionally, the wealth of information in the data can leave scientists at a loss as to where to start. They can extract features, calculate feature attributes, derive new data from the raw data, perform combinations of these and many more tasks. In addition, they may have to examine multiple variables in the data to make sense of it, each of which may lend itself best to different visualization methods.

Another difficulty arising from such data sets is time related. The simulations or experiments required to generate the data must be set up and executed. The resulting data must be processed to prepare it for visualization. In general, neither of these steps can be performed interactively. That is, if the scientists studying the data wish to alter the parameters that generated the data or change the data that is visualized, they must re-initiate the required steps and wait for them to complete.

An effective, interactive visualization system for 3D, multivariate, time-dependent data should address these challenges. To deal with data size, the data should be processed and filtered such that the minimum necessary amount of data is loaded for visualization, and the system memory should be creatively used and filled during visualization to maintain interactivity. To address the complexity, the system should present data features with sufficient contextual and derived information such that scientists can develop an understanding of the

data. Scientists should also be able to use appropriate visualization techniques for visualizing the different variables in the data. The system should also be extensible so that scientists can introduce new data into the visualization based on insight they have gained into the data. The system must also provide users an intuitive interface so that they can easily interact with the data, and the system must provide adequate exploration tools for the data to be studied. Lastly, the system should attempt to optimize the way researchers spend time. It should shorten the time required to adjust parameters both for generating the raw data and for generating the data for visualization.

1.1.4 Visualization in Virtual Reality

Virtual reality (VR) is a generic term that is applied to the experience that some computer systems can create where users have the impression that computer-generated objects and environments are "real". The more real these virtual objects and environments feel to the users, the more "immersive" the VR system is said to be. The "realness" of objects includes both their appearance as well as how they respond to user actions. VR systems can consist of a variety of components and use several technologies, but required features are presenting the user with a 3D, virtual environment and allowing the user to directly interact with the environment. One important caveat to virtual reality is that VR systems must be "interactive" in order to maintain the feeling of immersion. This implies two things. First, the user should be able to touch, manipulate, activate, and otherwise interact with objects in the world in a "natural" way. Second, the VR system must update the user's view on the world at least 10 and ideally 30 or more times per second to provide sufficient feedback about the user's interactions with the virtual world.

Most VR systems show users 3D environments in stereo, i.e. a different view of the environment is presented to each of the user's eyes so that objects have a feeling of depth. There are two common approaches to this: head-mounted displays (HMDs) and projection-based systems. HMDs are special goggles with two small built-in screens for the left and right eyes. Projection-based systems use projectors to project two different images onto a real surface, e.g. a screen on a wall. The two images are shown either using two projectors with special filters in front of them (passive stereo) or by using one or two projectors to rapidly alternate each of the two images (active stereo). For both systems, users must wear special glasses to ensure that each eye sees a different image.

VR systems typically enable users to directly interact with the virtual environment by tracking their physical movements. The tracking is usually done by tracking the 3D location and orientation of the user's head and various interaction tools like styli and wands. Tracking the user's head allows the VR system to render the 3D world from the user's viewpoint, which both gives the user the correct perspective on the world and lets the user move his head around to view the world from different vantage points. Tracking interaction tools allows the user to point at, select, and otherwise interact with the objects he sees in the virtual world.

Several factors make virtual reality attractive for visualizing 3D data. By providing users with a stereo view of the 3D simulation or measurement data domain, and particularly time-varying 3D data, it's easier for them to understand where features of interest are in space and

where they are in relation to each other. The direct interaction that VR provides also simplifies interacting with the data. Tasks like getting a different view of the data can be reduced to moving one's head. Features in the data can be selected by, e.g. pointing at them with a stylus. Once selected, a user can intuitively move and rotate objects by moving and rotating his hand. Since VR applications must be interactive, this can also ease the visualization process by providing users interactive feedback in response to their actions. The immersion and "wow" effect that VR provides are also an aid, particularly for novice users, in understanding 3D data. Van Dam et al. [20] and Koutek [58] discuss several uses of VR in scientific visualization. Bryson [12] also gives a good, if dated, introduction to scientific visualization in VR and its related challenges.

1.1.5 Simulation and Visualization

In many applications, numerical simulation and visualization are closely intertwined. Numerical simulation involves running specialized computer programs that solve mathematical equations describing the behavior of real-world phenomena, such as fluid flows. Simulating the real-world phenomena allows scientists to explore their behavior under a variety of conditions simply by changing simulation parameters. The results of simulations, however, are often difficult to interpret without specialized tools. Visualization is one of the tools scientists often turn to in order to help them understand their simulation results.

The role of visualization is not limited to analyzing the results produced by a simulation run, though. Visualization can also provide valuable insight in constructing the simulation software itself. A visual inspection of the simulation in progress can reveal bugs in the mathematical model or software implementation that are more difficult to identify using traditional debugging tools. Similarly, when adjusting the simulation parameters to simulate new case studies, visualization can be used to verify that the simulation is behaving correctly.

For numerical simulations that are sufficiently fast, scientists can use visualization to help adjust and control the simulation. This is called computational steering [54]. That is, by coupling the simulation with an interactive visualization, scientists can dynamically adjust the parameters of a running simulation. This can allow scientists to rapidly explore the effects of different parameters and also to guide the simulation towards a desired state. Koutek [58] describes an example from molecular dynamics simulations where scientists can exert forces on protein molecules to guide their folding. Wright et al. [109] provide a general overview of computational steering and Mulder et al. [69] discuss the components of various computational steering environments.

For some simulation research, a combination of simulation and visualization techniques is useful. For the initial work of determining useful simulation parameters for a new case study, it's useful to have a very fast simulation that is coupled with an interactive visualization. If the effects of interactive parameter changes are readily apparent when visualizing the simulation, then introducing computational steering capabilities can further help in identifying interesting parameters.

Once suitable simulation parameters have been identified, scientists usually wish to run much longer, much higher resolution simulations than the initial experimental simulation

runs. At this stage, it is less interesting to directly monitor the simulation progress so the simulations can be run off-line with the data stored for later analysis. Once the simulations are complete, it is still useful to visualize the final results of the simulation. This is especially true since a wider variety of visualization techniques, particularly related to feature tracking, can be employed when the data from the entire simulation run is available. Having the full data set available also makes it easier to browse both forward and backward through time, in time-dependent data sets, which is more difficult when running the simulation interactively.

1.1.6 Open Problems

There are many open problems in the field of scientific visualization that are the subjects of ongoing research. One of the better known discussions of these problems is provided by Chris Johnson et al. [53]. These problems are posed, in large part, by the increasing size and complexity of the measurement and simulation data that scientists wish to visualize. There are also challenges stemming from making efficient use of the technology available for visualizing the data. Another group of problems relates to effectively enabling domain scientists to visualize their data.

Currently, scientific data is often 3D, multifield and time-varying. Each of these types of data poses unique challenges, and, taken together, the challenges are only more complicated. Visualizing 3D data on a normal computer monitor can lead to difficulties in correctly perceiving the depth of features in the data and their spatial relation to each other. Features closer to the viewer can obscure or completely occlude features that are farther away from the viewer. With multifield data, it is difficult to provide a meaningful visual representation of two or three data points at a given spatial location. Furthermore, different fields in the data may be best suited to differing visualization techniques, which requires combining the disparate techniques together into a unified visualization. It's also difficult to visualize the spatial correlation between the different fields in the data. With time-varying data, it is challenging to give users accurate insight into how features in the data evolve and change over time. Another difficulty is helping users correlate the instantaneous 2D or 3D state with the larger picture in time. Time-varying data also brings the added complexity of tracking features detected in the data in time.

Making efficient use of technology in combination with visualization is another area of visualization research. On one side, there are a wide variety of input and output devices for use with visualization. Some commonly used output devices include standard computer monitors, stereoscopic displays, speakers, and speaker arrays for 3D audio. Some commonly used input devices include mice, keyboards, haptic devices, and 3D tracked devices like wands or styli. When attempting to put together an effective visualization environment, decisions must be made about which components to incorporate into the system. Factors like cost versus added benefit must be weighed carefully. Furthermore, once the devices are incorporated into a system, they must be combined into a user-friendly interface that allows users to focus on visualization tasks. On the other side, there is a wide variety of computational technology available for driving the visualization. These aspects include GPUs, computing clusters, disk arrays, and so on. Visualization software may be designed to seamlessly take advan-

tage of additional computing facilities that are available or be designed to make optimal use of one particular type of facility. Both approaches require significant effort and a thorough understanding of the hardware involved.

The third, broad category of research challenges is enabling domain scientists to visualize their data. One of the most basic challenges in this area is learning how to effectively cooperate with domain scientists to develop techniques that are useful to them. When developing new visualization techniques for domain scientists, it's also important to bear in mind that they are not visualization experts. Therefore, it's important to present them with intuitive user interfaces that make it easy for them to adopt the new techniques. Another challenge is increasing both the quantity and the quality of the time that the domain scientists spend using visualization tools. When visualizing their data, scientists should have the freedom to explore the data, develop hypotheses and test these hypotheses with minimal effort.

1.2 Cumulus Clouds

One of the goals of atmospheric research is to make predictions about what the weather and the climate will be like in the future. These predictions are often based on computer-driven simulations of the atmosphere, which make use of mathematical models of different atmospheric phenomena and their effect on the weather and climate. Climate simulations often build on weather simulations by using information learned from the weather simulations to model the long term effects of the weather. Perhaps surprisingly, the effect of clouds, such as cumulus clouds, on an evolving climate is one of the biggest unknowns in the models (Bony et al. [8], Heus [47]). Therefore, to improve climate prediction, it is important to better understand clouds. For clarity, the cumulus clouds discussed here are fair-weather cumulus clouds, such as those seen in Figure 1.4.

The traditional view of cumulus clouds is that they are the visible top of a column of rising, warm, moist air. As the air rises, it begins to cool, and, at a certain height, the moisture in the air condenses to form the cloud. As long as the thermal, the column of warm air, exists, the cloud also continues to exist. Once the thermal dies out, the cloud also dies out. This evolution of a cloud from its inception to when it dies out is referred to as the cloud life cycle. See Figure 1.5.

While the atmosphere underneath the fair-weather cumulus clouds, the sub-cloud layer, is turbulent, the atmosphere is fairly calm where the clouds themselves form. The traditional view holds that, in this calmer region, the clouds represent a forceful, upward protrusion of air from below. In response to this upward force, the atmosphere between the clouds then settles downward toward the more turbulent, sub-cloud layer. See Figure 1.6.

1.2.1 Research Topics of Interest

There are still fundamental questions about the behavior of cumulus clouds without definitive answers. Attempting to answer two such questions has been a motivating factor in much of the work related to the project. The first question is: what are the defining characteristics of

Figure 1.4: Fair-weather cumulus clouds in Delft.

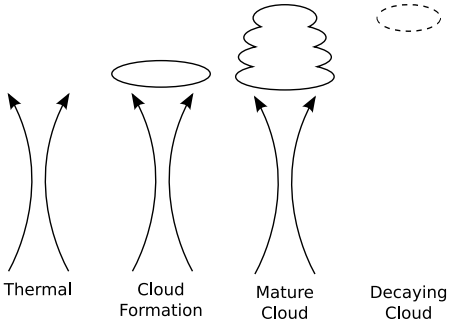


Figure 1.5: Traditional view of the cumulus cloud life cycle.

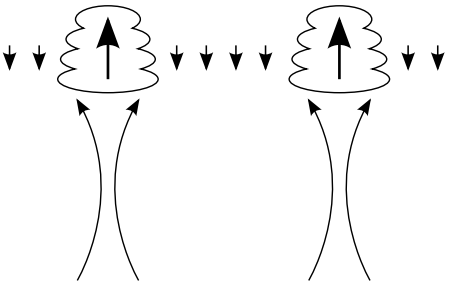


Figure 1.6: Traditional view of the subsiding air around cumulus clouds.

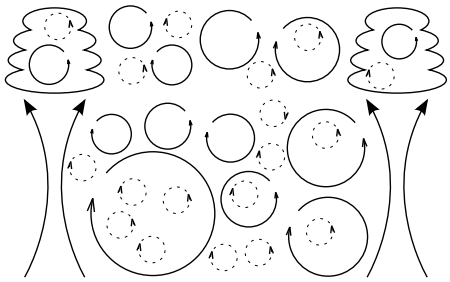


Figure 1.7: Large-Eddy Simulation resolves large-scale turbulence (solid eddies) but models small scale effects (dashed eddies).

cumulus clouds in the life-cycle stages depicted in Figure 1.5? The second question is: how does a cumulus cloud interact with and influence the dry air around it? Or, alternatively, how does the dry air around a cumulus cloud compensate for the upward force from the buoyant cloudy air?

Collecting the necessary data to answer these questions can be done through either observing real cumulus clouds or through simulating cumulus clouds. While observation is a popular approach, simulation offers many advantages when performing large-scale studies of many clouds. Simulations are less expensive, more easily repeatable, and they provide significantly more data about the clouds and their environment. Furthermore, observation data often requires processing before it can be analyzed while simulation data can be analyzed directly. For simulation, though, observation is necessary for validation, setting simulation parameters and so on.

The work presented in this thesis has been used by atmospheric scientists to help answer these questions. Their findings have departed from the traditional views presented in preceding section. Where appropriate, results from their research are highlighted in this thesis together with descriptions of how the work presented here played a role in the research. See, for example, Section 2.5.2, Section 3.5, and Section 6.5.

1.2.2 Large-Eddy Simulation

Atmospheric scientists use a variety of simulation techniques to study cloud dynamics. The technique most relevant to the work presented in this thesis is Large-Eddy Simulation (LES). LES is so named because it directly simulates only large-scale turbulence and eddies. The effects of smaller scale eddies are calculated using a computationally cheaper statistical model. See Figure 1.7.

For studying clouds, LES is used to simulate the large-scale dynamics of a small portion of the atmosphere under certain conditions. These dynamics include the wind, the temperature, water evaporation and water condensation. By accurately simulating these dynamics, clouds will form in the simulation, which can then be analyzed.

The challenge associated with LES is that it produces very large amounts of data. A small to medium-sized simulation run can simulate a $6.4 \times 6.4 \times 3.2\,\text{km}$ portion of the atmosphere over the course of several hours. It does this by dividing the space into $128 \times 128 \times 80$ discrete cells and generating and saving snapshots, 6 seconds apart, of quantities like the average temperature, wind speed, and humidity in each discrete cell. Depending on the size and duration of the simulation, this can add up to anywhere from 10 gigabytes to several hundred gigabytes or more of data. The process of identifying, analyzing and understanding the interesting phenomena in the data is difficult without an appropriate set of tools. Developing such a set of tools is the focus of this thesis.

1.3 Visualizing Cumulus Clouds in Virtual Reality

In order to visualize cumulus clouds, many of the open problems in scientific visualization must be addressed. The cumulus cloud data is very large, 3D, time-varying, multifield data. Typical data sets have 6 variables, hundreds of time steps and per-variable time-step sizes of 2 to 20 megabytes. The data variables consist of both scalar and vector quantities.

Visualizing this data has challenges at each stage of the visualization pipeline. The data processing must be able to detect and track features in the data. It must also support extracting new data based on insight gained from visualization. When mapping the features to geometry, care must be taken to overcome bottlenecks between storage and main memory and between main memory and the GPU. When visualizing the data, the additional processing power GPU must be effectively used to provide rich and interactive visualizations. These visualizations must also make the relationship between multiple variables, such as liquid water and air velocity, clear. Where appropriate, VR should also be used to enhance insight into the data and ease interaction with the data.

Perhaps the most difficult challenge with visualizing cumulus clouds is embedding new techniques into the atmospheric science workflow. Close collaboration with atmospheric scientists is required for developing useful visualization tools. Furthermore, the visualization tools must be presented to the scientists in such a way as to minimize the adoption threshold. The tools must be easy to use, and the software environment they run in must also be easily accessible and easy to use. When looking toward the future, the challenge of combining the simulation and visualization together into one interactive application must also be addressed.

1.3.1 System Requirements

Answering scientific questions about cumulus clouds using data from Large-Eddy Simulations requires a system with certain components in place. These required components allow scientists to interpret the raw simulation data at a higher level. That is, they can study the clouds, their properties, and their interaction with the atmosphere rather than working with the vast arrays of numbers that the raw data consists of.

The first major required component is a virtual environment where scientists can see the clouds in the data. Within the cloud viewing environment, scientists must be able to browse

through time in the simulation data and focus their attention on interesting clouds. Browsing through time is a logical extension of being able to see the clouds. This allows scientists to visually perceive patterns in the clouds as they evolve over time. It also allows them to relate their observational experiences of clouds with the behavior of the simulated clouds. Being able to focus on interesting clouds is important for developing a detailed understanding of individual cloud behavior. The focusing should entail excluding uninteresting clouds from view as well as providing additional information about the interesting clouds.

In addition to the viewing environment, scientists need a set of visualization tools to study the clouds with. These tools should also be specific to the research questions the scientists wish to investigate. In order to study cloud life cycles, scientists need to be able to see and understand the cloud behavior in time. The clouds can be visualized with isosurfaces or direct volume rendering. Their behavior in time can be directly observed by playing through the data in time and also by supplementing the "realistic" 3D cloud view with statistical plots of cloud properties over time. In order to study the motion of air along cloud boundaries, the scientists need to be able to visualize air movement and clouds simultaneously. The air movement can be visualized using particle tracing and it can be related to the clouds by including transparent or wireframe cloud isosurfaces.

From the practical side, the scientists also need a (graphical) user interface for interactively controlling the visualization. For interaction, the scientists should be able to use the available input devices. These are the mouse and keyboard for desktop versions of the visualization and direct interaction tools like a stylus for the virtual reality environment. With these devices, he should be able to select, rotate, and zoom in and out on clouds of interest. He should also be able to play through time. The visualization options can be controlled from within the visualization environment by incorporating various buttons and menus.

There are many factors involved in developing a system to meet these requirements. New and existing visualization techniques must be developed and adapted to deal with the complex nature of the data. Atmospheric scientists must be consulted to refine the visualization techniques for atmospheric data. The set of visualization techniques must be integrated with each other and be made usable for the scientists. The combination of these factors makes it an interesting challenge for a visualization researcher to address.

1.3.2 Existing Visualization Systems

There are many existing visualization systems. The existing systems broadly fall into two categories: general purpose and special purpose. The general purpose systems take a Swiss army knife approach to visualization, where the tools they offer can be applied to a myriad of problems. Special purpose systems are by definition more limited in their application areas, but they provide superior tools for their area of specialization.

General purpose systems incorporate a wide variety of visualization and data processing algorithms in the form of components, which are combined together to suit the user's needs. In these systems, the burden is often on the user to combine the components together to create a visualization pipeline for the data he wishes to visualize. Most of these systems present users with the algorithms as building blocks with input and output ports. These ports

are connected together in a network or tree structure to pipe the raw data through various processing steps and ultimately generate a visualization from it. Well known examples of this type of system include OpenDX [64] and AVS [101]. ParaView [4, 45], which builds on top of the Visualization Toolkit [86] and Amira [94] are more recent examples of this type of system.

Special purpose systems focus more on specific types of data or applications. This restriction allows them to exploit additional knowledge about the data that will be visualized in order to offer improvements over general purpose systems. These improvements can take the form of, among others, simpler user interfaces, better processing and visualization performance, and better visualization quality. One such system relevant to this thesis is Vis5D [49]. Vis5D was not wholly specialized, but it was primarily used for visualizing data from numerical weather simulations. Ziegeler et al. present a more directly relevant system for visualizing meteorological data in VR: MetVR [110]. Haase et al. describe VISUAL and other systems used by the German Meteorological Office for visualizing meteorological data [40].

While there are many existing general-purpose visualization systems, none are particularly well-suited to visualizing the type of cumulus cloud data discussed in this thesis. Systems like OpenDX and AVS tend to suffer from being jacks of all visualization tools but masters of none. To create effective visualizations, users must navigate a complex user interface to construct a complex network or tree of visualization components. Once the pipeline is in place, the visualization tends to be static or difficult to dynamically control and often suffers from poor performance. The support for time-varying data in these systems is also generally poor. These types systems can usually be extended by developing and incorporating new modules, but it is difficult to overcome their performance limitations.

There are also no suitable special purpose systems available. Vis5D could have developed into a reasonable candidate, but it has a number of limitations. Development work stopped on it several years ago, which means it is not taking advantage of the possibilities offered by modern GPUs. Its user interface is very complex, making it difficult to use. Vis5D along with MetVR and VISUAL focus more on numerical weather data, which deals with phenomena on a scale of tens or hundreds of kilometers whereas the clouds discussed here are on a scale of tens to hundreds of meters. This difference of scale and focus led to incorporating visualization tools, such as displaying weather data over cartographic maps, which are not relevant for cloud research. VISUAL and MetVR also focus on more complicated data input formats such as curvilinear grids or unstructured grids. Data in these formats requires extra care that is not necessary for the cumulus cloud data presented here.

1.3.3 Cloud Explorer and GALES

In this thesis, two specialized approaches have been taken with visualizing cumulus cloud data. These approaches have culminated in the creation of two experimental visualization systems: Cloud Explorer and GALES. Figure 1.8 provides a high level comparison of these systems. These have served as platforms for developing visualization techniques to deal with complex, atmospheric data. They have also been targeted at meeting visualization needs of atmospheric scientists by providing interactive visualization and exploration environments

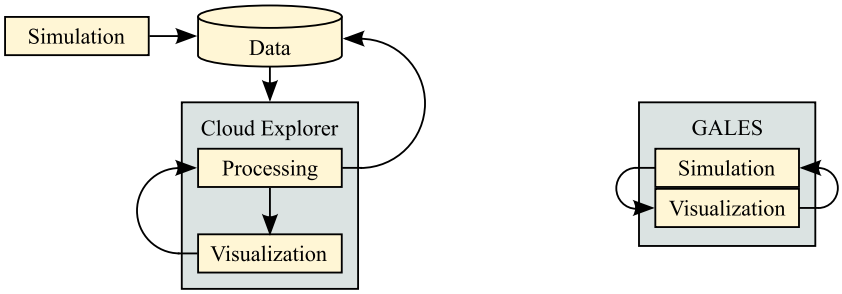


Figure 1.8: Left: Cloud Explorer forms part of the traditional, simulation-processing-visualization pipeline. Right: GALES is the fusion of simulation and visualization.

with sufficient visualization tools and interaction methods to help the scientists understand the data. The developments of this thesis have not been limited to Cloud Explorer and GALES, however. A number of the visualization techniques discussed in this thesis have been also incorporated into stand-alone experimental tools, such as a particle tracing environment, to demonstrate their wider applicability.

Cloud Explorer, Figure 1.8, left, fills the traditional role of a VR visualization system. It consists of two separate components: data processing and visualization. The data processing is an off-line process that prepares the raw simulation data for visualization. The processing can take place on the supercomputers that ran the original simulation or on a local computer. The visualization environment offers a variety of visualization tools mentioned in this chapter: slicing planes, particle tracing, isosurfaces, and supplementary statistical plots. It also allows the user to browse through time, select interesting clouds and switch between different visualization modes. This environment itself can run as either a virtual reality application or as a stand-alone, desktop application.

GALES, Figure 1.8, right, is an integrated, interactive simulation and visualization environment. It runs the atmospheric Large-Eddy simulation that generates the cloud data on the GPU of the visualization computer. The simulation is sufficiently fast that it can be interactively visualized while running. GALES also provides scientists with some statistical plots that give information about running simulations. This eliminates the lengthy simulation and processing steps in the Cloud Explorer pipeline, which allows scientists to get immediate feedback about their simulations.

With the recent advances in GPU computing power, interactive simulation approaches like GALES are feasible, but GALES is not a replacement for Cloud Explorer. Cloud Explorer can process data for large spatial domains. It can also do advanced data processing and feature tracking as it has access to all time steps of a simulation. GALES, in contrast, is limited to smaller spatial domains and can only move forward in time through the simulation.

Cloud Explorer and GALES can be best used in combination with each other. GALES is more suited to the initial, experimental stages of cloud research when scientists are experimenting with new mathematical models, simulation parameters and case studies. These

can be quickly tested in GALES's interactive environment. Once they have been finalized, longer, higher-resolution simulations can be run off-line on supercomputers. Cloud Explorer can then be used to study the final data in more detail.

1.3.4 Research Contributions

The research contributions in this system touch on a number of areas. From a high level perspective, this thesis presents a complete system, including processing and visualization, for cloud visualization that has been motivated by the visualization needs of atmospheric scientists. The system has been a means to develop, integrate, and experiment with several visualization techniques and interfaces. Many of its components are novel, represent improvements over what is available in the literature or have been specially targeted at atmospheric research. These include feature detection and tracking for semi-automatic cloud selection, data processing and reprocessing for generating visualization data, interactive, GPU-accelerated particle tracing for air motion analysis, and a VR visualization environment visualizing the data in. Enhancements to the data handling include region-of-interest extraction for feature-specific data processing and multi-resolution particle tracing. They also include bounded-error vector field compression to help overcome bottlenecks in moving data from storage to the GPU. Enhancements to the visualization environment include integrating the particle tracing and incorporating 2D statistical graphs providing contextual information. The UI of the visualization environment has also been designed for ease of use and for working both as a VR application and a desktop application. This thesis also presents the first GPU-accelerated atmospheric LES, GALES, which provides direct visualization of running simulations.

1.4 Thesis Structure

The structure of this thesis parallels the logical progression of the research presented. In Chapter 2, the groundwork for Cloud Explorer is laid. Detecting and tracking the clouds in the data is discussed and the initial visualization environment for viewing the clouds is presented. Chapter 3 builds on Chapter 2. The processing stage is extended to support generating derived data from the simulation data. The visualization environment is updated to support displaying the derived data. The UI is improved and slicing planes are also introduced into the environment. Chapter 4 addresses one of the major bottlenecks in the visualization system: getting data from disk to the video card. Vector compression is explored as an option for reducing the size of isosurface geometry and velocity field data on disk so that it can be more efficiently sent to the GPU. In Chapter 5, GPU-based particle tracing is discussed. The particle tracing is incorporated into Cloud Explorer and other stand-alone visualization applications as a means to interactively visualize the flow in and around the clouds. Chapter 6 presents the final overview of the Cloud Explorer system and how it was used to answer cumulus cloud research questions. Chapter 7 introduces GALES, the GPU-based Atmospheric Large-Eddy Simulation, which combines simulation and visualization. Chapter 8 concludes the thesis and provides an overview of future directions the work can take.

CHAPTER 2

Feature Tracking in VR for Cumulus Cloud Life-Cycle Studies

This chapter was originally presented as a peer-reviewed paper [36] at the Eurographics Workshop on Virtual Environments in 2005.

Abstract

Feature tracking in large data sets is traditionally an off-line, batch processing operation while virtual reality typically focuses on highly interactive tasks and applications. This paper presents an approach that uses a combination of off-line preprocessing and interactive visualization in VR to simplify and speed up the identification of interesting features for further study. We couch the discussion in terms of our collaborative research on using virtual reality for cumulus cloud life-cycle studies, where selecting suitable clouds for study is simple for the skilled observer but difficult to formalize. The preprocessing involves identifying individual clouds within the data set through a 4D connected component labeling algorithm, and then saving isosurface, bounding box, and volume information. This information is then interactively visualized in our VR Cloud Explorer with various tools and information displays to identify the most interesting clouds. In a small pilot study, reasonable performance, both in the preprocessing phase and the visualization phase, has been measured.

2.1 Introduction

Feature tracking and virtual reality often have conflicting requirements. Virtual reality must be interactive to be effective and maintain a feeling of immersion. Feature tracking often re-

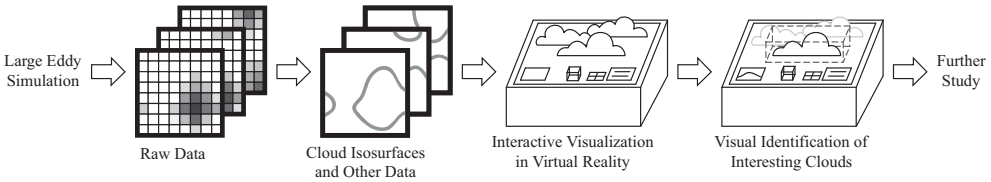


Figure 2.1: Raw data is generated by the LES, which, in turn, is processed to identify and track clouds in the data and produce isosurfaces and other data for them. The cloud isosurfaces and other relevant data are visualized in our virtual reality Cloud Explorer. A skilled user browses through the cloud field interactively and identifies interesting clouds for further study.

quires reading and processing large, time-dependent data sets. Limited reading speeds from storage generally mean that feature tracking cannot be done in real-time. However, certain types of features are relatively simple to track in software, but it is difficult to determine which of those are actually interesting to researchers. In such cases, visualization enables scientists, whose backgrounds are in observational studies, to use their perceptual skills to select features worthy of further study. Traditional scientific visualization, though, can be awkward and unwieldy when dealing with 3D features and, more so, when those features are also timedependent. VR, on the other hand, brings 3D data sets to life in three dimensions, which evokes a stronger perceptual response. This, coupled with the more natural and direct 3D interaction it provides, makes VR an attractive choice for dealing with 3D data sets. It is our claim that scientific visualization in virtual reality is the logical choice when aiding scientists in identifying the most interesting, 3D, time dependent features to investigate further.

This study arose out of our work on simulating cumulus clouds with Large-Eddy Simulations, or LES, and using VR visualization to explore the results. The long term goals of this collaborative project are to gain better insight into cloud dynamics through interactive exploration in VR. One of the first hurdles to be overcome in the project, though, is the selection of suitable clouds to study.

The size of the data sets and the nature of the clouds make it challenging to identify interesting clouds. The clouds develop unpredictably over the course of a simulation run, and it is common for pieces of clouds, or entire clouds, to merge together or split apart. These factors make it difficult to even track individual clouds through the data set. However, for an individual cloud to be interesting, it must satisfy certain qualitative criteria, such as being of sufficient size, going through the proper life-cycle stages, being a relatively cohesive, and not merging with other clouds. These properties are difficult to put into a consistent and automated algorithm. The situation is further complicated by data sets that are several gigabytes, and even up to 1 TB or more, in size. These difficulties are frustrating for atmospheric scientists because they are accustomed to selecting clouds based on observation. When they are presented with the opportunity to observe the clouds evolve over time, it is trivial for them to identify the most interesting clouds.

Traditionally, cloud selection for research has been accomplished through the painstak-

ing efforts of atmospheric scientists. We have streamlined the process with a combination of preprocessing and interactive visualization in virtual reality. Individual clouds evolving over time are identified and tracked in preprocessing through a 4D connected components algorithm, and then isosurface, bounding box, and cloud volume data are generated. This data is then visualized interactively in our Virtual Reality Cloud Explorer, which provides various tools and information displays. Scientists using Cloud Explorer are then able to select certain clouds for further study. Using the connected components data, we are then able to extract only those portions of the data containing the interesting clouds. See Figure 2.1. This data can then be used to study the cloud dynamics over the course of the life-cycles by, for example, generating mass flux plots or velocity profiles. The authors have successfully employed Cloud Explorer in a small pilot study to identify interesting clouds, and reasonable performance, both in preprocessing and in visualization, has been measured.

The remainder of this paper is structured as follows. The next section provides an overview of feature tracking, LES, and other attempts to visualize clouds. The data preprocessing phase is discussed in Section 2.3. Cloud Explorer is described in Section 2.4. Some results are presented in Section 2.5, and the paper concludes with Section 2.6.

2.2 Background

2.2.1 Feature Tracking

Feature extraction is an approach to visualizing very large data sets. It entails detecting structures or objects within the data that are of particular interest to scientists for a given research task. The features and their properties can be represented in a format that is both suitable for interactive exploration and much more compact than the original data. In fact, data reduction ratios of up to 10^3 or 10^4 can be achieved.

Feature tracking is the extension of feature extraction to time varying data sets. Applying feature extraction to time varying data sets involves more than just extracting the features from each time step. The correspondence between features in subsequent time steps must also be determined. By tracking the features across time steps, the temporal behavior of the features can be studied. A further step can be the detection of important changes in the lifetime of features. These changes, or so-called events, can be the appearance or disappearance of features, but they can also be the merging together or splitting apart of features.

Several approaches to feature tracking and event detection have been published. Correspondence between features can be determined by extracting the features directly from the 4D spatio-temporal domain [105] or by searching subsequent frames for corresponding features. This can be done by looking at the spatial extents of the features [89] or by calculating feature attributes and searching for the feature with the most similar attributes in the next frame [79]. For a survey of techniques, see [76].

2.2.2 Large-Eddy Simulation

LES is a popular numerical tool in the atmospheric sciences to give insight into characteristics of flows in the Atmospheric Boundary Layer, or ABL, where observational methods, e.g. by satellites, airplanes or radar, are limited because they do not give access to the entire 3D flow field. In LES, the Navier-Stokes equations are solved up to a certain scale. In this way, the largest and most energetic flow circulations, or eddies, are resolved, and the influence of smaller eddies are approximated via a statistical model. This type of simulation was developed during the sixties and seventies, e.g. [91], to solve turbulent flows for large (kilometer sized) domains and time scales while still taking into account the dissipation of energy, which takes place at scales of motion around $\mathcal{O}(1\text{mm})$ and is of vital importance for the dynamics of the mean flow. Our simulations are performed by a parallelized version of the code described by Cuijpers [17]. The data sets used in Section 2.5 are cases of the Barbados Oceanographic and Meteorological EXperiment (BOMEX), which is documented in [87].

To simulate a field of cumulus clouds, which are relatively small clouds located in the ABL, a typical domain size is $6.4 \times 6.4 \times 3.2$km. The simulation keeps track of several variables: three velocity components, temperature, liquid water, and total moisture. These are updated in the simulation by integrating between time steps representing 2 seconds of real time. Using, for example, a grid size of 512^3 and outputting every third simulation time step, the data set produced for simulating 1 hour of real time will be more than 500 GB. Thus, the size of the dataset can be a disadvantage when attempting to investigate specific features of the flow. This is especially true for exploring the evolution of flow characteristics in time since this requires both a suitable cloud in space and knowledge of its history. While a thorough selection procedure can slim the dataset considerably, in many cases it is still insufficient. In those cases, a 3D, interactive environment like VR can be an excellent way to increase the amount of data that can be handled by a user, and thus also enable searches of the dataset for criteria that can be most readily identified by human perception.

2.2.3 Cloud Visualization

Many examples exist of cloud visualization. One early example ([55]) describes techniques for ray tracing volume density fields, resulting in fuzzy, cloud-like images. Later examples are mainly concerned with clouds for application in animations or games, aiming at visual realism using complex lighting models [80, 85], or high performance [42, 104]. In our case, we are mainly interested in the physical accuracy of the cloud simulation, although visual realism may be of some help to benefit from the observational experience of the atmospheric researchers.

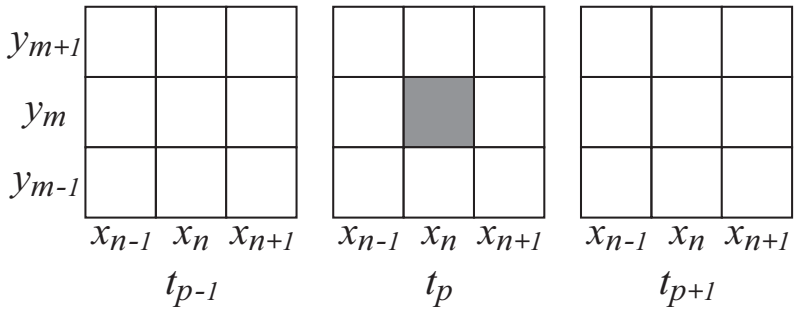


Figure 2.2: This figure depicts the 26 neighboring cells of cell (x_n, y_m, t_p) in a three dimensional, i.e. two spatial dimensions and one temporal dimension, binary array.

2.3 Data Preprocessing

The first phase of our approach is data preprocessing, which prepares the raw data for visualization in VR (Figure 2.1). This phase proceeds in two major stages: feature tracking and isosurface generation. For the purposes of this work, we refer to clouds as features in the data. We incorporate the usual feature detection step into the tracking phase because it is straightforward in our case. In the tracking phase, individual clouds in the data set are detected, identified and tracked through time. We also record data about the clouds during this stage. Once the tracking is complete, we generate isosurfaces representing the cloud shape for later visualization.

The cloud data sets that we worked with are three dimensional volumes for each time step. The volumes are $128 \times 128 \times 80$ or $256 \times 256 \times 160$ grid cells in size. The volumes are periodic in both the x and y directions. This means that clouds may "wrap around" the sides as they move with the wind. For preprocessing, we are only concerned with one of the scalar quantities generated by the LES: the quantity of liquid water in the air, which is centered at each grid cell. A non-zero amount of liquid water indicates there is visible cloud in the grid cell, and we use this for cloud detection.

2.3.1 Cloud Tracking

Tracking clouds can be quite complex. Fortunately, atmospheric scientists are not interested in clouds that go through large-scale merging events. We take advantage of this to greatly simplify the tracking. We label all cloud masses that ever come in contact with each other as a single cloud. During the later visualization process in VR, scientists can quickly identify and disregard any undesirable "clouds" that result from multiple collisions or subdivisions.

To accomplish the feature tracking, we employ a four dimensional variety of the standard connected components labeling algorithm. This type of algorithm is commonly used in computer vision. A general approach to connected component labeling is described in [24]. In

such an algorithm, a binary array, of arbitrary dimension, is examined, and all array cells that have a 1 value are assigned a label based on their neighbors. Specifically, a cell will have the same label as its neighbor cells, and no two distinct sets of cells will have the same label. In this way, neighboring cells with 1 values are "clustered" together. For the four dimensional algorithm, we consider all 80 spatio-temporal neighbors. See Figure 2.2 for a simplified illustration. We construct the four dimensional binary array by considering each successive time step, and we assign a 1 value for each grid cell with a non-zero amount of liquid water and a 0 value otherwise. All cells in the binary array with a 1 value are considered part of a cloud. Two grid cells that are part of clouds, $v1 = (x_1, y_1, z_1, t_1)$ and $v2 = (x_2, y_2, z_2, t_2)$ with t_1 and t_2 representing time steps, are said to be neighbors, and thus part of the same cloud, if:

$$\max(|x_1 - x_2|, |y_1 - y_2|, |z_1 - z_2|, |t_1 - t_2|) \leq 1.$$

The one caveat we must consider is that the x and y axes are periodic, whereas the z and t axes are not.

The 4D connected component labeling is attractive for several reasons. It will work in almost all cases, except for some small and uninteresting clouds, because of the slow cloud development and significant overlap between frames. It gracefully handles merging and splitting of clouds. This has the added advantage that it lumps large groups of merging clouds together, which allows them to be more readily ignored. It is computationally inexpensive, and it makes identifying corresponding features in adjacent time steps trivial. Furthermore, it offers fine-grained tracking, which other approaches like simple bounding box overlap may lack.

While creating the connected components, we also keep track of other data. We record the cloud volume by totaling the number of grid cells each cloud occupies in each time step. Additionally, we keep track of the bounding boxes for each part of a cloud in each time step. In a brief post processing step, these bounding boxes are enlarged for aesthetic reasons, and overlapping boxes are merged. These bounding boxes are used in isosurfacing and later visualization.

2.3.2 Isosurface Creation

Isosurface creation is a pipeline process. First, we prepare the data, and then we generate the initial isosurfaces with the marching cubes algorithm [63]. We refine the resulting triangle meshes with a series of filters, and finally we convert the refined meshes to triangle strips. We must take some care in this process, however. We want to keep the size of the isosurface data as small as possible to ease the burden on Cloud Explorer's interactivity, and we would like the clouds to have a more cloud-like appearance. Additionally, the triangle stripping algorithm we use requires manifold triangle meshes as input.

Creating the initial isosurfaces is fairly straightforward. First, an empty volume is created that is twice the size of the data volume in both the x and y directions. Next, the data for a particular time step is selectively placed into this empty volume. This is necessary for creating "complete," i.e. manifold, isosurfaces of clouds where clouds wrap around the periodic

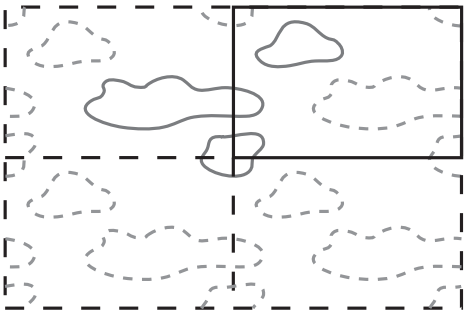


Figure 2.3: If the data for a time step is selectively placed into a larger volume in the fashion shown, each cloud will only appear in the volume once, and that one instance will be a manifold object.

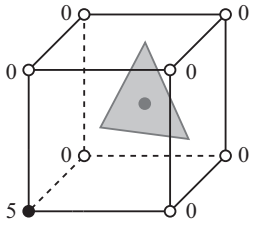


Figure 2.4: The centroid of the triangle lies within a cube where each corner of the cube is a grid cell. The black corner belongs to cloud 5 while the white corners do not belong to any cloud. Therefore, the triangle is part of cloud 5.

boundaries. By considering which cloud each grid cell belongs to and what the bounding boxes of that cloud are, it is possible to place the data for just one complete copy of each cloud into the volume. See Figure 2.3. Marching cubes can then be applied to this volume to generate triangle meshes representing the cloud shape.

After isosurfacing, the triangles are prepared for eventual storage and visualization. The first step is to separate the triangles into sets of meshes representing the individual clouds. By performing a look up into the connected components data for the centroid of each triangle, all the triangles can be separated into appropriate sets. See Figure 2.4. This produces a collection of sets of triangular meshes where each mesh represents a piece of a cloud and each set of meshes represents a single cloud. Next, we perform three operations on the triangle meshes. These are, in order of application, a windowed-sinc filter to smooth the mesh, a decimation pass to reduce the number of triangles in the mesh, and a pass to generate smooth normals for the mesh and improve its appearance. See Figure 2.5 for the results of this.

The last step before storing the meshes is to convert them to triangle strips to reduce the size of the stored data. We use a variation of the stripping algorithm presented by Gopi and Eppstein [30]. Their algorithm produces a single triangle strip if given a manifold triangle mesh as input, but it introduces new mesh vertices to do so. The cloud meshes are manifold,

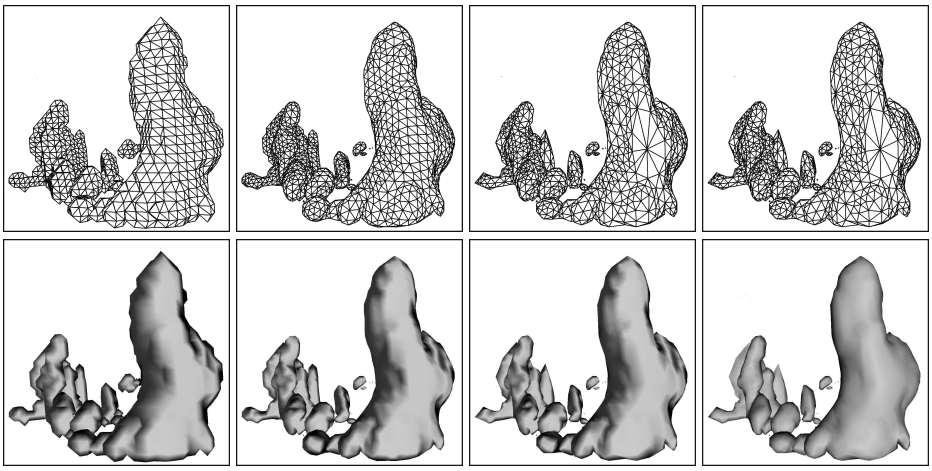


Figure 2.5: The top row illustrates the triangle mesh while the bottom row shows the corresponding shaded model. From left to right: raw output from marching cubes, after applying a windowed-sinc filter, after decimating the mesh, and after generating smooth normals.

but we do not insert new vertices so we end up with multiple triangle strips. It is important to note, though, that the resulting triangle strips are just strips of adjacent triangles, and actually consist of alternating sequences of triangle fans and triangle strips (Figure 2.6). We keep track of when the strips alternate between fans and strips so that we are able to recreate the surface later with consistent winding.

2.4 Virtual Reality Cloud Explorer

The second phase of our approach is interactive visualization in VR. Our VR Cloud Explorer application allows users to interact with evolving cumulus clouds produced by LES. The interface provides tools to allow users to focus on potentially interesting clouds. By examining these clouds more closely within Cloud Explorer, they can pick out certain ones to study further, e.g. for life-cycle studies. The application was built on top of OpenGL Performer and the RWB library, which is a custom VR library described in [58]. In the remainder of this section, we present a sample scenario for using Cloud Explorer, and then we describe the components and relevant interactions in more detail.

2.4.1 Interaction Scenario

The user begins her interactions with Cloud Explorer once the visualization data is loaded and the application is running. She first watches the cloud field over the entire simulation run (Figure 2.7a). This gives her a feel for the clouds that are present, and she can quickly tell if

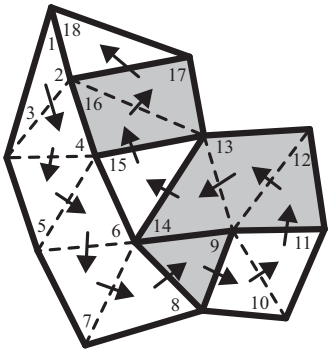


Figure 2.6: A strip generated by Gopi and Eppstein's algorithm [30] begins as a triangle strip, and then alternates between triangle fans and triangle strips. Triangle fans are indicated with gray triangles.

anything went wrong with the simulation run. If the clouds seem reasonable, she proceeds to hide all clouds not going through the entire life-cycle as they are not of interest (Figure 2.7b). She continues to browse through time to spot potentially interesting clouds among those remaining visible. She selects one of those clouds to examine the plot of its volume over time (Figure 2.7c, Section 2.4.2). This gives her an overview of how the cloud evolves. She is looking for clouds with bellshaped volume plots, which indicates that they go through the proper life-cycle stages. If it is acceptable, she limits the time playback to the cloud's life-cycle so she can focus on the time steps from just before the cloud comes into existence to just after it dies out (Figure 2.7d). Next, she hides all clouds except the cloud she is examining (Figure 2.7e). She then lets the application play through the cloud's life-cycle while she orients the cloud field and examines it from various perspectives (Figure 2.7f). If she is satisfied that the cloud is worthy of further study, she notes the number of the cloud down. Once she is done looking at a cloud, she shows all the clouds going through the complete life-cycle again, and she examines any other potentially interesting clouds in a similar manner. When she is finished with the application, she uses the numbers of the clouds she has written down, along with the connected components data, to extract just the portions of the data set that contain the interesting clouds. She can then perform various postprocessing steps on this data to further analyze the clouds, and she can also generate more detailed simulation results focusing on these particular clouds.

2.4.2 Application Components and Interaction

Cloud Explorer provides users with three interaction techniques: ray casting, direct manipulation, and world-in-miniature, or WIM. Ray casting and direct manipulation are used interchangeably for object selection and manipulation. The default interaction is ray casting, but if the tip of the interaction device, a stylus in our case, is within an object, then we switch to direct manipulation. Currently, we only use WIM as a method for orienting the cloud field.

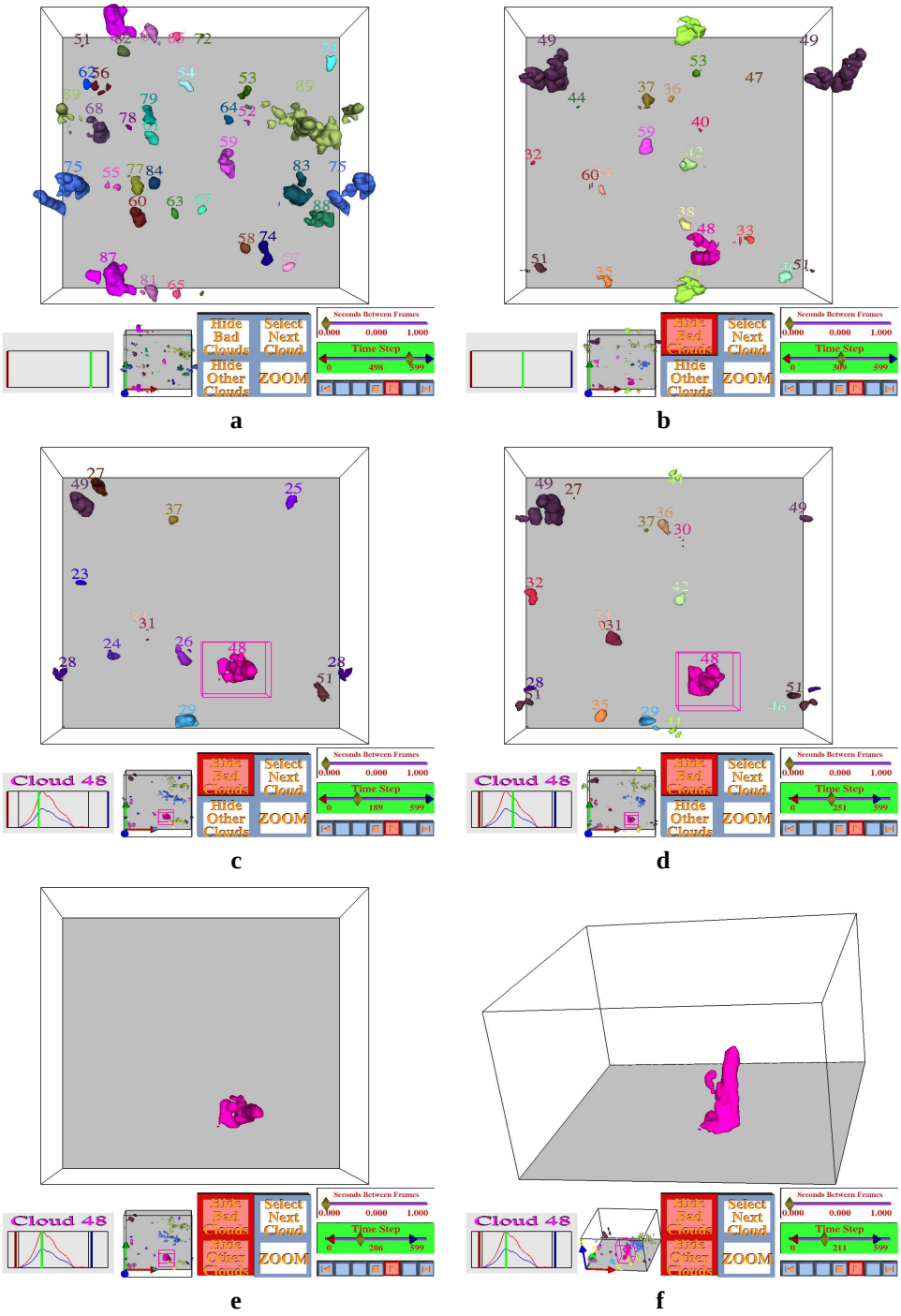


Figure 2.7: Cloud Explorer in various stages of use. See also Figure 2.8 and Section 2.4.2 for more details.

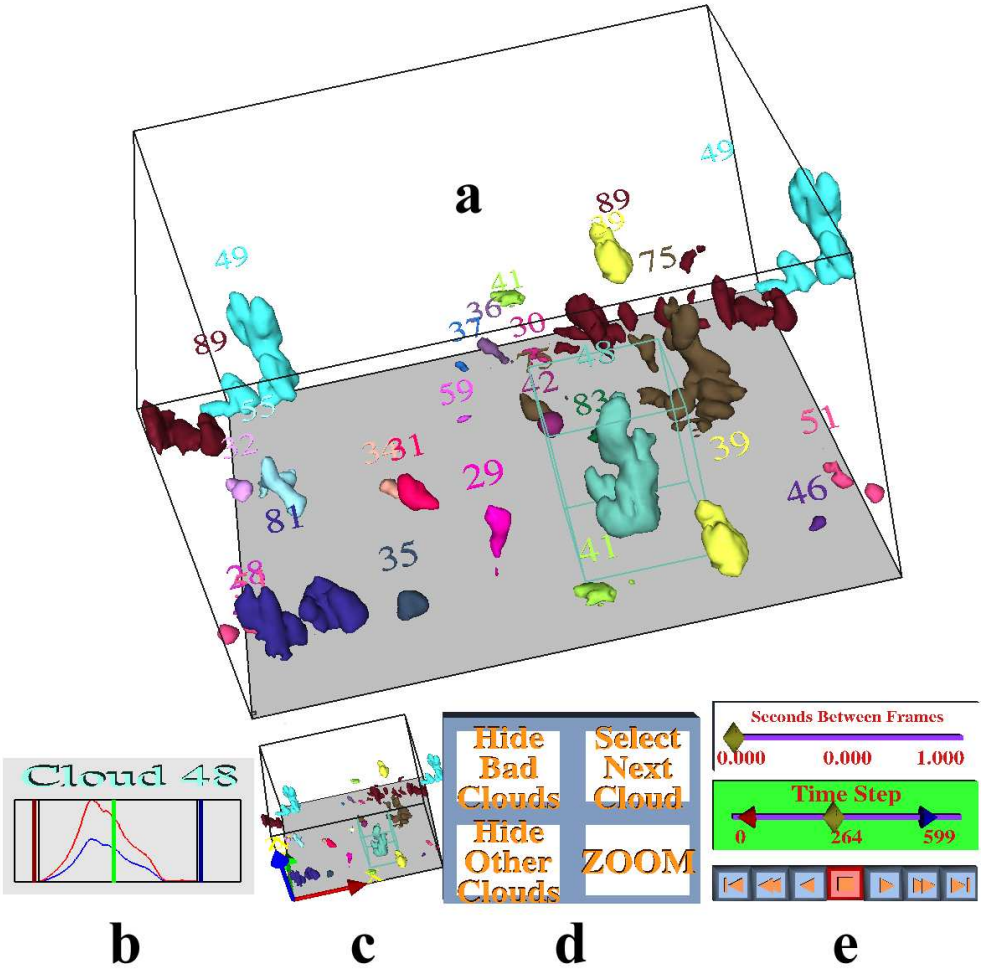


Figure 2.8: Cloud Explorer components: a) cloud field, b) volume graph, c) WIM, d) buttons, and e) time control panel.

The user's primary view on the data is the cloud field. This is the large box (Figure 2.8a), which displays the cloud geometries. In the default mode, each cloud appears in the cloud field in a distinct color. Above each cloud, a unique, identifying number is displayed for collaboration between multiple users and use in post processing. If the user has selected a particular cloud in the cloud field, that cloud will be displayed within its bounding box. The user also has the ability to hide certain clouds in the cloud field with the provided buttons (Figure 2.8d). In particular, he may choose to hide all clouds which do not go through the entire life-cycle or all clouds except the currently selected cloud. If he chooses to only display the selected cloud, then that cloud's bounding box and identifying number will be hidden.

The user's secondary view on the data is the volume graph. This panel displays two plots of the currently selected cloud's volume over time (Figure 2.8b). One is relative to the cloud's maximum volume, and the other is relative to the largest cloud's maximum volume. The plots give users an estimate of the currently selected cloud's relative size and behavior over time. When the user selects a new cloud, the plots are updated, and the identifying number of the selected cloud is displayed above the plots. To assist the user, the volume graph has several indicators. One indicates where along the time axis the current time step lies. Two others mark the cloud's birth and death. The final two indicate which time span the playback is currently limited to.

For additional perspective and control, the user is provided with the world-in-miniature, or WIM (Figure 2.8c). The WIM and the cloud field maintain the same orientation, making the WIM a convenient tool for reorienting the cloud field. However, the WIM also plays a minor informational role since it always displays all the clouds in the current time step and indicates the selected cloud with its bounding box. In addition, the positive directions in each of the x, y, and z directions are indicated in the WIM to serve as a reference point should the user become disoriented. If the user finds the WIM unhelpful, he is able to remove it from view.

The time control panel gives the user control over the time dimension. It consists of two scroll bars and several video player-like buttons (Figure 2.8e). The top scroll bar controls the frequency at which new time steps are displayed, and the buttons control the time step sequence. The lower scroll bar gives the user more interactive control over which time steps are displayed. He can move the slider around to rapidly browse through time or jump to a time of interest. He can also limit the range of time steps that the playback cycles through. Whenever the slider or the limits are moved on this scroll bar, the corresponding indicators on the volume graph are also updated. In this way, the user can use a combination of the time control panel, the volume graph, and the cloud field to explore the evolution of the clouds over time.

Cloud Explorer provides a set of button widgets for extra functionality (Figure 2.8d). The buttons provide tools to change the current mode of operation for Cloud Explorer rather than methods to directly interact with the data. With these, he is able to zoom in or out on the cloud field, cycle through the clouds in the data set, and determine which clouds are visible in the cloud field.

	Grid Size×Time Steps	q_l Size	Total Size
A	$(128 \times 128 \times 80) \times 600$	1.46 GB	8.76 GB
B	$(128 \times 128 \times 80) \times 1184$	2.89 GB	17.34 GB
C	$(256 \times 256 \times 160) \times 2169$	42.36 GB	254.16 GB

Table 2.1: Three data sets consisting of three dimensional grids for each time step for each of six variables. For preprocessing, we are only interested the variable q_l , which indicates liquid water. 600 time steps represent one hour of real time, with one time step every 6 seconds.

	CPU Time	Peak Mem. Usage	Avg. Mem. Usage
A	9m 44s	203 MB	156 MB
B	31m 24s	278 MB	223 MB
C	11h 38m 22s	2557 MB	1836 MB
A*	2m 49s	163 MB	117 MB
B*	6m 59s	238 MB	189 MB
C*	2h 04m 58s	1600 MB	1011 MB

Table 2.2: Processing time and memory usage for the three data sets. We used a dual 3.60 GHz Pentium Xeon Linux machine with hyperthreading and 3 GB of RAM, and the data was stored on a RAID0 system with read speeds up to 320 MB/s. 4 time steps were processed simultaneously. In those cases marked with a star, only clouds going through the entire life-cycle were fully processed.

2.5 Results

2.5.1 Preprocessing

We implemented the preprocessing phase as a multi-threaded, stand-alone application. The application is crossplatform, running on both Linux and Windows. We used three data sets (described in Table 2.1) to measure the application's performance in three key areas: overall processing time, memory usage, and data reduction.

For each data set, we produced two sets of output. In all cases, we first identified and

	Number of Triangles	Output Size	Data Compression Ration (vs. q_l)	
A	11,731,822	165 MB	54 : 1	(9 : 1)
B	33,468,700	468 MB	36 : 1	(6 : 1)
C	517,110,152	6.9 GB	36 : 1	(6 : 1)
A*	2,493,716	39 MB	228 : 1	(38 : 1)
B*	6,456,066	99 MB	180 : 1	(30 : 1)
C*	46,824,994	679 MB	384 : 1	(64 : 1)

Table 2.3: The number of triangles, total output data size, and the resulting compression ratios when compared with the total input data set size and just the q_l size. In those cases marked with a star, only clouds going through the entire lifecycle were fully processed.

tracked all clouds in the data set via the connected component labeling algorithm. We then used the data collected in this stage to filter some clouds out of the subsequent processing. In both sets, we eliminated all clouds having an average volume of less than ten grid cells per time step. In the second set of output, we also eliminated all clouds that did not go through the full life-cycle, i.e. those clouds present in the first or last time step. Table 2.2 relates the processing time and memory usage required for the preprocessing. Table 2.3 gives a sense of the content and size of the resulting data, as well as the compression ratios.

Eliminating clouds that did not go through the entire lifecycle yielded significantly smaller output, which is also reflected in the shorter processing time required. This is because each data set has one or two "super clouds", which do not go through the entire life-cycle and are present throughout most or all of the time steps. These clouds consist of several clouds that continuously merge and split. Removing them means fewer triangles are generated in isosurfaces, which in turn means less mesh refinement is necessary.

2.5.2 Cloud Explorer

To test the effectiveness of Cloud Explorer, we performed a simple pilot study on our Responsive Workbench (Figure 2.9) using data set C from Table 2.1. Currently, Cloud Explorer loads all of the visualization data into memory for performance reasons. This did not fit into memory so we, instead, confirmed the suitability of the data set by examining a handful of time steps of the full output, and then we worked with the output containing only clouds going through the entire life-cycle. Aside from these minor differences, our VR session proceeded similarly to Section 2.4.1 During the session, we experienced frame rates between 20 and 30 FPS.

Using Cloud Explorer, we identified seven clouds of particular interest. We then extracted the relevant parts of the data set. Using this per cloud data, we generated mass flux plots indicating the motion trends within the clouds over their life-cycles. See Figure 2.10 for one of the plots. The resulting plots were quite consistent, and they demonstrated interesting profiles when the clouds were decaying. We feel that the consistency in the graphs is a good preliminary indicator that our approach is successful.

2.6 Conclusions and Future Work

Cumulus cloud simulations generate very large, timevarying data sets of up to hundreds of gigabytes.We showed how feature tracking and VR can help to deal with such large data sets and how it can help scientists to interactively select good clouds for further study, e.g. life-cycle studies.

The interactive selection is performed by first preprocessing the data and then interactively visualizing the results. Through the efficient representation of the clouds as isosurfaces, sufficient data reduction is achieved to allow the interactive VR visualization as done in our Cloud Explorer application. VR has proved to be a very valuable tool for interactive exploration, allowing theoretical and observational considerations to be combined.

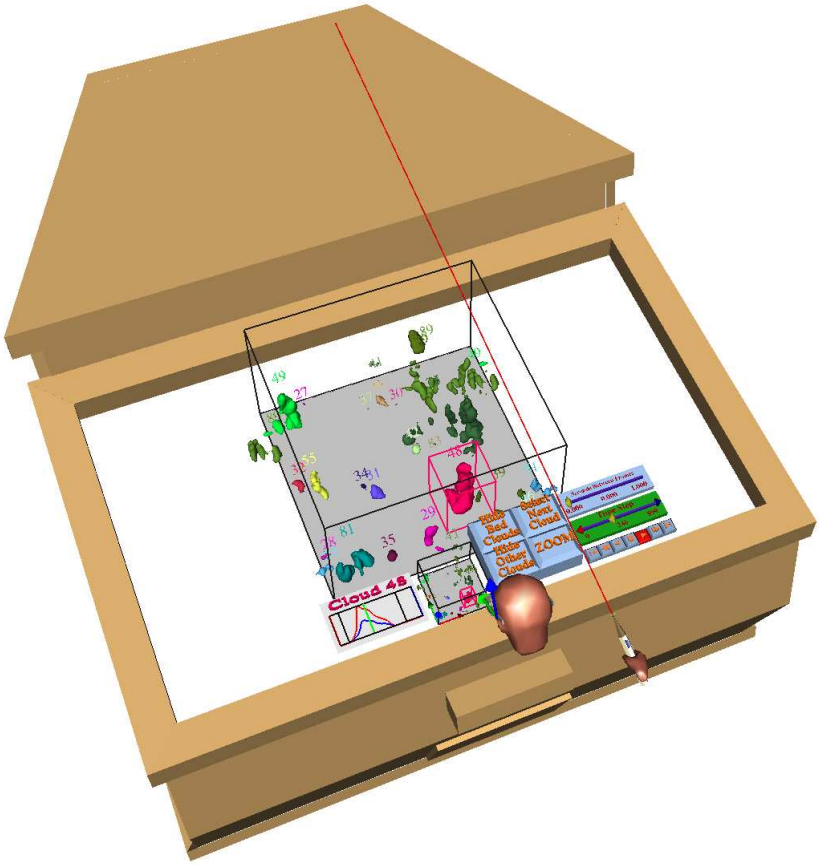


Figure 2.9: Playback of a recorded session on the Responsive Workbench.

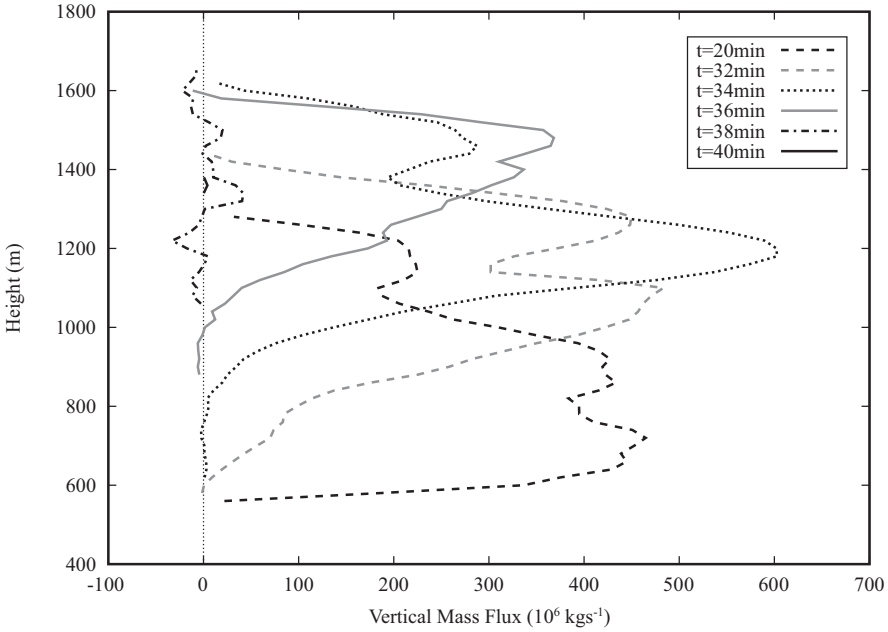


Figure 2.10: A mass flux plot for a cloud selected with Cloud Explorer. Positve flux (right of the zero line) indicates the cloud mass is moving upwards at that altitude. The cloud begins to decay at approximately $t = 32$ minutes. After this point, the cloud is no longer being fed by a thermal, and it drifts upwards and dies out. Interestingly, the profile retains most of its shape until the final minutes.

Having made their selections, the scientists extract the relevant parts of the data. They can then perform more detailed analyses of, for example, flow patterns and heat exchange at every stage of a cloud's life-cycle. This combination of simulation and VR visualization techniques will allow these types of studies to be done for the very first time.

This work is part of a larger project with many possible future directions. We hope to further formalize and automate the cloud selection process. We would also like to incorporate documentation facilities and new visualization techniques into Cloud Explorer. The aim of these steps is to further enable the atmospheric scientists to make use of their observational skills, though, rather than to eliminate them from the pipeline. Additionally, we will need to incorporate other large data handling facilities, such as out-of-core and multi-resolution representations, to cope with the full magnitude of the data sets LES can produce.

Acknowledgements

We would like to thank the Netherlands Organisation for Scientific Research (NWO) for providing project funding and for providing computer time at the National Computer Facility (NCF) at SARA for sumulations.

CHAPTER 3

Quantitative Data Analysis in Virtual Environments through Reprocessing

This chapter was originally published in the proceedings of the 2007 conference for the Advanced School for Computing and Imaging [35]. The conference paper was an expanded version of a peer-reviewed short paper [34] originally presented at the ACM Symposium on Virtual Reality Software and Technology in 2006.

Abstract

This paper presents an approach to help speed up and unify the exploration and analysis of time-dependent, volumetric data sets by easily incorporating new qualitative and quantitative information into an exploratory virtual environment (VE). The new information is incorporated through one or more expedited offline "reprocessing" steps, which compute properties of objects extracted from the data. These objects and their properties are displayed in the exploratory VE. A case study involving atmospheric data is presented to demonstrate the utility of the method.

3.1 Introduction

Today, many researchers make use of complex simulations to analyze various phenomena. These simulations often produce time-dependent data sets, which are growing increasingly large as more sophisticated simulation techniques and faster computing technology emerge.

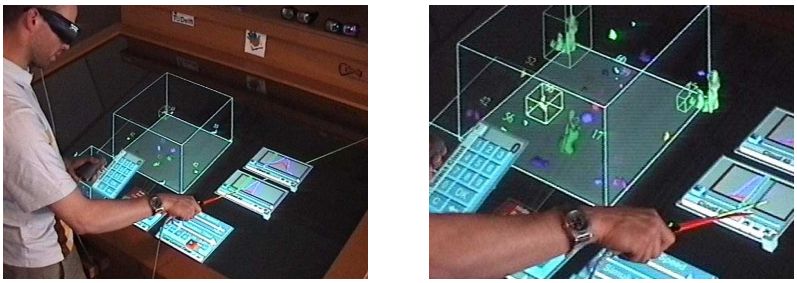


Figure 3.1: Left: The VE in use. Right: A closer view of the VE.

However, the time required to extract meaningful results from these simulations is also increasing with the complexity and size of the simulation output. Virtual reality (VR) can help scientists make sense out of such data sets, but most VR software for data visualization is designed for specific problems and often lacks integration into the larger data analysis process.

Due to the lack of adequate exploratory environments for data sets produced by such simulations, researchers must often spend a significant amount of time moving between several existing tools or devising new special purpose tools in order to analyze and understand their data. As the scientists investigate their data, they often come up with yet more properties they would like to examine. This can lead to using and developing even more tools, which then further slows the process down. By providing these scientists with a more unified process consisting of a set of flexible tools that are coupled with a suitable environment for exploring these large, time-dependent data sets, it is hoped that the ratio of time the scientists spend on exploration and analysis versus tool selection and development can be significantly increased.

This paper proposes an approach to help speed up and unify the analysis and exploration of large, time-dependent data sets. This is accomplished through the combination of an exploratory virtual environment (VE) and a data processing tool. The VE provides standard manipulation and data probing tools for exploring the 3D, multivariate data, and it incorporates quantitative and qualitative information generated by the data processing tool (Figure 3.1). The data processing tool generates this information by using a simple expression parsing grammar to perform a variety of computations on the data. Furthermore, the same data processing tool can be reused during an arbitrary number of "reprocessing" steps to generate new output data for inclusion in the VE. Currently this takes place offline after examining the data in the VE, but we plan to incorporate it as an interactive, real-time component of the VE in the future.

The work presented in this paper stems from our cumulus cloud research project. As the work evolved, we closely followed the analysis cycle described by Upson et al. [101] (Figure 3.2). Later in the paper, we describe the iterations we went through, and we relate how the combination of the exploratory VE and the data processing tool we have developed support the steps we took.

The remainder of this paper is organized as follows. We discuss related work in the next

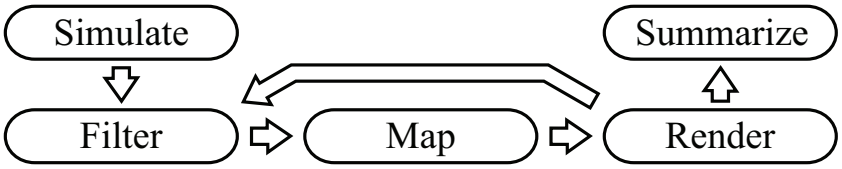


Figure 3.2: The analysis cycle as described by Upson et al. in [101]. Simulation data is filtered to produce data for visualization, which is then mapped to geometric primitives. These primitives are rendered and studied yielding insight, which can be used to initiate a new round of filtering.

secion. In Section 3.3, we give a general overview of our approach. In Section 3.4, we present our software system and the salient points in its evolution from our previously developed VE exploration tool for life-cycle studies [36]. In Section 3.5, we describe how this work relates to the atmospheric research in our cooperative project. We conclude the paper in Section 3.6, and we discuss our plans for creating a fully interactive system from the current one.

3.2 Background and Related Work

In their presentation of the AVS software, Upson et al. [101] describe the scientific data analysis process as largely a filter, map, render loop (Figure 3.2), which transforms the simulation data into meaningful images that researchers can analyze. The loop gives scientists insight into their data, which also helps drive further iterations of the loop. Researchers may also generate plots, movies, or other visual representations for communicating the insight they have gained. This model, however, focuses mainly on the visualization software itself, and it does not really consider the supplemental tasks, which are still an important part of the process.

Springmeyer et al. [93] further characterized the scientific data analysis process. He breaks the process down into four components: analysing representations of the data, performing calculations, maneuvering through and in the data, and expressing the ideas gleaned from the process. These components each encompass a variety of tasks that researchers perform during the analysis process. Based on these tasks, he suggests five functional requirements for software that is to support scientific data analysis: allow interactive, quantitative exploration; assist in maintaining records of sessions; link materials from different stages of a study; simplify navigation requirements; and provide support for culling large data sets. We have attempted to incorporate most of these into the software we have developed.

A variety of exploratory data visualization environments exist. Popular examples using the network data flow architecture are AVS [101], OpenDX (originally IBM Visual Data Explorer) [64] and IRIS Explorer [27]. These environments employ a modular, visual programming approach to allow users to piece together a visualization pipeline that generates one or more views on their data. Another common approach is to build a custom application on top of libraries such as the Visualization ToolKit (VTK) [86]. However, these environ-

ments tend to be more geared towards image or film generation rather than the exploration process itself. Interaction with the data can be clumsy due to the limitations of the mouse and keyboard interface, and it is often difficult to effectively integrate multiple views on the data.

Much work has also been done on data visualization within VR. See [16, 19] for a variety of examples. Still, as Johnson points out in [52], effectively visualizing time-varying, multivariate data remains a challenging and open problem. On deskop systems, and to a lesser extent in VR, this is further complicated by a lack of effective interaction tools. Another weakness Johnson points out in current systems is their lack of integration with the overall problem solving environment. With our method, we are attempting to address these challenges by bringing together data processing and the exploratory VE and incorporating more information into the VE.

In a process of 'selective visualization', Van Walsum [103] used expression parsing techniques to compute derived data from an original data field, to be used in defining Boolean selection expressions for selecting important parts of the data. Facilities for performing matrix/vector operations, calculating gradients, and determination of descriptive statistics were included in this system. However, no facilities for direct exploration of the derived data were available. This issue was addressed by an approach called 'linked derived spaces' [46], in which derived quantities could be calculated and also visualized in user-defined 2D coordinate spaces that were directly linked to the original data. In our approach, a similar expression parsing facility is used to calculate new data, but in our case the new data are imported in the VE, so the full range of visualization and exploration techniques is available to the user.

3.3 Method Overview

With our previous Cloud Explorer application [36], we supported a research pipeline illustrated in Figure 3.3a. This linear approach helps cull the data set by allowing the scientists to study only those portions of the data containing interesting objects, but the narrow focus of the preprocessing and the VE prevent the scientists from gaining further insight into the data. Instead, they must continue to rely on their traditional research software and methods.

Our newly proposed pipeline incorporates five steps:
Step 1: Simulation
Step 2: Data (re)processing
Step 3: Data visualization
Step 4: Repeat Steps 2 and 3 as needed
Step 5: Generate quantitative results.
The approach is reflected in Figure 3.3b. Aside from the generalization away from clouds, the most important change is the data "reprocessing" cycle introduced into the process with the optional jump from Step 4 to Step 2. With this, the new pipeline closely parallels the analysis cycle described by Upson et al. [101] seen in Figure 3.2.

The crux of the initial data processing phase is identifying important objects in the data. The lifespans and bounding boxes of these objects are determined and saved to disk. At the researchers' request, new copies of the raw data are created containing only those portions of

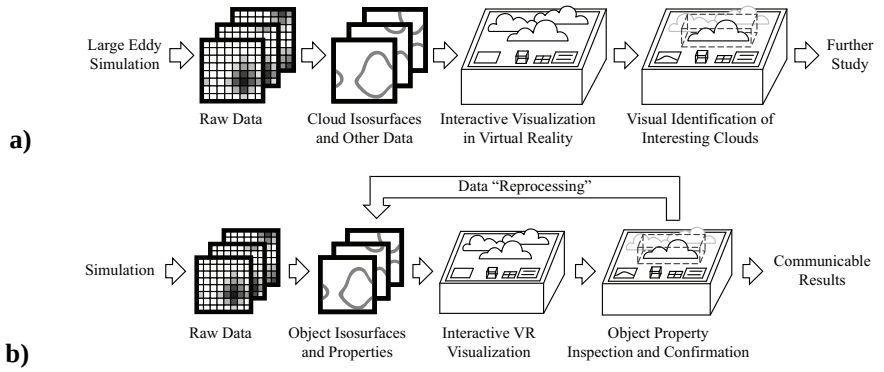


Figure 3.3: **a)** This pipeline represents the pipeline supported by our original Cloud Explorer application [36]. **b)** This pipeline represents our newly proposed pipeline, which incorporates a "reprocessing" cycle.

the data set contained in each object's bounding box. This can greatly speed up later processing of the data as, in our experience, objects often only occupy a small percentage of the total data space and scientists are most interested in objects and their immediate surroundings.

The data visualization is the most important step in the method. The visualization process gives the researchers insight into the data. This insight can inspire new lines of investigation into the behavior and properties of the objects of interest. By integrating quantitative information about the objects into the environment, researchers can correlate object properties with their behavior in time, and they can test the validity of their hypotheses by comparing several objects during a session in the VE.

The data reprocessing step is the key new step in the method. This introduces a cycle into the process, and it helps researchers spend more time in the VE, while making the time spent in the VE more productive. By making the data processing tool flexible enough, it can be used to generate new quantitative data based on the objects in the data and expressions supplied by the scientists. Due to the initial data processing step, this reprocessing can proceed quite quickly. The generated information can then be directly loaded into the virtual environment.

The last step of the process is the generation of communicable results, in the form of statistical data, plots, numbers, animations, or other representations. The researchers must transform what they have learned from the VE into a form, which can effectively convey the insight gained to other scientists or a wider audience. Publication or public demonstration quality visual and numerical results often require precision, flexibility, and functionality beyond that which is sufficient for the exploration process. Hence, this is currently left as a post-processing step for the researchers to tackle with more specialized tools. In the future, however, we would like to incorporate more of this functionality into the VE.

```
<input>
    <path>/data/simulation001</path>
    <variable name="ql" file="ql.001" />
    <time steps="2000" />
    <grid x="256" y="256" z="160" />
</input>
```

Figure 3.4: This example snippet of the input section from a processing specification file describes the grid size, the number of time steps, and the name and location on disk of one simulation variable.

3.4 Software System

The software described in this paper can be seen as an extension and generalization of our initial work on the Cloud Explorer application [36], and it is an intermediate step along the way towards our goal of a fully interactive virtual environment for exploring and analysing large, time-dependent, volumetric data sets. The expansion and generalization of Cloud Explorer involved three major steps: generalization of the preprocessing, extension of the preprocessing, and expansion of the Cloud Explorer environment.

3.4.1 Preprocessing Generalization

The Cloud Explorer software was specially designed for our atmospheric research project. The result was successful, but it was not really applicable to other data sets because it placed some restrictions on the format and nature of the data.

In the new system we have developed, most of these restrictions have been eased or eliminated. The data can now be periodic in any direction, or it can be completely aperiodic. The objects in the data are now identified by a user specified threshold, and they no longer need to have manifold surfaces. The data must still be stored as a sequence of z major grids, but allowances are now made for file and time step headers and trailers.

The data processing program loads and interprets a processing specification file, which is in extensible markup language (XML) [10] format. The specification file is broken into three sections, which describe the input data (the simulation data), the desired output data, and any necessary mappings from the input data to the output data. Figure 3.4 illustrates an example portion of the input section.

3.4.2 Preprocessing Extension

Our primary goal when developing this software was to be able to incorporate more quantitative information in the VE. However, it is difficult to know in advance what type of quantitative information will be interesting to the research scientists. Therefore, the data processing needed to be flexible enough to generate arbitrary quantitative data from the data set. To make the software practical to use and to facilitate exploration, we required that it be possible

to later generate new quantitative data from the data set. We also wanted it to be able to work with multivariate data sets to take advantage of the full range of information generated by the simulations.

The first and most basic extensions to the data processing application are the production of subvolumes extracted from the data and downsampled versions of the data. The downsampled versions of the data are for use with slicing tools, and the subvolumes, which each contain a object and some of its surroundings, are used as a means for speeding up data calculations in two ways. First, the size of all the subvolumes for a particular time step is often significantly smaller than the size of the total grid, which means that the subvolumes can be read in from disk much more quickly. Secondly, the researchers are often interested in the properties of the objects themselves, or just around the objects, and therefore only necessitates performing calculations on the subvolumes for each object instead of the entire volume for a given time step.

To support calculations based on the data set, we have implemented a small domain specific language (DSL). See [23] for an overview of DSLs. This language supports simple mathematical expressions involving numbers, variables, and functions. Variables can be either scalar or vector values or multidimensional matrices. A set of functions, in addition to basic mathematical operations, has been included in the language to support various operations. These include summing and averaging data across a volume or a single dimension; basic image processing functions like clustering, thresholding, and dilation / erosion operations; detecting maxima, minima and extents; and extracting a subvolume. Operations like gradients are not currently supported, but could be easily incorporated in future versions.

We then coupled this DSL with the generalized data processing application. The user writes expressions, which describe how to convert the input data into a meaningful quantitative result. These are included in processing specification file. The user specifies which input variables are required for the expressions, how they should be prepared, e.g. limit the calculation to the object itself or also include the surrounding volume, and what output should be saved from the evaluation of the expressions. The data processing application then evaluates the expressions on the relevant (appropriately prepared) extract grids for each object in each time step and saves the results to disk.

Currently, the data processing program can generate two types of quantitative data. The first is simply one scalar value for each object for each time step, which allows traditional plots of object properties as a function of time. The second is a vector value for each object for each time step. The intention with the vector values is to generate a scalar value for each voxel plane along a coordinate axis in the subvolumes (Figure 3.5). Taking the example of the *z* axis, an image can be generated where the pixel coordinates represent time and vertical height in the (sub)volume and the pixel color or intensity represents the value for that voxel plane at that point in time. Figure 3.8 illustrates such *z* vector images.

Consider the following set of expressions follows:

bin_grid = **bin**(*grid*)
volume = **sum**(*bin_grid*)
ql_vec = **sumdim**(**sumdim**(*grid*, 0), 0) /
sumdim(**sumdim**(*bin_grid*, 0), 0)

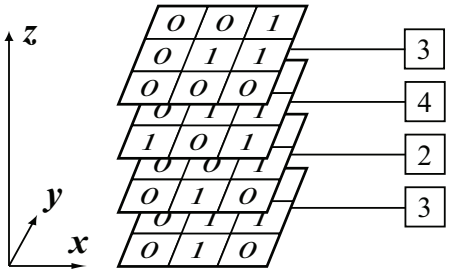


Figure 3.5: An illustration of a vector value. Here, each value in the vector represents the sum of all values in the corresponding *x-y* voxel plane.

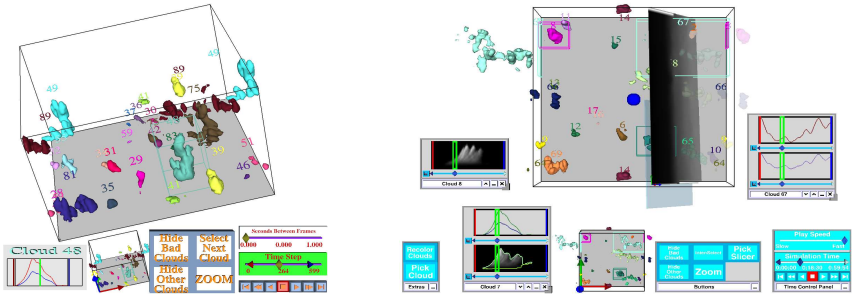


Figure 3.6: Left: The old Cloud Explorer interface. Right: The new application interface. New additions include the use of windows, the slicing plane, and the new graph window.

Here, *grid* is the input subvolume, which has had all voxels not belonging to the object in question set to 0. The **bin** function converts all non-zero voxels to a 1, and the **sum** function adds up the values of all voxels in the grid. Thus, *volume* is set equal to the object volume, in terms of voxel units. The **sumdim** function reduces the dimensionality of the data by summing all values along the specified dimension. In this case, the effect is to set *ql_vec* equal to the average liquid water value at each height in the subvolume. See Figure 3.8 for an example of the result.

The final extension was the inclusion of the "reprocessing" ability. This allows the processing description file to be updated with new sets of expressions. If the data processing program is then run again, any new output results are added to the existing output without having to recompute all prior calculations. This lets the scientists quickly see new quantitative information in the VE.

3.4.3 Cloud Explorer Expansion

The last part of the generalization and expansion process was to update the Cloud Explorer application itself. For this we updated the user interface, included new data probing tools, and

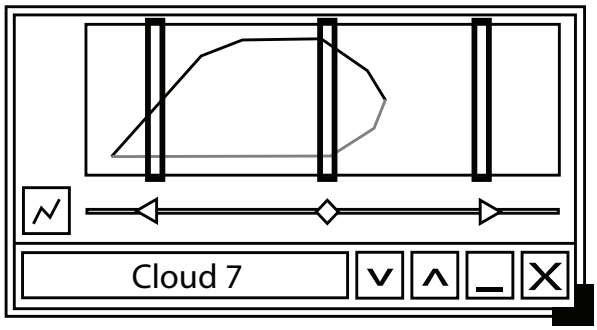


Figure 3.7: An illustration of the graph window.

we expanded the capabilities for displaying graphs and 2D images for selected objects. We also incorporated the IntenSelect interaction metaphor into the environment to make selection and manipulation tasks easier, especially with small widgets. See [37] for a description of the interface widgets and the IntenSelect interaction metaphor. Figure 3.6 illustrates the difference between the old interface and the new interface.

One revision worth describing in detail here is the new graph window. For each selected cloud, up to five, a graph window is displayed with information about that cloud. See Figure 3.7 for an illustration of the window and Figures 3.6 and 3.8b for examples of the actual windows. The window has a caption, displaying the number of the cloud it is showing data for. This can also be used to move the window around. Next to the caption are either one or two buttons, denoted by up or down wedges, used for adding or removing plot display areas to the window. Next to these buttons are buttons for putting the window into windowshade mode and hiding the window. In the lower right corner is a sizing widget, which can be used to resize the window. The main area of the window is divided into one or more plot display areas. For each plot display area, there is the actual display area, which displays one or more user selected plots. This allows users to overlay various plots to check for correlations or other interesting properties. Below the display area is a scroll bar representing time. This includes sliders to adjust the time limits for playback as well as the currently visible time step. Rectangles over the display area highlight those sections of the plots represented by each of the three sliders. To the left of the scroll bar is a button, which opens the plot selection window. This window allows the user to choose which plots to display in each plot area.

With the new interface, scientists can spend more time using the VE. They are able to view quantitative information about several clouds at one time. This helps them compare the properties of objects in the data and ascertain what is “normal”. With the addition of the slicing plane, they can also check the behavior of simulation variables in and around the objects. This can be used to confirm hypotheses or develop new lines of inquiry. If the data processing application is run again to add new quantitative data, then this will be incorporated into the VE the next time it is run.

3.5 Case Study

In this section, we present as a case study the evolution of our research into shallow cumulus cloud life-cycles. We relate how the actual steps taken are supported by our method and the software system we have developed.

3.5.1 Overview

Our research involves the use of a Large-Eddy Simulation (LES) of the Atmospheric Boundary Layer (ABL) containing shallow cumulus clouds. LES is a simulation method where the ruling Navier-Stokes equations are solved up to a certain scale, while the influence of smaller scale (turbulent) motion is approximatied via a statistical model. This way a field of clouds in a $6.4km \times 6.4km \times 3.2km$ volume can be resolved in time with a resolution of $256 \times 256 \times 160$ gridpoints and a timestep of $2s$ without the need to resolve eddies smaller than the grid size ($20m$ down to the smallest turbulent scales of $\mathcal{O}(1mm)$). The simulations are performed with a parallelized version of the code described by [17]; the specific simulated case is based on the Barbados Oceanographic and Meteorological EXperiment (BOMEX), see [87].

During the simulation run, every third time step (every $6s$ of simulation time) is written to disk. Each time step consists of temperature, the amount of water (both in gaseous and in liquid phase), buoyancy, and 3 velocity components for each grid point in the domain. Each value is recorded as an unsigned, 16 bit integer yielding data sizes of 20 MB per time step per variable. For one hour of simulation time, this results in about 12 GB of data per variable. The average life-cycle of a cloud lasts 30 minutes.

3.5.2 Data Preprocessing and Observation

The first step in the research process was to identify interesting clouds for further study, which was the focus of our previous work. See [36] and [48] for a detailed overview. In short, the cloud isosurfaces were extracted from the data, along with volume information and their bounding boxes. This information was interactively visualized in the Cloud Explorer application. The VR environment enabled the visual identification of 40 interesting clouds for further study. At first sight, it may be unclear as to why VR is useful in this task, but the utility stems from the rather qualitative criteria defining interesting clouds and the ability of human perception to single them out. See Figure 3.1 for an example of the VE in use.

Since our new software is based on the extension and generalization of our previous software, this phase of the process continues to be supported, and it does not differ substantially from its previous form.

3.5.3 Study and Exploration

The next step after identifying interesting clouds was to start studying them. The study and exploration phase proceeded iteratively, with each new analysis inspiring the next one.

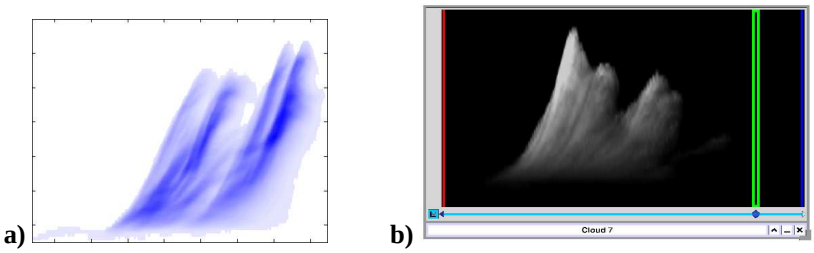


Figure 3.8: a) Horizontally integrated liquid water vs. height and time. This plot was generated via a special tool. Darker areas indicate a higher concentration of liquid water. Each point represents the average amount of liquid water at a given height in the cloud at a particular time step. **b)** An equivalent plot generated by our new software and displayed within the VE.

The examination of the individual clouds began by looking at simple properties, such as how the base and top of the cloud moved up and down in time. The results of this were interesting, but these just served as indicators of interesting behavior. They did not yield information about the cause of the behavior.

To try to understand the cause, we examined the amount of liquid water in the clouds because of its influential role in cloud dynamics. The first analysis step here was to plot the horizontally integrated amount of liquid water as a function of height and time. Surprisingly, the liquid water seemed to be concentrated in pulses that rose with time (Figure 3.8).

Continuing the investigation into these pulses, we repeated the analysis again, but with buoyancy instead of liquid water. The same behavior was seen in these plots. By then examining the total water in and around the clouds, we discovered a correlation between an increase in the amount of water underneath the clouds and the onset of the pulses.

At each step of the process, we performed the same analyses on not just one cloud, but several clouds. This enabled us to check whether the observed phenomena were generic or specific to a particular cloud. It turned out that the observations were consistent across the clouds, suggesting that the pulses are a defining feature of the clouds.

This iterative advancement of ideas is at the core of our proposed system. The analytical data generated at each step is something that our new software supports. The height of the cloud base and cloud top are scalar values, represented by the maximum and minimum extent of the object in a given time step, and these scalar values can be calculated and stored via the data processing. The horizontal integration of liquid water and buoyancy can be achieved via the vector calculatinos supported by the data processing. Figure 3.8, illustrates the original liquid water integration plots and the equivalent plots generated by the data processing as seen from within the VE. The slicing plane can be used to examine the behavior of total water in the cloud base. By looking at the integration plot of liquid water or buoyancy and using the slicing plane simultaneously, the build up of liquid water preceding the appearance of the pulses in the clouds can be verified.

3.6 Conclusions and Future Work

In this paper we have presented an approach geared towards helping scientists working with large, time-dependent data sets reach results more swiftly. The approach depends on the idea of "reprocessing" the data, and incorporating new quantitative information into a scientific visualization virtual environment. This introduces a cycle into the data analysis pipeline, which reduces the amount of time scientists must spend selecting or developing their own analysis tools.

We have described the expansion and extension of our previous software to meet the objectives of the proposed method. We restructured the data processing to handle a wider variety of data sets, and we incorporated a small domain specific language of mathematical expressions to generate quantitative data from the data sets. Lastly, we updated the VE to incorporate this new quantitative data, among other improvements. The inclusion of the quantitative data is an important point since it allows the research scientists to spend more time working in the virtual environment, and it helps them get more out of the time spent in the VE.

We presented a case study relating how our research into cloud life-cycles progressed, and we described how our new software supports each step of the process. Out of the five functional requirements for data analysis software suggested by Springmeyer et al. in [93] (allow interactive, quantitative exploration; assist in maintaining records of sessions; link materials from different stages of a study; simplify navigation requirements; and provide support for culling large data sets), we now support, at some level, all except assisting with maintaining records of sessions.

In the future, there are three major directions in which we would like to go. First, we would like to begin making regular use of the software in our cloud studies, whereby we hope to further refine and polish it. Secondly, we would like to continue working towards the goal of making the "reprocessing" step a real-time addition to the virtual environment, where new expressions can be entered at run time with their results immediately visible. Finally, we would like to further enhance the environment by including such things as particle tracing, object surface properties, more intelligent large data handling capabilities, and session recording capabilities. It is our hope that moving in these directions will lead to a VE with visualization and analysis facilities capable of supporting the performance of virtual experiments, which will allow scientists to develope and test new hypotheses within the VE.

3.7 Acknowledgments

We would like to thank the Netherlands Organization for Scientific Research (NWO) for providing project funding. We would also like to thank Gerwin de Haan for his valuable insight and comments on the paper.

This chapter was originally published as a peer-reviewed short paper [33] at the Eurographics Symposium on Geometry Processing in 2007.

CHAPTER 4

Fast Normal Vector Compression with Bounded Error

Abstract

We present two methods for lossy compression of normal vectors through quantization using "base" polyhedra. The first revisits subdivision-based quantization. The second uses fixed-precision barycentric coordinates. For both, we provide fast (de)compression algorithms and a rigorous upper bound on compression error. We discuss the effects of base polyhedra on the error bound and suggest polyhedra derived from spherical coverings. Finally, we present compression and decompression results, and we compare our methods to others from the literature.

4.1 Introduction

Since Deering [22] introduced geometry compression in 1995, it has been a popular research topic. Much work in the field has focused on mesh simplification, connectivity compression, and vertex position compression, but vertex attribute and normal compression have received less attention. However, most work dealing with normal compression lacks rigorous analysis. Additionally, many techniques require reasonable amounts of computational resources for decompression, which, when applied to large amounts of compressed data, quickly becomes a bottleneck.

We find compression necessary when visualizing time-dependent data due to the amount of data that must be read from disk during interactive visualization. However, little processing time is available for decompression because visualization often involves a large amount

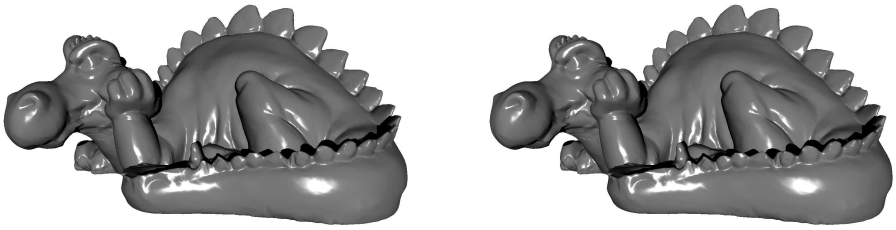


Figure 4.1: Ray-traced images of the smoothed Phlegmatic Dragon with (left) and without (right) compressed normals.

of interactive data processing, e.g. for volume rendering or particle tracing. Furthermore, decompression techniques requiring contextual knowledge are undesirable since they hinder GPU-based decompression and operating on subsets of the data.

Most existing normal compression techniques offer only image-based analysis of compression error, if an analysis is provided. However, image artifacts introduced by errors in normal directions are more visible in some areas than others, e.g. specular highlights and reflections. Thus, image quality assessment is scene, and often viewer, dependent, and it makes quantitative method comparisons difficult.

Here, we focus on compressing normal vectors, with the goals of bounded error and fast (de)compression. Oliveira and Buxton [71] expanded on index-based normal compression using subdivided "base" polyhedra and measured the resulting error. We extend and formalize these ideas, and we provide a method using barycentric coordinates when higher precision is necessary. We provide fast compression and decompression algorithms for both methods, where GPU-based decompression is possible. Furthermore, we are able to analytically derive upper bounds on the error for both methods. Using 16 bits per compressed normal, we are able to achieve an upper bound on the angular error of less than 0.57°, which presents almost no visual difference when used for rendering (Figure 4.1). Based on our error analysis, we suggest the use of new base polyhedra derived from spherical coverings [90].

The remainder of this paper is organized as follows. We discuss related work in Section 4.2. In Section 4.3, we give an overview of our methods, and we cover the mathematical underpinnings. We describe and analyze the methods in Section 4.4. In Section 4.5, we present the results from our work. We conclude and discuss future work in Section 4.6.

4.2 Related Work

A major goal in geometry compression is overcoming transmission bottlenecks. Some work targets the RAM/GPU bottleneck (e.g. [22, 15]). Other work focuses on network transmission (e.g. [99, 97, 100]). Peng et al. [74] and Gotsman et al. [32] give overviews of several techniques. A recent example from Purnomo et al. [78] quantizes all vertex data based on

an image quality metric. Here we primarily list work specifically describing methods for quantizing normal vectors.

One alternative to quantization is entropy encoding. [32] and [74] list several techniques. Entropy encoding, however, requires contextual knowledge, which makes it less desirable when independent normal decompression is important, such as GPU-based implementations or working with data subsets. We note, though, that through careful quantization, it is possible to combine quantization with entropy encoding, such as in [22, 3, 51]. Thus, our techniques could be combined with entropy encoding at the expense of decompression speed.

Most normal quantization methods exploit the face symmetry of face-transitive polyhedra to generate a "uniform" distribution of points on the unit sphere. Deering [22] uses warped spherical coordinates within the faces of a disdyakis dodecahedron. Ahn et al. [3] generate regularly spaced points on the unit cube. The MPEG-4 BInary Format for Scenes (BIFS) [2], also used in QSplat [82], generates non-linearly warped sets of points on the unit cube. MPEG-4 3D Mesh Compression (3DMC) [1] uses a method described by Taubin et al. [98] that uses representative points from the triangles of a recursively subdivided unit octahedron. Botsch et al. [9] also subdivide the unit octahedron, but they project the result onto the unit sphere and use the face normal normals as the representative points. Oliveira and Buxton [71] further expanded on this idea by considering each of the Platonic solids as "base" polyhedra.

Several alternatives to the polyhedral methods exist. One approach is to use fixed precision spherical coordinates. Isenburg and Snoeyink [51] extract and quantize the smallest two components of each normal vector. Another possibility is to quantize normals using the points generated by the HEALPix method (Górski et al. [31]), which divides the unit sphere into regions of equal area.

One element lacking from these methods, however, is a rigorous error analysis of the quantization. Deering [22] stated that compression errors should be at most 0.01 radians (0.573°), to prevent visible artifacts, but he presented no analysis to verify that his method satisfied this criterion. Aside from image-based metrics, the only error analysis we are aware of since then is from Oliveira and Buxton [71]. They measured the errors resulting from compressing normals from various models using quantized normals derived from subdividing the Platonic solids. We are able to provide a rigorous upper bound on the error resulting from quantization based on our methods and based on optimal quantization. Using these error bounds, we have experimented with even more base polyhedra, and we have improved, often significantly, on the error bounds from methods in the literature. Based on our comparisons with other methods, we found that our subdivision method is the only method currently able to satisfy Deering's criterion at 16-bit precision.

4.3 Overview and Underpinnings

We propose two methods for compressing normal vectors through quantization: a subdivision method and a barycentric method. Both methods refine convex polyhedra with vertices on the unit sphere by splitting each face into a set of similar triangles and projecting the new vertices onto the unit sphere (Figure 4.2, left). Quantization is is based on replacing normal vectors

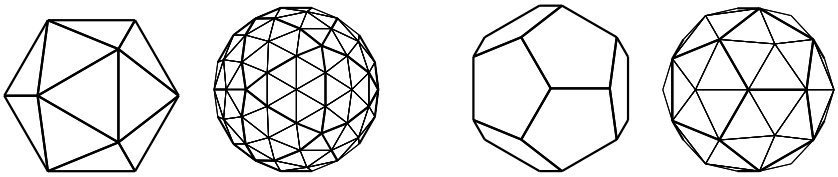


Figure 4.2: Left: An icosahedron "refined" by subdividing each face and projecting the new vertices onto the unit sphere. Right: A dodecahedron and a dodecahedron triangulated by introducing a vertex at the center of each face.

with the "closest" vertex from a refined polyhedron. Any non-triangular polyhedral faces are first triangulated by introducing a vertex at the face centroid, connecting each of the face's vertices to this new vertex, and projecting the new vertex onto the unit sphere (Figure 4.2, right). For the method details, see Sections 4.4.2 and 4.4.3.

4.3.1 Definitions and Notation

Here, we consider *unit normal vectors*, or *normals*, and points on the unit sphere to be interchangeable. Boldface will be used to indicate when point p on the unit sphere is being treated as normal vector $\mathbf{p}$. In general, we will refer to a normal as $\mathbf{n}$ and the quantization of $\mathbf{n}$ as $\mathbf{n}^\star$.

A *polyhedron*, $P = (F,V,E)$, has edges, $E = \{e_1,\ldots,e_k\}$, faces, $F = \{f_1,\ldots,f_\ell\}$, and vertices, $V = \{v_1,\ldots,v_m\}$. All vertices lie on the unit sphere, and each face, f_i, has a face normal $\mathbf{n}_{f_i}$. $F(P)$ are the faces of polyhedron P, and $V(P)$ are the vertices. A *triangulated polyhedron* has only triangular faces.

The shortest distance between two points, p and q, on the unit sphere is the angular distance between them:

$$\text{dist}(p,q) = \arccos(\mathbf{p}\cdot\mathbf{q}). \tag{4.1}$$

The error, ε, between a normal, $\mathbf{n}$, and its quantization, $\mathbf{n}^\star$, is the angular distance between them:

$$\varepsilon = \text{dist}(n,n^\star) = \arccos(\mathbf{n}\cdot\mathbf{n}^\star). \tag{4.2}$$

4.3.2 Normal Quantization

A normal is quantized in two steps. First, given normal, $\mathbf{n}$, and triangular faces, F, where $\mathbf{n}$ intersects at least one $f \in F$, one such $f \in F$ must be selected. This f is selected with the FIND_FACE algorithm (Algorithm 1). This algorithm iterates through all faces in F and chooses the first face that is intersected by $\mathbf{n}$. If, due to numerical precision issues, no such face is found, then f is chosen as the face whose normal is closest to $\mathbf{n}$. Second, the closest vertex from that face to $\mathbf{n}$ is selected using the QUANTIZE algorithm (Algorithm 2). This algorithm compares the distance from $\mathbf{n}$ to each of the vertices of f in turn and chooses the closest one to $\mathbf{n}$. This closest vertex is the quantized normal, $\mathbf{n}^\star$. Thus, given a triangulated

polyhedron, P, $\mathbf{n}$ is quantized:

$$\mathbf{n}^\star = \text{QUANTIZE}(\text{FIND\_FACE}(F(P), \mathbf{n}), \mathbf{n}). \tag{4.3}$$

Algorithm 1 FIND_FACE($F, \mathbf{n}$)

$F \leftarrow f_1, \ldots, f_n$ {triangular faces}, $\mathbf{n} \leftarrow$normal vector
$d \leftarrow -1$, $best \leftarrow 0$
for all $f_i \in F$ **do**
  v_a, v_b, $v_c \leftarrow$ vertices of f_i, counterclockwise
  $\mathbf{a} \leftarrow (\mathbf{v}_c \times \mathbf{v}_b)$, $\mathbf{b} \leftarrow (\mathbf{v}_a \times \mathbf{v}_c)$, $\mathbf{c} \leftarrow (\mathbf{v}_b \times \mathbf{v}_a)$
  if $(\mathbf{a} \cdot \mathbf{n}) > 0$ **and** $(\mathbf{b} \cdot \mathbf{n}) > 0$ **and** $(\mathbf{c} \cdot \mathbf{n}) > 0$ **then**
    return f_i {$\mathbf{n}$ intersects f_i}
  else if $(\mathbf{n}_{f_i} \cdot \mathbf{n}) > d$ **then**
    $best \leftarrow i$, $d \leftarrow (\mathbf{n}_{f_i} \cdot \mathbf{n})$ {$\mathbf{n}$ is close to face normal $\mathbf{n}_{f_i}$}
  end if
end for
return f_{best}

Algorithm 2 QUANTIZE($f, \mathbf{n}$)

$f \leftarrow$ triangular face, $\mathbf{n} \leftarrow$normal vector
v_a, v_b, $v_c \leftarrow$ vertices of f
if $(\mathbf{v}_a \cdot \mathbf{n}) > (\mathbf{v}_b \cdot \mathbf{n})$ **and** $(\mathbf{v}_a \cdot \mathbf{n}) > (\mathbf{v}_c \cdot \mathbf{n})$ **then**
  return $\mathbf{v}_a$ {$\mathbf{n}$ is closest to $\mathbf{v}_a$}
else if $(\mathbf{v}_b \cdot \mathbf{n}) > (\mathbf{v}_c \cdot \mathbf{n})$ **then**
  return $\mathbf{v}_b$ {$\mathbf{n}$ is closest to $\mathbf{v}_b$}
else
  return $\mathbf{v}_c$ {$\mathbf{n}$ is closest to $\mathbf{v}_c$}
end if

4.3.3 Error Bound

We will now prove an upper bound on the error for quantizing a given normal. We will show that, if Equation 4.3 is used for quantization, then, given a convex, triangulated polyhedron, P, and a normal, $\mathbf{n}$, and its quantization, $\mathbf{n}^\star$:

$$\text{dist}(n, n^\star) \leq \max(\{\text{dist}(v_i, n_{f_j}) | v_i \in f_j, f_j \in F(P)\}).$$

That is, the maximum error between any normal and its quantization is at most the maximum distance between a face's normal and its vertices. Thus, the error bound can be analytically determined by examining each polyhedral face.

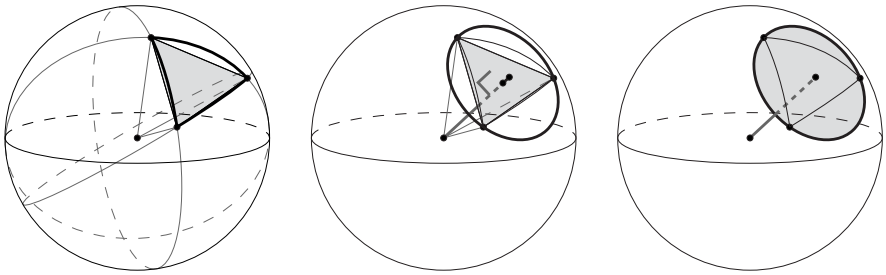


Figure 4.3: Left: Spherical triangle and underlying planar triangle. Middle: The face normal of the planar triangle intersects triangle's circumcenter and the spherical triangle's circumcenter. Both triangles share the same circumcircle. Right: The circumcircle defines a spherical cap.

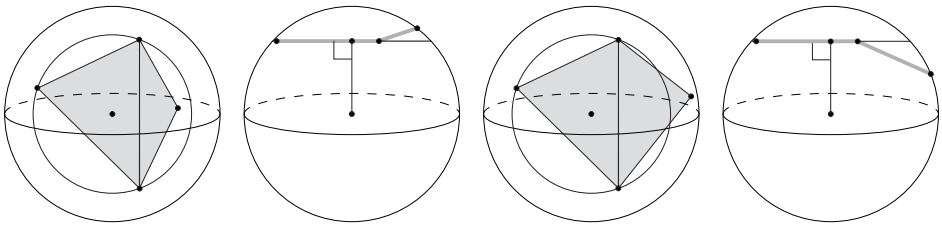


Figure 4.4: Left: Front and top view of a vertex from one face lying on the spherical cap defined by another face. Right: Front and top view a vertex from one face lying outside the spherical cap defined by another face.

First, we observe that every face of P represents a planar triangle and the spherical triangle that is its projection onto the unit sphere (Figure 4.3 left). Hence, the faces of P also represent a set of spherical triangles covering the unit sphere.

Next, we note that the vertices, v_a, v_b, and v_c, of triangular face, f_i, lie on the unit sphere. These three points define a circle on the unit sphere, which is the circumcircle of f_i (Figure 4.3 middle) and its spherical triangle. This circumcircle also defines a spherical cap, which lies "above" the plane defined by f_i (Figure 4.3 right). The line from the origin through the circle's center is normal to the plane in which the circle lies, which is the plane defined by f_i. Thus, the normal vector of f_i, $\mathbf{n}_{f_i}$ intersects f_i at its circumcenter. $\mathbf{n}_{f_i}$, also intersects the center of the spherical cap defined by the circle. Therefore, $\mathbf{n}_{f_i}$ is also the circumcenter for the spherical triangle and so $\text{dist}(n_{f_i}, v_a) = \text{dist}(n_{f_i}, v_b) = \text{dist}(n_{f_i}, v_c)$.

To proceed, we introduce the following lemma.

Lemma 1. *If triangulated polyhedron P is convex, then $\forall f_i \in F(P)$, $\text{dist}(n_{f_i}, v_j) \leq \text{dist}(n_{f_i}, v_k)$ with $v_j \in f_i$, $v_k \notin f_i$, and $v_k, v_j \in V(P)$.*

Briefly, the lemma states that, for a convex, triangulated polyhedron, no vertex from one face may lie on the interior of the spherical cap defined by another face. See Figure 4.4.

Proof. Given the triangulated polyhedron, P, select face, $f_i \in F(P)$. Suppose that there is

some vertex, $v_k \in V(P)$, with $v_k \notin f_i$, that is closer to n_{f_i} than the vertices of f_i. Therefore, v_k lies on the interior of the spherical cap defined by f_i and is "above" the plane defined by f_i. Thus, there must exist a line segment between v_k and some vertex of f_i such that some part of the line segment passes "above" f_i. Since P is convex, nothing inside P can be "above" f_i so some part of the line segment must pass through the exterior of P. However, no line segment connecting two vertices of a convex polyhedron may pass through its exterior. Therefore, there can be no such vertex v_k if P is convex. □

One result of this lemma is that the spherical triangulation defined by convex, triangulated polyhedron P is, in fact, a Delaunay triangulation of the unit sphere since no vertex from one face lies on the interior of the circumcircle (spherical cap) of another face (see, for example, [29]). Hence, the vertices of P are the "sites" of a spherical Voronoi diagram covering the unit sphere, and the face normals are the Voronoi vertices. This leads to an "ideal" error bound and an error bound specific to our normal compression methods.

The ideal error bound deals with normals quantized by replacing them with the closest vertices from P, and the more specific error bound deals with normals quantized using Equation 4.3. These bounds are equivalent, and we will use the ideal bound to compare our methods to others from the literature. We will now use Lemma 1 to first prove the ideal error bound and then to prove the specific error bound.

Theorem 1. *Given a convex, triangulated polyhedron, P, a normal, $\mathbf{n}$, and the closest vertex, $v^\star \in V(P)$, to $\mathbf{n}$, then:*

$$\text{dist}(n, v^\star) \leq \max(\{\text{dist}(v_i, n_{f_j}) | v_i \in f_j, f_j \in F(P)\}). \tag{4.4}$$

Essentially, this theorem states that, if a normal is quantized by replacing it with the closest vertex from P, then the quantization error will be less than the maximum distance between a triangular face vertex and the normal for that face.

Proof. From Lemma 1, P defines a spherical Voronoi diagram covering the unit sphere, where $V(P)$ are the sites and the face normals of P are the Voronoi vertices. Clearly, all Voronoi cells from a spherical Voronoi diagram are bounded, and, for bounded Voronoi cells, the farthest points on a cell from the cell's site are Voronoi vertices of the cell. The Voronoi vertices for the cell with vertex, $v \in V(P)$, as its site are the face normals, n_{f_k} from all faces, $f_k \in F(P)$, with $v \in f_k$. Thus, for any point, p, in that cell, we have: $\text{dist}(p, v) \leq \max(\{\text{dist}(n_{f_k}, v) | v \in f_k\})$. Since, a given point, n, on the unit sphere, must lie on some Voronoi cell with some vertex, $v^\star \in V(P)$, as its site, then we know that: $\text{dist}(n, v^\star) \leq \max(\{\text{dist}(v_i, n_{f_j}) | v_i \in f_j, f_j \in F(P)\})$. □

We now prove the bound on quantization with Equation 4.3.

Theorem 2. *Given a convex, triangulated polyhedron, P, a normal, $\mathbf{n}$, and its quantization, $\mathbf{n}^\star$, from by Equation 4.3, then:*

$$\text{dist}(n, n^\star) \leq \max(\{\text{dist}(v_i, n_{f_j}) | v_i \in f_j, f_j \in F(P)\}). \tag{4.5}$$

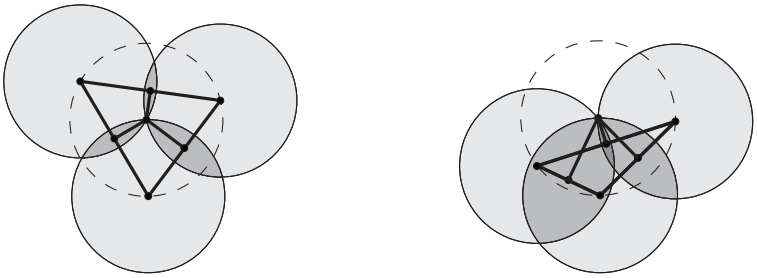


Figure 4.5: Acute (left) and obtuse (right) triangles, with the perpendicular bisectors shown and copies of the circumscribing circle placed at the vertices.

Here, the idea is that quantizing normals by replacing them with the closest vertex from the triangular face they intersect gives the same error bound as Theorem 1, where normals are replaced by the closest vertex from P.

Proof. From Lemma 1, P defines a spherical Voronoi diagram covering the unit sphere, where $V(P)$ are the sites and the face normals of P are the Voronoi vertices. We also know that each face, $f_i \in F(P)$, has n_{f_i} as the center of the circle/spherical cap circumscribing f_i. As the circumcenter, n_{f_i} is coincident with the intersection of the perpendicular bisectors of the sides of the spherical triangle defined by f_i. These bisectors define three regions in the spherical triangle, where all points from each region are closest to one triangle vertex and are at most the radius of the spherical cap away from the that vertex. See Figure 4.5 for examples of the planar case. The radius of the spherical cap is precisely the distance between n_{f_i} and any vertex of f_i. Thus, the maximum distance that any point on the unit sphere can be from the closest vertex of the spherical triangle that contains it, is the distance between the normal of that face and any of its vertices. Since P is triangulated and convex and all normals are quantized using Equation 4.3, the inequality in Theorem 2 holds. □

4.3.4 Quantization Optimality

Here, we will prove that, given a convex, triangulated polyhedron, P, Equation 4.3 will optimally quantize normal vector, $\mathbf{n}$, if P has only acute triangles. By optimal quantization, we mean that $n^\star$ will be the closest vertex from P to n.

Theorem 3. *Given convex, triangulated polyhedron P, with all faces acute triangles, normal,* $\mathbf{n}$*, and its quantization,* $\mathbf{n}^\star$*, from by Equation 4.3, then:*

$$\text{dist}(n, n^\star) = \min(\{\text{dist}(n, v_i) | v_i \in V(P)\}). \tag{4.6}$$

Proof. From Lemma 1, P defines a spherical Voronoi diagram covering the unit sphere, where $V(P)$ are the sites and the face normals of P are the Voronoi vertices. We also know that, since

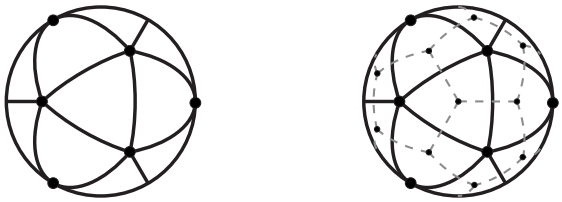


Figure 4.6: Left: Spherical triangles defined by an icosahedron. Right: The same spherical triangles with the spherical Voronoi diagram illustrated.

the faces of P are acute triangles, the circumcenters of the faces lie within the faces themselves. Therefore, the circumcenters of the spherical triangles also lie within the spherical triangles. Thus, each spherical triangle contains one Voronoi vertex, and the Voronoi edges perpendicularly bisect the sides of the spherical triangles (Figure 4.6). Each spherical triangle lies in three Voronoi cells, the Voronoi sites of which are the spherical triangle's vertices. Therefore, no point in spherical triangle, $\triangle ABC$, is closer to a vertex from another spherical triangle than it is to one of the vertices of $\triangle ABC$. Thus, quantizing $\mathbf{n}$ using Equation 4.3 will result in the smallest possible distance between $\mathbf{n}^\star$ and $\mathbf{n}$. □

One property of this theorem is that Equation 4.3 is relatively robust to numerical error. The Voronoi cell for each vertex occupies a portion of all spherical triangles containing that vertex. Thus, even if a normal is near the edge of a spherical triangle and Algorithm 1 selects the incorrect face, Algorithm 2 will likely select the correct vertex. The cases where Algorithm 2 could select the incorrect polyhedron vertex are those when the normal is near a Voronoi edge. Since Voronoi edges are equidistant from Voronoi sites, i.e. the polyhedron vertices, then, in those situations, selecting the wrong vertex has little effect on the quantization error.

It is important to note that this theorem does not hold for convex, triangulated polyhedra containing obtuse triangles. In such polyhedra, some triangles will contain multiple Voronoi vertices, and, therefore, some regions of those triangles will be closer to a vertices from adjacent triangles. However, from Theorems 1 and 2, we know that, while some normals may be non-optimally quantized in such polyhedra, the upper bound on the error remains the same. In order to preserve the optimality of the quantization, though, we should use polyhedra containing only acute triangles.

As an aside, existing subdivision methods [71, 9, 1] are relatively susceptible to error. In these methods, a normal is quantized by replacing it with a representative point from the face it intersects. However, there is no guarantee that a normal is closer to the representative point from that face than it is to those of neighboring faces, leading to possible non-optimal quantizations. This exacerbates potential errors from incorrect face selection.

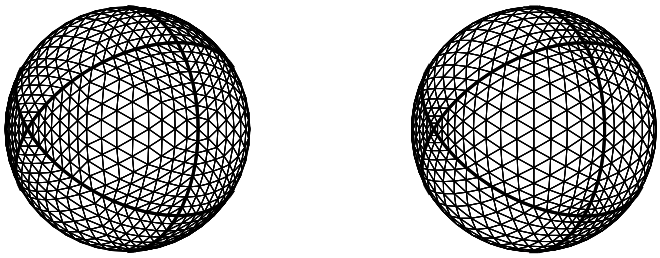


Figure 4.7: Octahedron refined using the subdivision method (left) and the barycentric method (right).

4.3.5 Euler Characteristic

The Euler Characteristic for a polyhedron is defined as:

$$X = V - E + F$$

where X is the Euler Characteristic and V, E and F are respectively the numbers of vertices, edges, and faces of the polyhedron. $X = 2$ for the simply connected polyhedra we work with, and, since we only work with triangulated polyhedra, we have this useful relation:

$$V = \frac{F}{2} + 2 \tag{4.7}$$

4.4 Normal Compression

In this section, we provide the details over our two proposed normal compression methods: the subdivision method and the barycentric method. See Figure 4.7.

4.4.1 Bit precision and efficiency

Bit precision is the number of bits used to represent a compressed normal. In our approaches, all compressed normals from a given set are represented with the same bit precision.

For a given bit precision, b, there are 2^b unique bit strings. Ideally, this would also mean quantization methods would generate 2^b unique normals. However, this is often not the case. In general, for a given bit precision and quantization scheme, a polyhedron generating more unique normals will have a lower error bound than one generating fewer unique normals (Table 4.1). Therefore, it is desirable to seek out combinations generating more unique normals.

4.4.2 Subdivision Method

The subdivision method generates a set of quantized normals by recursively subdividing the faces of a triangulated polyhedron. At each subdivision level, each polyhedral face is subdivided by introducing new vertices at the midpoints of face edges and projecting the new

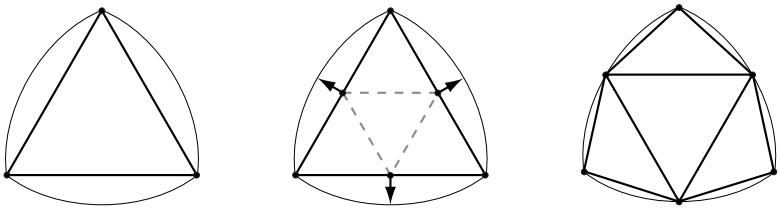


Figure 4.8: Triangular face (left) subdivided by introducing vertices at edge midpoints (middle), which are projected onto the unit sphere (right).

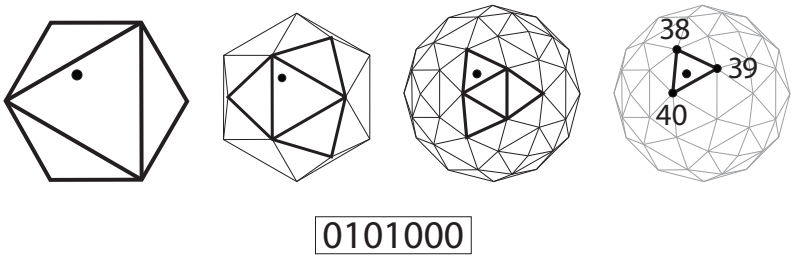


Figure 4.9: A normal quantized by recursively finding the face it intersects and then selecting the closest vertex from the final face. The quantized normal is a normal table index.

vertices onto the unit sphere. See Figure 4.8. For a triangulated polyhedron, P_0, polyhedra, $P_1,\ldots,P_n$, represent different levels of subdivision.

Before a set of normals can be compressed, the base polyhedron, P_0, is first refined to a certain subdivision level, P_ℓ, and every vertex from P_ℓ is assigned a unique ID number. This defines the normal table. Once the table is constructed, normals are quantized using Algorithm 3. See Figure 4.9. If the base polyhedron, P_0, and all refined polyhedron, P_i, are convex, then the guarantee from Theorem 3 will hold, and the compression process will always find the closest polyhedron vertex for each normal.

Given a base polyhedron, P_0, and a level of subdivision, s, the number of unique quantized normals generated by the subdivision method is $|V(P_s)|$. Since P_s has $(4^s)|F(P_0)|$ faces, we can use Equation 4.7 to compute this:

$$|V(P_s)| = \frac{(4^s)|F(P_0)|}{2} + 2. \tag{4.8}$$

However, the normal table can contain at most 2^b entries at bit precision, b. Therefore, we must calculate the maximum subdivision level, ℓ, for P_0 such that $|V(P_\ell)| \leq 2^b$ using the following equation derived from Equation 4.8:

$$\ell = \lfloor \log_4(2^{b+1} - 4) - log_4|F(P_0)| \rfloor. \tag{4.9}$$

Given M normal vectors and base polyhedron P_0 refined to subdivision level ℓ, decom-

Algorithm 3 COMPRESS_SUBDIVISION($P, \mathbf{n}, \ell$)

$P \leftarrow$ triangulated polyhedron {vertices have unique ID}
$\mathbf{n} \leftarrow$normal vector
$\ell \leftarrow$ maximum level of subdivision
$f \leftarrow$ FIND_FACE($F(P), \mathbf{n}$)
for $i = 1 \ldots \ell$ **do**
  $f_a, f_b, f_c, f_d \leftarrow$ subdivided faces of f
  $f \leftarrow$ FIND_FACE($\{f_a, f_b, f_c, f_d\}, \mathbf{n}$)
end for
$v \leftarrow$ QUANTIZE($f, \mathbf{n}$)
return v.id

pression and compression complexities are as follows. Decompression only requires a normal table look-up, and thus is $O(M)$. The normal table can be constructed in $O((4^\ell)|F(P_0)|)$ time. For large values of M and small values of ℓ and $|F(P_O)|$, this time is negligible. Compression of the normals takes $O(M(|F(P_0)| + 4\ell))$ time.

The subdivision method offers two chief advantages. First, decompression is trivial, and thus incurs almost no computational overhead. Second, with careful polyhedron selection, it is able generate almost the maximum number of unique normals, which generally results in a lower error bound. The normal table must be kept in memory, however, which is costly for sufficiently large normal tables. Given that normal table index bit precisions are likely to be multiples of 8 bits, then tables with indices above 16-bit precision begin to be impractical.

4.4.3 Barycentric Method

The barycentric method is based on refining faces of a base triangulated polyhedron by computing fixed-precision barycentric coordinates. This divides the face into a set of similar triangles, and the newly introduced vertices are then projected onto the unit sphere. See Figure 4.10. Unlike the subdivision method, the barycentric method is not recursive, and it does not generate any intermediate polyhedra between the base polyhedron, P, and the refined polyhedron, P_r. In fact, P_r is never explicitly generated. Instead, faces from P_r are computed as necessary.

The compressed normals are represented as bit strings consisting of three integers (Figure 4.11). The first identifies the face from the base polyhedron, which the quantized normal intersects. The next two integers represent the u and v barycentric coordinates. Since the barycentric coordinates u, v and w sum to 1, there is no need to explicitly store w. The compression algorithm is presented in Algorithm 4. If both the base polyhedron, P, and the refined polyhedron, P_r, are convex, then the guarantee from Theorem 3 will hold, and the compression process will always find the closest polyhedron vertex for each normal vector.

The bit precision for the barycentric method is divided between the face identifier and the (equal precision) integer u and v coordinates. A base polyhedron, P, with $|F(P)|$ faces, requires a minimum bit precision of $\log_2 |F(P)| + 2$. If u and v are each c bits long, then the

Algorithm 4 COMPRESS_BARYCENTRIC($P, \mathbf{n}, p$)

$P \leftarrow$ triangulated polyhedron {faces have unique ID}
$\mathbf{n} \leftarrow$normal vector
$p \leftarrow$ barycentric coordinate bit precision
$f \leftarrow$ FIND_FACE$(F(P), \mathbf{n})$
v_a, v_b, $v_c \leftarrow$ vertices of f
$\hat{n} \leftarrow$ intersection point between $\mathbf{n}$ and f
$d \leftarrow 2^p - 1$
u, v, $w \leftarrow$ barycentric coordinates of $\hat{n}$ in f
if $d - (\lfloor u*d \rfloor + \lfloor v*d \rfloor + \lfloor w*d \rfloor) = 2$ **then**
  $u_x \leftarrow \frac{\lceil u*d \rceil}{d}$, $v_x \leftarrow \frac{\lceil v*d \rceil}{d}$, $w_x \leftarrow \frac{\lfloor w*d \rfloor}{d}$
  $u_y \leftarrow \frac{\lceil u*d \rceil}{d}$, $v_y \leftarrow \frac{\lfloor v*d \rfloor}{d}$, $w_y \leftarrow \frac{\lceil w*d \rceil}{d}$
  $u_z \leftarrow \frac{\lfloor u*d \rfloor}{d}$, $v_z \leftarrow \frac{\lceil v*d \rceil}{d}$, $w_z \leftarrow \frac{\lceil w*d \rceil}{d}$
else if $d - (\lfloor u*d \rfloor + \lfloor v*d \rfloor + \lfloor w*d \rfloor) = 1$ **then**
  $u_x \leftarrow \frac{\lfloor u*d \rfloor}{d}$, $v_x \leftarrow \frac{\lfloor v*d \rfloor}{d}$, $w_x \leftarrow \frac{\lceil w*d \rceil}{d}$
  $u_y \leftarrow \frac{\lfloor u*d \rfloor}{d}$, $v_y \leftarrow \frac{\lceil v*d \rceil}{d}$, $w_y \leftarrow \frac{\lfloor w*d \rfloor}{d}$
  $u_z \leftarrow \frac{\lceil u*d \rceil}{d}$, $v_z \leftarrow \frac{\lfloor v*d \rfloor}{d}$, $w_z \leftarrow \frac{\lfloor w*d \rfloor}{d}$
else
  return f.id, $u*d$, $v*d$
end if
$\mathbf{x} \leftarrow u_x\mathbf{v}_a + v_x\mathbf{v}_b + w_x\mathbf{v}_c$
$\mathbf{y} \leftarrow u_y\mathbf{v}_a + v_y\mathbf{v}_b + w_y\mathbf{v}_c$
$\mathbf{z} \leftarrow u_z\mathbf{v}_a + v_z\mathbf{v}_b + w_z\mathbf{v}_c$
$\mathbf{n}^\star \leftarrow$ QUANTIZE$(\{\frac{\mathbf{x}}{|\mathbf{x}|}, \frac{\mathbf{y}}{|\mathbf{y}|}, \frac{\mathbf{z}}{|\mathbf{z}|}\}, \mathbf{n})$
if $\mathbf{n}^\star = \mathbf{x}$ **then**
  return f.id, u_x*d, v_x*d
else if $\mathbf{n}^\star = \mathbf{y}$ **then**
  return f.id, u_y*d, v_y*d
else
  return f.id, u_z*d, v_z*d
end if

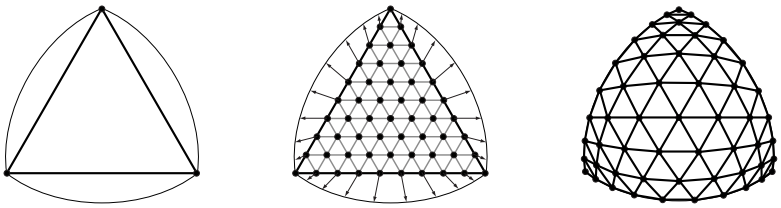


Figure 4.10: Triangular face (left) refined by introducing vertices at barycentric coordinates of fixed-precision (middle), which are projected onto the unit sphere (right).

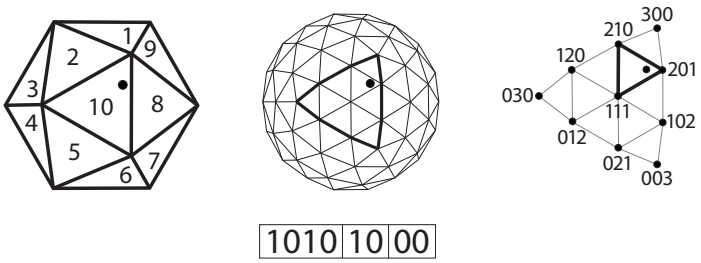


Figure 4.11: A normal quantized by finding the face it intersects and computing barycentric coordinates. The quantized normal is a face index and two fixed-precision coordinates.

number of unique vertices in P_r can be computed with this equation:

$$|V(P_r)| = \frac{(2^c - 1)^2 |F(P)|}{2} + 2 \tag{4.10}$$

Given M normal vectors and a base polyhedron P, decompression and compression complexities are as follows. Decompression requires a face look-up, computing a weighted sum of the vertices and normalizing the result. This requires constant time, and thus decompression is $O(M)$. Compression of the normals takes $O(M|F(P_0)|)$ time since the face from the base polyhedron containing the normal must be found before the barycentric coordinates can be computed.

The primary advantage of the barycentric method is its fast decompression algorithm, which does not require a large normal table. In comparison with the subdivision method, the barycentric method has two disadvantages. For an equivalent bit precision, the barycentric method will generate fewer unique normals. This is not a significant disadvantage though, since, for higher bit precisions, it is impractical to keep a normal table in memory. Secondly, the distribution of quantized normals generated by the barycentric method tends to distort more in the center of the base polyhedron faces, whereas the subdivision method produces more homogeneous distributions. For smaller base polyhedron faces, this tends to mean that the subdivision method generates more even distributions, which usually result in lower maximum errors. See Figure 4.7.

4.4.4 Base Polyhedron Selection

Up to this point, we have not discussed the selection of a base polyhedron for these methods, but this selection has a significant impact on the error bound. While we are able to analytically determine the upper bound on the error for a given polyhedron, different base polyhedra will result in different refined polyhedra, which will have different upper bounds on the quantization error. We examined a variety of base polyhedra to determine which offered the lowest upper bound. We looked at the Platonic solids, used by Oliveira and Buxton [71], Archimedean solids, Catalan solids, and polyhedra generated by computing the convex hulls of spherical coverings from Sloane et al. [90]. To compute the convex hulls, we used the QHull software [7]. For all the polyhedra we considered, we triangulated any non-triangular faces (Figure 4.2 right), and we projected all vertices onto the unit sphere.

We defined five criteria for a polyhedron to be considered suitable for use with our methods.

1. The polyhedron must be convex.
2. The faces of the polyhedron must all be acute triangles.
3. The polyhedron must have 256 or fewer faces.
4. The polyhedron must remain convex when refined.
5. Faces of refined polyhedra must also be acute triangles.

In our experience, the fourth and fifth criteria are met if the angles of the spherical triangles defined by the base polyhedron faces are all less than or equal to 90°. We have been able to verify this numerically, but we have not yet been able to provide a formal proof.

From our set of polyhedra, these criteria ruled out several of the Catalan solids and, notably, the tetrahedron, which fails to satisfy criterion 4. We then tested the remaining polyhedra to see which yielded the lowest maximum compression error at different bit precisions for both methods. Results from the polyhedra in Figure 4.12 are in Table 4.1. We found that carefully chosen spherical coverings from Sloane et al. yield a lower maximum error in than the Platonic, Archimedean, or Catalan solids in each of the four cases.

Furthermore, we found that the better spherical coverings for the barycentric method performed at least as well as some of the poorer polyhedra from the subdivision method for equivalent bit precisions. The best spherical covering from the barycentric method with 24-bit precision reduces the maximum error by an order of magnitude over the best spherical covering for the subdivision method at 16-bit precision.

4.5 Results

In this section, we present results related to our compression and decompression algorithms. First, we present details about the performance of our compression and decompression methods. Secondly, we compare our method to various methods from the literature.

			Subdivision Method			
			12-Bit Precision		16-Bit Precision	
Polyhedron	F	V	$\|N\|$	ε_{max}	$\|N\|$	ε_{max}
Cube*	24	14	3,074	2.54°	49,154	0.636°
Octahedron	8	6	1,026	5.05°	16,386	1.27°
Dodecahedron*	60	32	1,922	2.99°	30,722	0.747°
Icosahedron	20	12	2,562	2.73°	40,962	0.684°
Disdyakis Triacontahedron	120	62	**3,842**	2.38°	**61,442**	0.595°
Rhombicuboctahedron*	80	42	2,562	3.24°	40,962	0.811°
Spherical Covering 1	30	17	**3,842**	**2.24**°	**61,442**	**0.561**°
Spherical Covering 2	126	65	**4,034**	**2.18**°	**64,514**	**0.546**°
Spherical Covering 3	64	34	2,050	2.94°	32,770	0.735°
Spherical Covering 4	256	130	2,050	2.93°	32,770	0.733°

			Barycentric Method			
			16-Bit Precision		24-Bit Precision	
Polyhedron	F	V	$\|N\|$	ε_{max}	$\|N\|$	ε_{max}
Cube*	24	14	11,534	1.35°	3,133,454	0.0821°
Octahedron	8	6	15,878	1.29°	4,186,118	0.0792°
Dodecahedron*	60	32	28,832	0.773°	7,833,632	0.0469°
Icosahedron	20	12	9,612	1.41°	2,611,212	0.0857°
Disdyakis Triacontahedron	120	62	13,502	1.29°	3,901,502	0.0761°
Rhombicuboctahedron*	80	42	9,002	1.73°	2,601,002	0.204°
Spherical Covering 1	30	17	14,417	1.16°	3,916,817	0.0703°
Spherical Covering 2	126	65	14,177	1.10°	4,096,577	0.0649°
Spherical Covering 3	64	34	**30,754**	**0.759**°	**8,355,874**	**0.0461**°
Spherical Covering 4	256	130	**28,802**	**0.764**°	**8,323,202**	**0.0450**°

Table 4.1: Base polyhedra (Figure 4.12) with face (F) and and vertex (V) counts. Unique normals, $|N|$, and error upper bound, ε_{max}, are listed for each at 12 and 16-bit precisions (subdivision method) and at 16 and 24-bit precisions (barycentric method). Polyhedra marked with an asterisk were triangulated for use with our methods. The best polyhedra in each column are bolded.

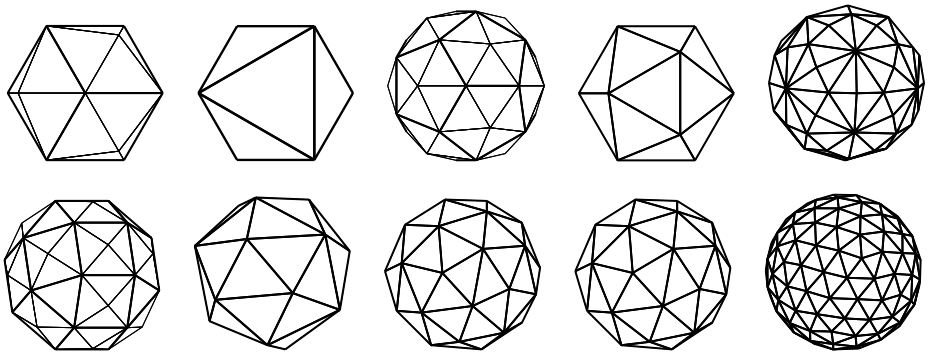


Figure 4.12: Ten (triangulated) base polyhedra. Top row, from left to right: Cube, Octahedron, Dodecahedron, Icosahedron, Disdyakis Triacontahedron. Bottom row, from left to right: Rhombicuboctahedron, Spherical Coverings 1 through 4.

4.5.1 Performance

We tested the performance of our methods on six well-known models. For models lacking normals, normals were generated using the PLY tools provided by the Stanford 3D Scanning Repository[1]. Degenerate normals were ignored. We recorded the running time for both compression and decompression for the subdivision and barycentric models using a 3.0 GHz Pentium 4 machine. In our timings, we only timed the performance of compression and decompression on data resident in main memory, and we did not include the time necessary to load the required data from disk. For each method, we recorded both the maximum error found between a normal vector and the compressed normal during quantization and the average of all the errors. For the subdivision method, we used Spherical Covering 1 (Figure 4.12, Table 4.1) as the base polyhedron, and, for the barycentric method, we used Spherical Covering 3. We chose these coverings over 2 and 4 for performance reasons. The error bounds are only slightly higher, and, since the compression times for these schemes are linear in the number of faces in the base polyhedron, the compression times are lower with these base polyhedra.

Table 4.2 lists the performance of our subdivision and barycentric methods on six well-known models. In all cases, the maximum recorded error remained below the analytically derived error upper bound, and the average recorded error was slightly more than half of the maximum error. The compression and decompression times for the barycentric method follow a clear linear trend that increases with the number of normal vectors. The compression times for the subdivision method shows a non-linear trend due to the overhead of explicitly constructing the refined polyhedron from the base polyhedron. For larger numbers of normal vectors, though, the subdivision method proves to be faster than the barycentric method.

The memory requirements for compression and decompression are not listed in the table,

[1] http://graphics.stanford.edu/data/3Dscanrep/

		16-Bit Precision Subdivision Spherical Covering 1			
Model	Normals	t_{com}	t_{dec}	$\max(\varepsilon)$	$\text{mean}(\varepsilon)$
Stanford Bunny	35,947	0.798s	0.0002s	0.548°	0.312°
Armadillo	172,974	1.60s	0.0012s	0.557°	0.312°
Happy Buddha	543,652	3.43s	0s.0036	0.556°	0.312°
Phlegmatic Dragon	703,018	3.53s	0.0040s	0.554°	0.310°
David (2mm)	6,924,951	30.3s	0.042s	0.560°	0.311°
Lucy	14,027,872	60.8s	0.086s	0.556°	0.311°

		24-Bit Precision Barycentric Spherical Covering 3			
Model	Normals	t_{com}	t_{dec}	$\max(\varepsilon)$	$\text{mean}(\varepsilon)$
Stanford Bunny	35,947	0.180s	0.008s	0.0458°	0.0267°
Armadillo	172,974	0.92s	0.042s	0.0458°	0.0266°
Happy Buddha	543,652	2.80s	0.131s	0.0459°	0.0267°
Phlegmatic Dragon	703,018	3.56s	0.169s	0.0448°	0.0267°
David (2mm)	6,924,951	34.0s	1.64s	0.0460°	0.0267°
Lucy	14,027,872	70.0s	3.36s	0.0460°	0.0267°

Table 4.2: Compression, t_{com}, and decompression, t_{dec}, times for the normals from various known models. Average, $\text{mean}(\varepsilon)$, and maximum, $\max(\varepsilon)$, error recorded during compression are listed. Normals were compressed in both methods using polyhedra derived from spherical coverings (See Section 4.4.4). In all cases, $\max(\varepsilon)$ remained below ε_{max} (Table 4.1).

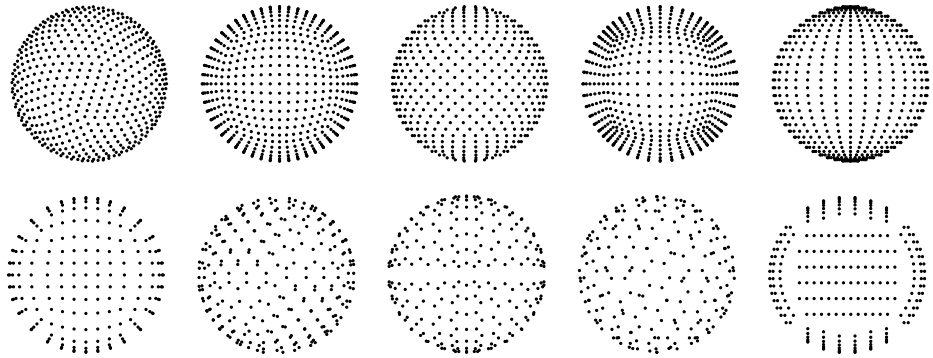


Figure 4.13: Ten sets of quantized normals generated at 10 bits of precision. Top row, from left to right (corresponding with Table 4.3): our method, BIFS, HEALPix, Cube, Spherical Coordinates. Bottom row, from left to right: Deering, Octahedron, 3DMC, PNORMS, and Projection.

but they are quite low. For compression with the subdivision method, the normal table of about 65,536 12-byte normals must be generated and kept in memory. This table is at most 768 kilobytes. For decompression with the subdivision method, this table must also be kept in memory. However, all compressed normals share the same normal table so only one copy of this table need be kept in memory. Note, though, that memory use for the subdivision method grows exponentially with the bit precision of the quantized normals so 16-bits is a practical upper limit on the index bit precision. For the barycentric method, the base polyhedron must be kept in memory, which means that each vertex must be stored as well as a list of faces, which index into the vertices. For Spherical Covering 3, this is totals 600 bytes for 34 unique 12 byte vertices and 64 faces consisting of three one byte indices.

4.5.2 Method Comparison

Due to the lack of error analysis from existing methods, it is difficult to compare our method to those from the literature and other spherical point distributions. Here, we attempt a comparison of our method with those of Oliveria and Buxton [71] (PNORMS), the MPEG-4 3D Mesh Coding [1, 98] (3DMC), Botsch et al. [9] (Octahedron), Deering [22] (Deering), the MPEG-4 BInary Format for Scenes [2, 82] (BIFS), Isenburg and Snoeyink [51] (Projection), and Ahn et al. [3] (Cube). We also include a comparison with the point distributions generated by fixed precision spherical coordinates and by the HEALPix method [31]. For the PNORMS method, we used an icosahedron as a base polyhedron, and, for our method, we used the subdivision method on Spherical Covering 1 (Figure 4.12, Table 4.1). Figure 4.13 illustrates the spherical point distribution generated by each method at 10-bit precision.

In our test, we used each method to generate the set of all unique quantized normals at 16-bit precision. Using QHull [7], we then converted this point set into a triangulated, convex polyhedron. Taking advantage of Theorem 1, we are able to use this polyhedron to

Method	Unique Normals	ε_{max}
Our Subdivision Method	61,442	0.561°
BIFS [2]	64, 896	0.612°
HEALPix [31]	49,154	0.682°
Cube [3]	64,896	0.779°
Spherical Coordinates	65,026	0.787°
Deering [22]	24,578	1.26°
Octahedron [9]	32,768	1.26°
3DMC [1]	32,258	1.27°
PNORMS [71]	27,200	1.36°
Projection [51]	41,712	1.38°

Table 4.3: Number of unique normals generated and ε_{max} for each method (Figure 4.13) at 16-bit precision.

analytically determine an upper bound on the error for quantizing normals based on that set of points. Note that this error bound assumes that the actual normal quantization process will not exceed the error bound from optimal quantization. The actual error upper bound is likely to be higher since the methods offer no such guarantees.

Table 4.3 contains the results of our comparisons. In general, the methods that had a low error bound generated near the maximum number of unique normals and produced relatively "uniform" distributions on the unit sphere. The error bounds from our subdivision method were consistently lower than those of the other methods (Tables 4.1 and 4.2). Interestingly, the method proposed by Deering, the Octahedron method, the 3DMC method and even the PNORMS method all have a higher error bound than using fixed precision spherical coordinates. Our barycentric method also has a lower bound than fixed precision spherical coordinates, but its error bound is higher than that of BIFS and HEALPix.

4.6 Conclusions and Future Work

We presented two methods for lossy normal vector compression through quantization based on refining base polyhedra. The first revises the existing subdivision methods, using a table of normals comprised of the refined polyhedra vertices. The second method quantizes normals by computing fixed-precision barycentric coordinates within base polyhedron faces. We provided fast compression and decompression algorithms with low memory requirements for both methods, and we tested their performance on various known models.

We used the property that our quantized normals are vertices of refined polyhedra to introduce three results. First, we showed an analytical upper bound on error for a normal vector optimally quantized using a convex, triangulated polyhedron. Second, we showed that this error bound also holds for quantization using our methods. Third, we showed that, if all the faces of the base polyhedron and refined polyhedra are acute, then our methods will optimally quantize normals.

We performed several comparisons with the error bounds we derived. First, we analyzed

our methods using several base polyhedra. We found that base polyhedra derived from spherical coverings from Sloane et al. [90] gave the lowest error bound. We also found that the subdivision method gives a lower error bound than the barycentric method at the same bit precision. Thus, when a normal table can reasonably be kept in memory, the subdivision method is preferable. Next, we were able to compute the error upper bound, assuming optimal quantization, for a variety of methods from the literature at 16 bit precision. We showed that our subdivision method had the lowest error bound out of the methods we tested, and that several existing methods had higher error bounds than using fixed precision spherical coordinates. Further, our subdivision method was the only method, at 16-bit precision, to satisfy Deering's criterion that the error be at most 0.01 radians [22].

In the future, there are a variety of objectives we would like to meet. We would like to refine the compression process so that base polyhedra with more faces can be used. We plan to implement GPU versions of the decompression algorithms. We also plan to extend this work for use on arbitrary vectors. Lastly, we would like to be able to further formalize the error bounds so that we analyze base polyhedra more quickly.

Acknowledgements

We would like to thank the Netherlands Organization for Scientific Research (NWO) for providing project funding.

The Bunny, Happy Buddha, Armadillo and Lucy models are from the Stanford 3D Scanning Repository. The David model is from the Digital Michaelangelo project. The Phlegmatic Dragon is from the Academy of Sciences of the Czech Republic and the Czech Technical University in Prague.

CHAPTER 5

Interactive Particle Tracing for Visualizing Large, Time-Varying Flow Fields

Abstract

Particle tracing is a classical method of flow field visualization. For interactive exploration, particles must be advected and displayed in real-time. Graphics Processor Unit (GPU) based techniques can advect hundreds of thousands or millions of particles in real-time. We have investigated such GPU-based techniques for interactive exploration of large, time-varying flow fields. Our approach can be roughly divided into three categories: data preprocessing, visualization and interaction. The preprocessing involves data compression, region of interest computation and preparation of multi-resolution data. For flow visualization, we use use the GPU for both data decompression and particle advection. More than 1,000,000 particles can be visualized at interactive frame rates and data rates. We support the standard particle visualization techniques of pathlines, streamlines and streaklines. We also represent particles as flow-oriented ellipsoids, which can additionally be moved over their traversed pathlines to explore their behavior in time. Dynamic features in the data are explored by interactively seeding and tracking particles through time in both a standard display screen and a stereoscopic virtual environment. Further, we have validated our particle system by comparing its particle trajectories with those generated by a Large-eddy Simulation.

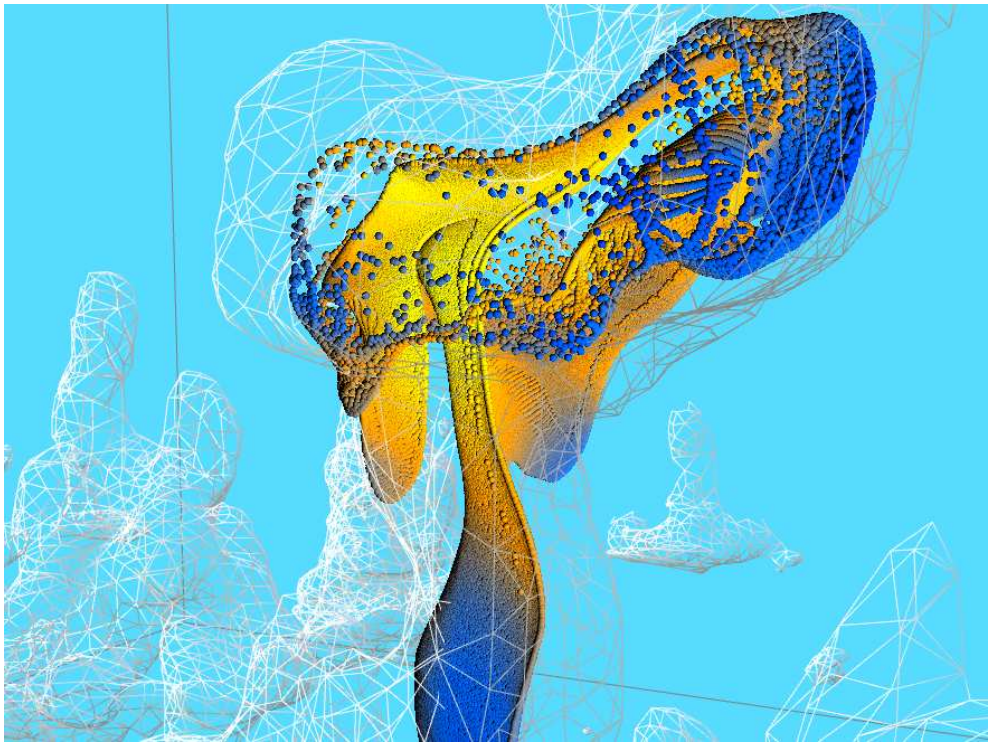


Figure 5.1: Over 1,000,000 particles in a cumulus cloud being advected in a time-varying velocity field at interactive frame rates.

5.1 Introduction

Computational Fluid Dynamics (CFD) techniques such as Large-Eddy Simulation (LES) or Direct Numerical Simulation (DNS) can produce very large, time-varying, multi-field data sets. Exploration and analysis of these data sets is difficult due to their size, complexity and time-varying nature. Various techniques have been developed over the years to explore different aspects of the data. One popular technique for studying the fluid flow characteristics, particle tracing, has recently been extended to handle time-varying flow data interactively with the help of the Graphics Processing Unit (GPU) ([83], [84], and [13] , Figure 5.1).

One of the main challenges to extending particle tracing to time-varying data has been the sheer size of the data. Interactive exploration requires loading data time steps at interactive rates in addition to interactive frame rates for the particle tracing algorithm. To ensure interactive frame rates, particle tracing algorithms are executed on the GPU. In order to maintain an interactive data rate, new data must be continuously transferred from a source, such as disk or a network computer, to the CPU and ultimately to the GPU. Both of these transfers represent potential bottlenecks, and, to overcome them, data reduction and filtering techniques must be applied.

We present a system that supports interactive exploration of large, time-varying flow fields. With the help of the GPU, we are able to advect more than 1,000,000 particles at interactive frame rates and data rates, see Figure 5.1. The system is targeted at our primary source of data: LES data on a staggered, Cartesian grid with relatively compact features. The system includes both a preprocessing phase and an interactive particle tracing engine. The preprocessing step downsamples and quantizes the flow fields for each time step, and it identifies, extracts, and quantizes full resolution regions-of-interest (ROIs) around features in the data. The preprocessed data is loaded from disk and streamed to the GPU, where it is used by the particle tracing engine. The engine supports a variety of data formats, integration schemes and visual representations of the particles. The particle tracing engine has been integrated into both a stand-alone desktop application and a stand-alone Virtual Reality (VR) application, and it has been integrated into our existing Cloud Explorer application [36]. Further, we have validated our particle system by comparing its particle trajectories with those generated by the Large-eddy Simulation system.

In addition to the standard features, we have added new functionality to our particle tracing engine to aid in the interactive exploration of time-varying data. The input data has been quantized to help overcome transmission bottlenecks, and the engine decompresses it on the GPU to generate the actual flow fields used for particle advection. Since we are primarily interested in how the particles move in and around features of interest in the data, we use downsampled data for particle advection in the full domain, and, when an interesting feature has been identified, the engine performs multi-resolution particle advection, using full resolution data for particles around the feature. To give a clearer idea of the instantaneous flow characteristics, we generate ellipsoid glyph representations for the particles on the GPU. Further, without storing extra information, we are able to interactively move the ellipsoids along their pathlines to more closely study their behavior in time.

The remainder of this paper is organized as follows. We first provide an overview of

related work in Section 5.2. In Section 5.3, we provide a top-down view of our system, after which we discuss the type of data we are using and how we prepare it for interactive exploration in Section 5.4. Then we describe our GPU-based particle system in Section 5.5. This includes the GPU-based data decompression and particle advection as well as the GPU-based visualization tools. Section 5.6 focuses on the interaction using a virtual environment for exploration of the LES data. We show our results in Section 5.7 and end our paper with conclusions and suggestions for future work in Section 5.8.

5.2 Related Work

Particle tracing in time-varying data has long been a popular topic for visualization research. Lane [60] introduced one early approach. However, the ability to interactively advect and view large numbers of particles in time-varying data has not been possible until recently.

In the last few years, the very high bandwidth, processing power and parallel architecture of the GPU have been exploited to be used for other purposes than just graphics rendering [73]. The suitability of the GPU for processing large data in parallel makes it a good candidate for algorithms such as particle advection.

Particle advection algorithms that exploit the processing power of the GPU are described in [56], [59], and [57]. These initial algorithms worked only with a stationary flow field (i.e. non-time-varying). While interesting, performing the particle advection on the GPU is only one aspect of dealing with time-varying data. These methods lack the necessary framework to deal with such data.

More recently, these GPU particle tracing techniques have been extended to work with time-varying data by Schirski et al. [83] and [84] and Bürger et al. [13]. Neither of these solutions addresses the problem of dealing with large time-varying data in its entirety, though. Schirski et al. use a high performance computing (HPC) back-end to generate a series of velocity fields on a Cartesian grid based on a region-of-interest for GPU-based visualization. However, their approach restricts particle advection to the region-of-interest and it lacks out-of-core support, requiring all velocity fields to fit in RAM on the computers used for visualization. The work by Bürger et al. represents a truly out-of-core approach that works on standard desktop computers, but their approach does not employ any data filtering or reduction techniques. According to their hardware specifications, it takes 3.5 seconds to load the velocity field for each time step from their example data set. Our approach, on the other hand, is both out-of-core and incorporates data handling to help ensure interactive performance.

Other work similar to our ellipsoid glyphs also exists. Max, Crawfis and Grant [66] generate motion-blurred, semi-transpartent elliptical spots. Van Wijk [108] introduced particles used to approximate stream surfaces, but these particles are flat. The most similar technique to our approach is using glyph atlases for representing pseudo-3D shapes as in [59] and [57]. Unlike these techniques, we make use of the GPU to accurately draw the ellipsoids as 3D objects from any viewing direction.

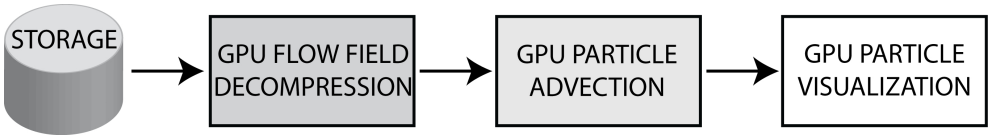


Figure 5.2: The different stages of the GPU-based particle tracing pipeline. First, the compressed velocity field for the current time step is transferred from a storage device to the GPU where it is decompressed on the fly. Next, the decompressed velocity field is used as input for the GPU-based particle advection. Finally, the updated particles are drawn to screen in an interactive application.

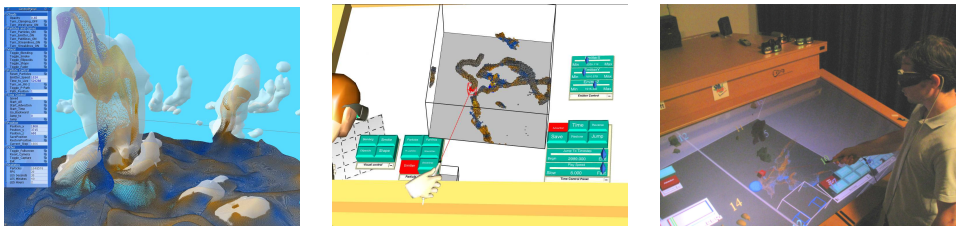


Figure 5.3: Our particle tracing engine in three applications. Left: Our desktop particle tracing application. Center: Our stand alone VR particle tracing application. Right: Our CloudExplorer application running on the Virtual Workbench.

5.3 System Overview

Our particle tracing system works with time-varying, velocity fields. We concentrate on data where the features of interest are defined by one of the scalar variables in the data. This is in contrast to many fluid flow applications, where the features are defined by patterns in the flow field itself, such as vortices and shock waves [76]. We are, therefore, interested in how the features and the flow field interact with and influence each other. The dynamic nature of both the fluid flow and the features makes this a challenge.

Figure 5.2 represents our approach to supporting interactive particle tracing for time-varying flow fields. In order to maintain interactive frame rates, we rely on the GPU for particle advection. In order to maintain interactive data rates, there are two major bottlenecks we must overcome. The first is transferring data from disk to the CPU, and the second is transferring the data from the CPU to the GPU. Our strategy for dealing with this is to reduce the size of the data to load from disk through subsampling, region-of-interest extraction, and data compression. Through our use of vector quantization for compression, we are able to decompress the velocity fields directly on the GPU, which helps with the second bottleneck. To enable interaction with the data, we have integrated the particle tracing engine into three applications (Figure 5.3).

5.4 Data Handling

We primarily work with data generated by Large-eddy Simulations of cumulus clouds. The features we typically work with are the clouds in the data set, which are generally compact, i.e. are bounded by a box with volume less than $\frac{1}{64}$th of the domain in size. The data output by the simulations is arranged in a "staggered" grid, or Arakawa C grid [5]. We work directly with the staggered grid as converting to a standard Cartesian grid would result in smoothing and information loss. We have tested the system on data sets with resolutions of 64x64x40, 64x64x80, 128x128x80 and 256x256x220 and between 600 and more than 3000 time steps. Each time step generally consists of 1 to 3 scalar variables and 1 vector variable. The vector variable represents the velocity. Each scalar and each vector component are 16-bit short integers. The total size of the velocity field for one time step at these resolutions is, respectively, 0.94 MB, 1.9 MB, 7.5 MB, and 82.5 MB.

5.4.1 Data Preprocessing

In order to interactively load and stream new data time steps to our particle tracing engine, we must ensure that the amount of data to be loaded for each time step remains reasonably small. For example, if the hard disk can read 30 megabytes per second, then each time step should not have more than 3 megabytes of data to ensure a data rate of 10 time steps per second. In order to ensure that the data size per time step remains small, we first preprocess the raw simulation output.

The first step in the preprocessing is to perform feature tracking to locate all the features in the data. In the data we typically use, this involves a 4D connected component labeling algorithm [36]. Once all the features are found, sub-volumes around each feature in each time step for each variable are defined by inflating the feature's bounding box for that time step. These sub-volumes are the possible regions-of-interest (ROIs) in the data.

The second step in the preprocessing is to extract the ROIs and create subsampled versions of the velocity fields. We subsample by reducing the grid dimensions in each direction by a power of two and averaging the values from the full resolution cells contained in each low resolution cell. We use the subsampled volumes to perform less accurate particle tracing in the entire domain, and we use the ROIs to perform particle tracing in and around the features at the data's full resolution. By combining these two approaches, we are able to ensure that the input data size remains small, which allows us to maintain an interactive data rate during particle tracing, while providing sufficient detail in and around features of interest.

The last step of the preprocessing is to generate the input velocity fields. Our particle engine works with three input data formats: 16-bit integer output created from the subsampling and ROI extraction, 16-bit half-float data, and compressed data. The 16-bit integer is converted from signed to unsigned data for use on the GPU by adding 32,768 to it. This data is not interleaved since the individual components can be used for other parts of the visualization pipeline such as slicing planes. The half-float data is interleaved and can be treated as an RGB image. The half-float data only has 10 bits in the mantissa, but this is sufficient for

most of the data we work with. The compressed data uses 32 bits to represent each velocity vector.

Our data compression involves separately quantizing the length and direction of each vector in the velocity field. This results in two new scalar fields of 16-bit unsigned integers. We interleave these two fields together to represent each vector by one 32-bit unsigned integer. We use our earlier unit vector quantization algorithm [33] to quantize the vector directions with 16-bit precision. This method guarantees that the angular quantization error will not exceed 0.561°. We quantize the length of the vectors based on the range between the minimum and maximum vector length for the data set. With this compression scheme, we are able to achieve a compression ratio of 3:2 for our raw data. For raw data consisting of three 32-bit components, the compression ratio would be 3:1. While this compression ratio is relatively small, it enables an entire vector to fit into one standard, 8-bit RGBA pixel, which is convenient for use on the GPU.

5.4.2 Data Transfer

Transferring uncompressed velocity fields to the GPU is done by storing the velocity field for the current time step into one or more 3D textures. For uncompressed data, the separate 16-bit unsigned integer velocity components are stored in three separate 3D alpha textures. For the half-float data, each the x, y and z velocity components are stored in the red, green and blue color components, respectively of a 3D texture.

For our compressed velocity fields, we initially transfer a 2D texture containing a 256x256 look-up table to GPU memory. This look-up table holds up to 65,536 unit vectors, which can be accessed using two 8-bit texture coordinates [33]. These vectors are used as the set of quantized unit velocity vectors. For each time step, both the quantized lengths and direction indices of the instantaneous velocity field are transferred to the GPU using one interleaved RGBA 3D texture. The vector lengths are stored in the red and green components and the vector direction look-up coordinates are stored using the blue and alpha components.

While data must initially be loaded from disk, we maintain a cache of recently loaded time steps. If the data set is sufficiently small, then the entire data set will be cached in memory after one pass through all time steps. Once cached, the data set can be very rapidly sent to the GPU. If the entire data set does not fit into memory, then the cache functions as a “time window”. Performing advection within this window also benefits from the rapid transfer of data to the GPU.

5.5 GPU-based Visualization

We use the GPU for both the particle advection and the visualization of the particles and their trajectories. In addition we use a GPU-based decompression algorithm to decompress the flow data prior to or during our particle advection routine. In Figure 5.2, the different stages of the particle advection pipeline are shown.

5.5.1 GPU-based Data Decompression

Our compressed data helps to reduce transfer latency, and it can be rapidly decompressed on the GPU. Decompression can be performed on the fly by decompressing vectors on demand for particle advection, or it can be performed at once for an entire velocity field with off-screen rendering. If the off-screen rendering option is used, then the 3D texture generated by the rendering is used as the velocity field for particle advection. The choice as to which option to use depends on the size of the velocity field, the number of particles, and the number of texture lookups per particle per velocity field required for particle advection.

In the off-screen rendering, each slice of the velocity field is decompressed separately in a rendering pass. Each rendering pass renders a rectangle in a view port constructed to match the size of one slice of the velocity field texture. This enables a fragment program to execute the decompression algorithm. For each slice, each texel in the texture generated during rendering represents one vector in the velocity field.

The decompression algorithm takes two textures as input: an interleaved texture of quantized vector lengths and quantized vector directions, and a 256x256 vector direction look-up texture. The blue and alpha components in the interleaved texture are used to look up the direction of each vector in the velocity field. The direction vectors are then scaled according to the length given by the red and green components in the interleaved texture.

5.5.2 GPU-based Particle Advection

We have implemented a GPU algorithm for updating the particle positions in relation to the (decompressed) velocity field. Initially, the particle positions are stored in the color components of a texture. The x, y and z positions for each particle are stored in the separate RGB color components. The alpha component is used for storing a separate scalar value such as a color or particle life time value (which, in the case of a particle emitting seed point, indicates whether the particle is alive and should be visible, or that it has died and should not be drawn), similar to the particle system described in [56]. Two of these textures containing the particle positions are used alternately for input and output of the off-screen particle advection pass.

The velocity fields necessary for advection are stored as textures on the GPU. OpenGL treats texel values as being centered in the middle of the 3D cell that the texel represents. For particle advection with Cartesian grids, velocities at points in space can be found by a straightforward texture lookup. For staggered grids, each velocity component must be looked up independently by shifting the original point position one half-cell length in the negative component direction. For example, if the width of a cell is c_w, then to find the x component of the velocity field at point $p = (x_p, y_p, z_p)$, the texture value at $(x_p - \frac{c_w}{2}, y_p, z_p)$ must be retrieved. For the 16-bit integer and 16-bit half-float data formats, the texture lookup can make use of hardware supported trilinear interpolation. However, if the compressed velocity fields are being decompressed on the fly during particle advection, a software trilinear interpolation must be used.

During particle advection, a position update for each particle is calculated. We have im-

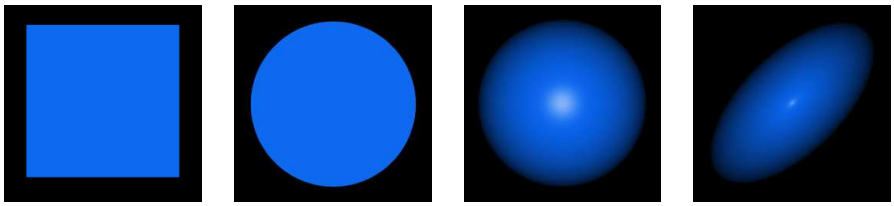


Figure 5.4: Creating a point sprite ellipsoid texture, from left to right. The point sprite is drawn as a square, which is then rounded. Lighting is added to create the appearance of a real 3D sphere. Finally, the sphere is stretched and oriented with respect to the velocity field.

plemented three possible integration schemes: Euler integration, second-order Runge-Kutta (RK2) integration and fourth-order Runge-Kutta integration (RK4). Furthermore, we can match the integration time step size to the data time step size or we can interpolate between data time steps for smaller integration time steps. When using any integration scheme other than Euler integration without time interpolation, two data time steps must be resident in GPU memory.

If multi-resolution mode is enabled, then velocity fields for the ROI must also be resident in GPU memory. Care must be taken when more than one time step is required for integration as the location and presence of the ROI is time-dependent. If a particle lies within a region of interest in one velocity field, then it is advected with the ROI velocity field. Otherwise, it is advected with the general velocity field. It should be noted that this will lead to velocity discontinuities as particles enter or leave the ROI, but, as we are primarily interested in their behavior within the ROI itself, this is unimportant to us.

The particle position update calculated during advection is stored in an intermediate velocity texture, which can then be used during visualization. This velocity texture is then used to write updated particle positions in the output texture. The updated particle positions are fetched from the texture in the vertex shader and used for visualizing the particles and their trajectories.

5.5.3 GPU-based Visualization Tools

In order to explore the time-varying flow using particles, we have added several GPU-based visualization tools. With these we are able to change the color and shape of the particles, visualize their trajectories and provide real-time interaction. We visualize particles in our system as separate solid objects, using flow curves, or as additively blended point sprites, to give the impression of a substance (smoke or liquid).

In order to make the solid particles look more realistic, we create a sphere shaped point sprite for each of the particles on the fly using a shader program. Adding both diffuse and specular lighting creates the impression of a real 3D object (Figure 5.4). This can all be done on the GPU in a fragment shader by having OpenGL apply texture coordinates to the point sprite. To enhance the sense of depth and to maintain the illusion that the point is a 3D object,

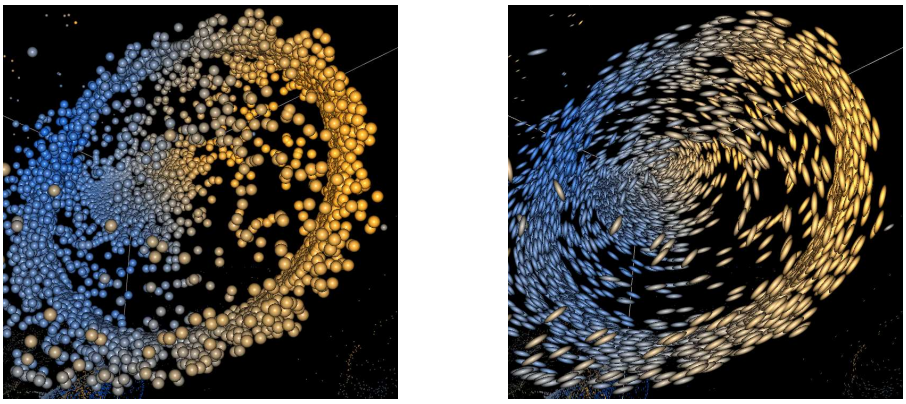


Figure 5.5: A comparison between the spherical point sprites (left) and the velocity-based view-oriented ellipsoids (right). Both shapes only require one vertex. The ellipsoids give a better impression of the instantaneous flow characteristics.

the point size of the particle is adjusted with respect to its distance from the viewpoint.

Although the moving solid spheres can give a good representation of the underlying flow, viewing the animated particles from different angles can be deceptive. Likewise, if the particle advection is halted, the spheres do not give any information about the local flow direction. As an alternative, glyphs can be used to improve perception of flow direction. We have chosen to use ellipsoid shaped glyphs to illustrate the flow direction (Figure 5.5).

Unlike using a glyph atlas as in [59] and [57], our glyphs are generated on the fly and are adjusted to the current viewing angle. The ellipsoid shape is created for each particle individually using a shader program (Figure 5.4). The viewing direction vector is sent to the shader program to construct a viewing plane. Next, to compensate for the viewing orientation, the velocity vector is fetched from the flow data and projected onto the viewing plane, resulting in the length, width and angle of the ellipsoid. In the same way as the impostor spheres, diffuse and specular lighting are added to enhance the realism of the ellipsoid shape. The result is a view-oriented ellipsoid glyph that indicates velocity direction and magnitude at the cost of only one vertex.

Our system is capable of rendering all of the standard flow curves: streamlines (Figure 5.6 left), streaklines (Figure 5.6 center), timelines (Figure 5.6 right) and pathlines (Figure 5.7). It can also render illuminated streaklines like those in [111]. This is done by saving the particle positions over a number of time steps in a large texture and connecting the corresponding particles with line segments, described in [13]. In addition, we have implemented functionality to move the particles along their paths, showing their position history over time.

The ellipsoid shaped particles-on-pathline give good insight on the movement and change of velocity of the particle over time in a certain region (Figure 5.7). Fast browsing of the particles over their pathline is possible since all the positions of the particles over the defined time interval are already stored in the graphics memory. To determine the ellipsoid shape

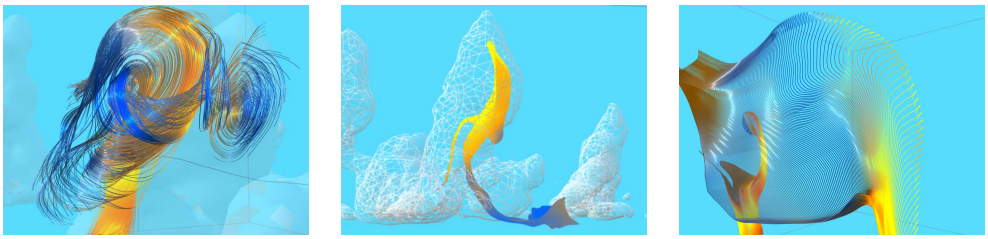


Figure 5.6: Left: Illuminated streamlines depict the underlying flow in a region in space at a certain position in time. Center: Streaklines simulate the injection of dye into the flow. Right: Timelines represent sets of points released at fixed points in time.

of each of the particles, we need the particle's velocity at that time step. Since the we do not have the velocity field data of all the time steps available, we determine the velocity by calculating the positional difference between two subsequent particles.

5.6 Interaction

In this section we describe the type of tools we have chosen for interaction with the advected particles within the domain of the time-varying flow field.

5.6.1 Particle Emitter

In order to explore the domain interactively, the ability to release particles at arbitrary positions is desirable. Our particle emitter is a box that can be moved around the domain in all three dimensions while emitting a large number of particles in the time-varying flow field. We assign each particle a time to live value so that it will die after a certain period of time and be reincarnated at the emitter. Thus, the particle emitter can function as an infinite source of particles. For our research, it is important to be able to move the particle emitter within the domain because of the dynamic nature of the features in our LES data. Moving the emitter allows us to search for and follow these features and observe how the particles behave in and around the features.

We have also added save and restore functionality to allow the repetition of certain observations. When exploring this kind of simulated data, it may be desirable to observe the behaviour of the particles over a certain time interval starting at a certain position and then repeating this experiment with identical or slightly shifted initial particle positions. If the starting point in space and time is exactly identical, then the particle advection will proceed exactly as before.

Researchers can perform experiments by placing the emitter at a certain place in time and space and observing the emitted particles. The shape of the particle emitter can be adjusted

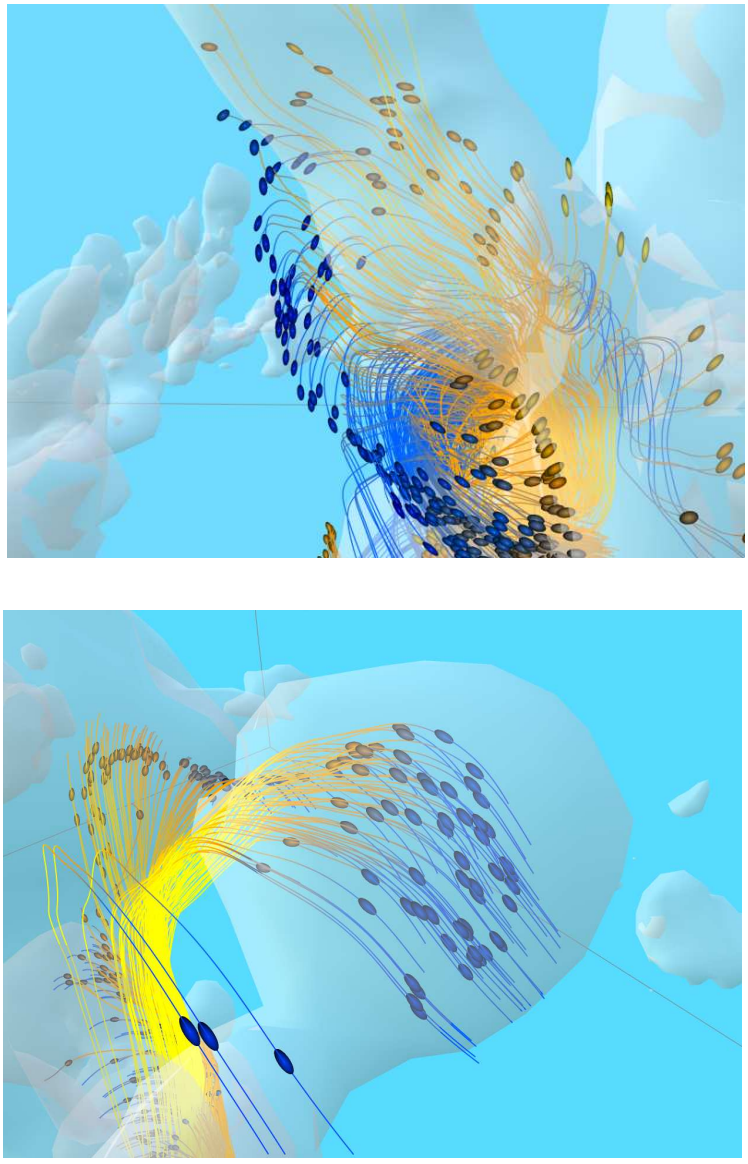


Figure 5.7: Ellipsoid particles shown on their pathlines. When browsing through time, the ellipsoids move along the pathline, and their shapes depict the magnitude and direction of the underlying flow. The feature geometry also changes to reflect the visible particle time step.

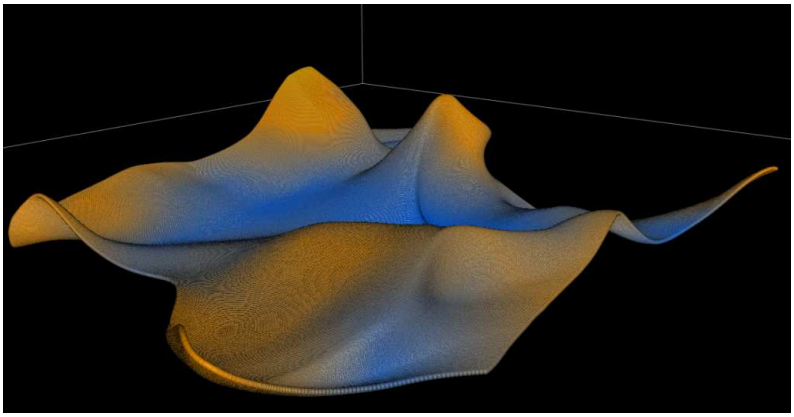


Figure 5.8: Plane of particles indicating the vertical flow over a wider region.

to suit different kinds of experiments. The emitter shape and the distribution of particles (random or regular) can be set. A small but very dense box of particles can be used for a smaller region-of-interest, while, in the case of a larger ROI, a large plane of particles can be used in order to maintain high particle density (compared to a larger box with low density). For example, the dense plane of particles can be positioned at a certain height where the movement of the particles in the plane indicates the vertical flow over a larger lateral region (Figure 5.8).

5.6.2 Multi-Resolution Data for Regions-of-Interest

While exploring the data, selecting features enables multi-resolution mode. In this mode, a full-resolution velocity field is used for particle advection inside an inflated bounding box that encloses the selected feature during each time step. This combination allows researchers to observe the general trend of the flow in the whole domain, while still being able to see the details of the flow where it is interesting: in and around the features in the data (Figure 5.9).

5.6.3 VR and Interaction

Using the particle engine in VR offers two advantages: improved depth perception and improved interaction. With the stereoscopic view, it is much easier to determine where the particles are in 3D than in a standard desktop environment. Using the Virtual Workbench, the particle emitter can be attached to a VR interaction tool such as a stylus. The stylus and the stereoscopic view of the Virtual Workbench provide a simple way to position the particle emitter while exploring the time-varying flow field. Likewise, it is possible to move the seeding point of the stream- and streaklines inside the domain during advection.

In addition to the stereoscopic view, depth fogging is added to the particles to enhance the

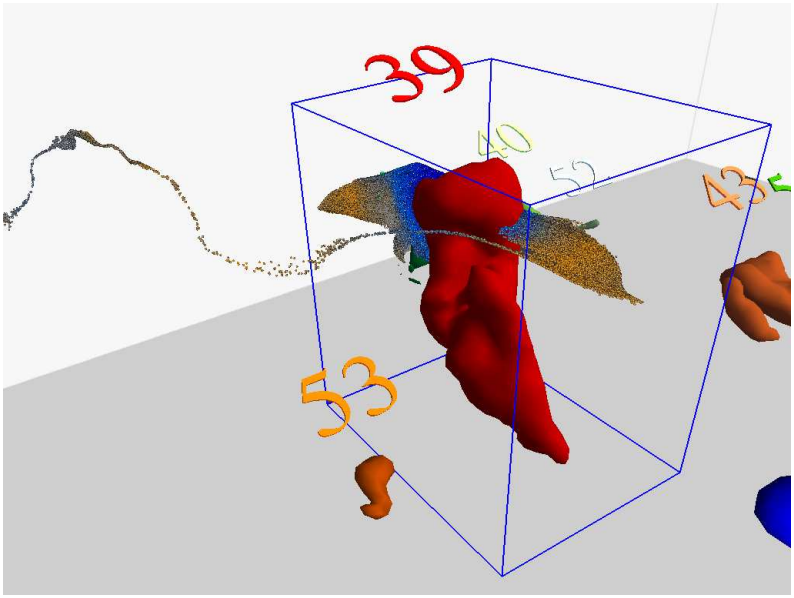


Figure 5.9: In multiresolution mode, particles inside the ROI are advected by a higher resolution velocity field. Outside the region particles are advected using a lower resolution velocity field.

sense of depth. The depth fogging is implemented by adjusting the intensity of the color with respect to the viewing distance. To avoid the sudden vanishing of the particles after they have exceeded their life time parameter, they slowly fade out by blending in with the background or having their transparency increased.

Our particle system has been implemented in a stand-alone desktop application allowing the user to explore the data while sitting at his desk (Figure 5.3 left). In addition, we have integrated the particle tracing into a standalone VR application supporting stereo vision and electro-magnetically tracked interaction tools (Figure 5.3 center). Finally, our particle engine has been integrated into our CloudExplorer application [36] to combine the particle visualization with dynamic geometric isosurfaces (Figure 5.3 right). We do not currently account for off-axis stereo when rendering point sprites as spheres or ellipsoids, but, in our experience, the effects are not very noticeable in normal use.

5.7 Results

5.7.1 System Performance

Interactive particle tracing using time-varying velocity fields requires interactive frame rates as well as interactive data rates. By data rate, we mean the rate at which new data time steps

	Euler		RK2		RK4	
Data Type	CG	SG	CG	SG	CG	SG
Short	106	106	37	37	16	16
Half	190	104	73	28	34	12
Compressed	67	27	19	6.3	7.9	1.8

Table 5.1: This table lists the performance of our advection integration schemes in millions of integrations per second. The tests were performed with 1024x1024 particles on staggered (SG) and Cartesian (CG) grids for each of three data formats: the original signed short integer data (Short), the half-float data (Half), and the compressed data (Compressed).

are used for particle advection. Our GPU-based particle advection algorithm provides an interactive frame rate for a very large number of particles, but, in order to have interactive data rates, the data needed for each time step should be as small as possible. Our data preprocessing combined with GPU-based data decompression helps keep the data rate interactive.

We have measured the performance of our standalone application with the integrated particle tracing in terms of both frame rate and data rate under a variety of conditions. We have performed the different tests on a desktop PC with an Intel Pentium Core 2 Duo 2.2 GHz processor, 2 GB of RAM and an NVIDIA GeForce 8800 GTX graphics card.

To test the performance of advection algorithms, we advected 1,048,576 particles using all three data types and both types of grid. Table 5.1 lists the results in terms of millions of integrations per second. Advection using half-float data on Cartesian grids performs the best. Advection using staggered grids is slower than advection on Cartesian grids for the half-float and compressed data because it requires extra texture lookups. For the short integer data, the velocity components are stored in separate textures so the number of texture lookups is the same regardless of the grid type.

To test the performance of the system as a whole, we tested the frame rate under several conditions. For these tests we used a data set of 3,000 time steps, which we downsampled to 64x64x40 per time step. Using each data type and grid type, we measured the frame rate for particle rendering and advection alone. We first tested the performance using 600 out of the 3,000 time steps, which we cached in memory. This gives an idea of the "in-core" performance. We also tested the performance on advecting particles in the full 3,000 time steps both syncing the integration time step with the data time step and having 6 integration time steps per data time step. This gives an idea of the "out-of-core" performance.

Table 5.2 lists the results from the tests when advecting 65,536 particles. All variations performed well when the data resided in main memory. However, when the data must be streamed in from disk to the GPU, the advantage of using the smaller, compressed data can be seen. We have also performed these tests with 1,048,576 particles. For the short integer data and the half-float data, the results remained the same for the streaming performance. For the compressed data, the cost of decompressing the compressed data becomes dominant. This can be alleviated by decompressing the entire velocity field at once. The short integer data performs worse than the half-float data largely because the data is stored in 3 files on disk instead of 1 file.

Data Type	Grid Type	No Time	Cached Data	1s Integration	6s Integration
Short	CG	691 fps	477 fps	241 fps	29 fps
Short	SG	668 fps	471 fps	233 fps	28 fps
Half	CG	817 fps	404 fps	240 fps	56 fps
Half	SG	664 fps	409 fps	226 fps	56 fps
Compressed	CG	617 fps	516 fps	275 fps	84 fps
Compressed	SG	294 fps	276 fps	173 fps	83 fps

Table 5.2: This table lists the results of our performance evaluations. We tested each of the three data formats: short integer data (Short), half-float data (Half) and compressed data (Compressed). We tested each format on both Cartesian (CG) and staggered (SG) grids. For each data and grid combination, we tested the performance of the advection alone (No Time), advecting data in 600 cached time steps (Cached Data), advecting particles in 3000 time steps using an integration step size of 1 second (1s Integration), and advecting particles in 3000 time steps using an integration step size of 6 seconds (6s Integration). The data was downsampled to 64x64x40 and each data time step is 6 seconds apart. The particles were advected with Euler integration. Results are given in frames per second. Except for the compressed data, the results were similar for 1,048,576 particles.

We have also tested the effects of regions-of-interest on the system performance. The impact is dependent on the size of the ROI. Most of the ROIs in our data are small, i.e. less than $\frac{1}{4}$ the size of the downsampled grid, so their effect is small. As the ROI size approaches the size of the subsampled domain, then the performances decreases by approximately 50%.

The effect of the various particle visualization options on performance is also relatively small in comparison to the base cost of rendering the particles. The more complex options, such as ellipsoids, have a somewhat higher impact, but it is not very significant.

5.7.2 System Validation

Validating the particle advection is important for several reasons. We must ensure that our algorithm is correctly implemented, and we must be able to demonstrate this to users of the system so that they can trust the visualization. Furthermore, since we make use of quantized data, we must ensure that the errors introduced in the quantization are not too significant. The results of the validation can also be used to improve the visualization by, for example, coloring or discarding particles that have likely deviated beyond a certain threshold.

To measure the accuracy of the GPU-based particle advection, we compare the positions of particles advected with our GPU particle advection with particles advected in an LES. For our validation tests, we use particles advected by a LES as the “ground truth”. The LES was performed on a grid size of 128x128x80 using time steps of 6 seconds. The particles were integrated using Euler integration with an integration step size of 6 seconds. For each time step, the LES wrote the velocity field and particle positions to disk. Inside the LES, this data is represented using 64-bit double precision floating point numbers. The output data is converted to 16 bit signed integers.

In our tests, we use velocity fields generated by the LES for our GPU advection. We seed particles in our GPU implementation at the same starting locations as the particles in the LES. We then advect the particles on the GPU using both Euler and RK4 integration using

Configuration	6s	60s	600s	1800s
Short Data, 64x64x40, Euler, 6s step	1.05m	7.06m	52.1m	163m
Short Data, 64x64x40, Euler, 1s step	1.05m	7.03m	51.6m	162m
Short Data, 64x64x40, RK4, 6s step	1.06m	7.10m	52.2m	163m
Short Data, 64x64x40, RK4, 1s step	1.05m	7.03m	51.6m	162m
Short Data, 128x128x80, Euler, 6s step	0.49m	1.38m	16.3m	74.3m
Short Data, 128x128x80, Euler, 1s step	0.47m	0.94m	11.8m	55.6m
Short Data, 128x128x80, RK4, 6s step	0.50m	1.53m	17.8m	80.1m
Short Data, 128x128x80, RK4, 1s step	0.48m	1.01m	12.4m	58.8m
Compressed Data, 64x64x40, Euler, 6s step	1.06m	7.15m	53.2m	167m
Compressed Data, 64x64x40, Euler, 1s step	1.06m	7.12m	52.7m	165m
Compressed Data, 64x64x40, RK4, 6s step	1.06m	7.18m	53.3m	167m
Compressed Data, 64x64x40, RK4, 1s step	1.06m	7.12m	52.7m	165m
Compressed Data, 128x128x80, Euler, 6s step	0.49m	1.51m	17.3m	74.7m
Compressed Data, 128x128x80, Euler, 1s step	0.48m	1.10m	14.0m	62.1m
Compressed Data, 128x128x80, RK4, 6s step	0.50m	1.66m	18.9m	81.5m
Compressed Data, 128x128x80, RK4, 1s step	0.48m	1.16m	14.5m	65.2m

Table 5.3: This table lists the results of our system validation tests. We compare the distances between particles advected within the LES itself at a resolution of 128x128x80 and particles advected with our GPU algorithm. The data time steps are 6 seconds apart. The distances are given in meters, and the size of our test domain in meters is 6400x6400x3200. The average particle velocity during the 300 time steps (1800 seconds) is 2.1 m/s. We give the results averaged over 1,310,720 particles after 6s, 60s, 600s and 1800s.

integration steps of both 6 seconds and 1 second. We then average the distance between the particle positions for the two systems after a certain number of time steps. We compare the accuracy of the particle trajectories for two different resolutions: the original 128x128x80 resolution and a subsampled 64x64x40 resolution. We also compare for both the compressed and short integer flow fields.

The results from our tests are in Table 5.3. The errors are listed in meters, and, for the 128x128x80 and 64x64x40 resolutions, one grid cell is 50x50x40 meters and 100x100x80 meters respectively. The average particle velocity for the LES particles was 2.1 m/s. The positions of the LES particles were rounded to the nearest meter before the comparison. The error for particles advected with the subsampled grid is higher than that of the full resolution grid, which is to be expected. After 300 time steps (1800 seconds) the particles remain, on average, within 2 and 4 grid cells of the LES particles in the 128x128x80 and 64x64x40 resolution data, respectively. After 100 time steps, the particles are, respectively, less than 1 and around 1 grid cell away from the LES particles. This is important since the features we typically study have a lifetime of approximately 300 time steps, and we are most interested in the behavior of particles during shorter time spans. From the results, we can also see that it can be advantageous to use smaller integration time steps but not especially so. This is likely due to the error introduced by interpolating in space dominating the error introduced by interpolating in time. Also of note is that our compressed data maintains nearly the same

Configuration	Time	50m	100m	150m	200m
Short Data, 64x64x40, Euler	6s	100.0%	100.0%	100.0%	100.0%
Short Data, 64x64x40, Euler	60s	99.65%	99.98%	100.0%	100.0%
Short Data, 64x64x40, Euler	600s	66.77%	89.51%	95.42%	97.35%
Short Data, 64x64x40, Euler	1800s	36.45%	54.90%	66.53%	75.30%
Short Data, 128x128x80, Euler	6s	100.0%	100.0%	100.0%	100.0%
Short Data, 128x128x80, Euler	60s	99.82%	99.98%	100.0%	100.0%
Short Data, 128x128x80, Euler	600s	97.18%	98.51%	98.89%	99.10%
Short Data, 128x128x80, Euler	1800s	70.42%	83.30%	89.37%	92.63%
Compressed Data, 64x64x40, Euler	6s	100.0%	100.0%	100.0%	100.0%
Compressed Data, 64x64x40, Euler	60s	99.65%	99.98%	100.0%	100.0%
Compressed Data, 64x64x40, Euler	600s	66.86%	89.59%	95.46%	97.37%
Compressed Data, 64x64x40, Euler	1800s	36.13%	53.85%	66.56%	75.33%
Compressed Data, 128x128x80, Euler	6s	100.0%	100.0%	100.0%	100.0%
Compressed Data, 128x128x80, Euler	60s	99.82%	99.98%	100.0%	100.0%
Compressed Data, 128x128x80, Euler	600s	97.39%	98.57%	98.92%	99.11%
Compressed Data, 128x128x80, Euler	1800s	73.25%	84.98%	90.39%	93.27%

Table 5.4: This table lists the percentage of particles with errors less than 50m, 100m, 150m and 200m after 6s, 60s, 600s and 1800s. All configurations use Euler integration and a time step of 6s. The original simulation grid cells are 50x50x40 meters.

accuracy as the short integer data.

Table 5.4 gives a more detailed analysis of the errors for four configurations. In each of these configurations, Euler integration with a time step of 6s was used. These results again show that the compressed data maintains nearly the same accuracy as the short integer data. This table also highlights the effects of using a subsampled velocity field for advection: after approximately 10 time steps, the accuracy begins to fall off markedly. However, in all cases, the majority of the particles, remain within about 4 simulation grid cells of the LES particles, with the particles advected using the full resolution data remaining even closer than those advected using lower resolution data. These results give us confidence that the GPU-advected particles are sufficiently accurate to give insight into the flow behavior around features of interest in the data during visualization.

5.8 Conclusions and Future Work

In this paper we have presented an approach for interactively visualizing large, time-varying flow fields. Our approach involves data preprocessing, real-time GPU-based flow field decompression, GPU-based particle advection and particle visualization. We have integrated our particle advection engine into three interactive applications: our CloudExplorer application, a standalone VR application, and a desktop application.

In our data preprocessing, we find and extract regions-of-interest around features in the data, create subsampled versions of the velocity fields, and compress the subsampled fields and ROIs through quantization. The reduced size of the compressed velocity fields allow for a faster data transfer from disk to the GPU, which enables us to stream the data at interactive rates.

Our GPU-based particle advection pipeline supports rendering and advecting more than 1,000,000 particles at interactive frame rates. Our pipeline supports GPU-based decompression of the quantized data, and it can use both a ROI and a subsampled velocity field simultaneously for multi-resolution particle advection.

Our different visualization tools support the exploration of time-varying flow fields in a virtual environment. These tools include the standard flow curves and our additions of ellipsoidal particles and the ability to move the particles over the traversed paths. The tools are available in both a standard desktop environment and two stereo VR environments.

In the future, we would like to make several improvements. We would like to better integrate the particle visualization into the research process through, for example, providing support for recording quantitative information about the particles. We would also like to further improve the accuracy of the advection itself. One possibility for this is incorporating the statistical subgrid model used in the LES for modeling subgrid scale turbulence. Additionally, we would like to further improve the application performance. One way to accomplish this is through optimization of our GPU implementation. We could also improve the data transfer rate by further compressing the data for storage on disk and performing an intermediate decompression step on the CPU.

Acknowledgements

We would like to thank René Molenaar and Jorik Blaas for their help with technical details. We would like to thank Harm Jonker and Thijs Heus for contributing the LES data. We would also like to thank the Netherlands Organisation for Scientific Research (NWO) for providing project funding.

CHAPTER 6

Cloud Explorer

Cloud Explorer is a cloud visualization system designed for visualizing cumulus clouds simulated in Large-Eddy Simulations. Chapters 2 and 3 describe portions of the system. Cloud Explorer also incorporates the particle tracing system described in Chapter 5 and makes use of the quantization techniques presented in Chapter 4. This chapter completes the picture by giving a high level overview of the complete system, presenting some components not described elsewhere, and discussing how Cloud Explorer was put to use by atmospheric scientists.

6.1 System Overview

Cloud Explorer has many components. The visualization itself is part of a larger framework for working with Large-Eddy Simulation data. Figure 6.1 shows the various system components. The two main components are data processing and data visualization. Data processing transforms the raw simulation data into a data format suitable for visualization. Data visualization encompasses both a visualization environment and a set of visualization tools available within that environment. Within each of these components there are a number of subcomponents.

6.1.1 Data Processing

One aspect of the data processing is feature processing. This involves detecting and tracking the clouds in the simulation data. Once the clouds have been identified, isosurfaces are

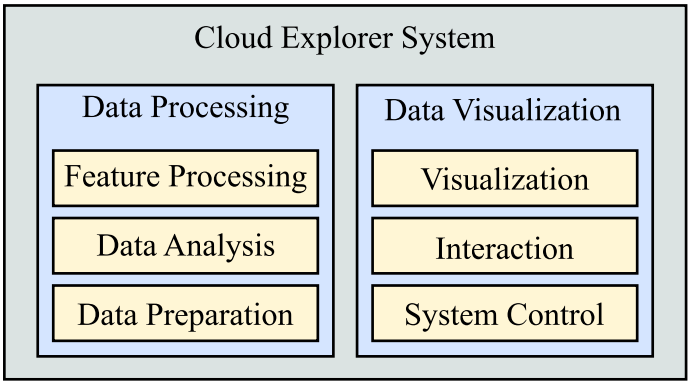


Figure 6.1: Overview of the components of the Cloud Explorer visualization system.

extracted. Next, the isosurfaces are smoothed, and the geometry is compressed using quantization. Additionally, during feature processing, axis-aligned bounding boxes are computed around each cloud in each time step for later use as regions-of-interest.

Another aspect of the data processing is preparing the data for use with particle tracing and slicing planes. Cloud Explorer works with downsampled data in the full domain and full resolution data around clouds of interest. The data processing generates the downsampled data, and it also extracts the full resolution regions-of-interest around clouds for variables that the user specifies. When preparing data for particle tracing, the user can also choose to compress the vector fields using quantization, which can give better performance when the data is read from disk.

The third aspect of data processing is generating derived data and computing statistical plots. As described in Chapter 3, the user can supply expressions that are evaluated to derive scalar or vector quantities from the data. These are computed on a per-feature basis using the regions-of-interest around the clouds. The results of the computations are stored for use in the plot windows in Cloud Explorer. Users can run the data processing multiple times with different expressions to compute new statistical data for use in Cloud Explorer.

6.1.2 Visualization

Cloud Explorer supports several types of visualization. The data can be explored in time by browsing through the data time steps. The scalar simulation variables can be visualized using isosurfaces (the clouds) and slicing planes. The vector variables can be visualized using particle tracing. Contextual information about the clouds is presented with statistical plots and "decorations" on the visualization.

The user can interact with the data both in space and time. Using a stylus, the user can select and manipulate the clouds, and he can interact with the various widgets in the user interface. Using a plexiglass panel, the user can probe the data domain directly. With the

time control window, the user can control the direction and speed of playback through time, and he can also browse freely through the data time steps.

Cloud Explorer gives the user a wide variety of visualization options. The user can:

- enable or disable particle tracing,
- enable or disable the slicing plane,
- choose which variables to visualize using the slicing plane,
- select and edit a color map to use for the slicing plane,
- hide and show different sets of clouds in the data domain,
- zoom in and out on clouds of interest,
- change the colors of the clouds,
- choose the which plots to display in each plot,
- highlight different time steps or altitudes in the plot windows,
- scale the plots in the plot windows,
- choose color maps for vector plots, and
- choose the interaction technique he would like to use in the environment.

6.2 User Interface and Interaction

In Cloud Explorer, user interface tasks are divided between interacting with the data, gaining additional information about the data, and system control tasks. Data interaction tasks include selecting and manipulating clouds of interest, positioning the particle emitter, probing the data with a slicing plane, and manipulating the data domain. Information gathering tasks involve showing additional information in the visualization to better understand cloud properties in time or space. System control tasks include controlling time step playback and enabling and disabling different visualization modes. Cloud Explorer employs a hybrid interface to support interaction and system control, and it adds “decorations” to objects in the VE to provide additional contextual information.

6.2.1 Hybrid Interface

Cloud Explorer’s user interface integrates 3D interaction techniques with more traditional 2D interface elements into the virtual environment. The input devices in the VE are a stylus and a tracked, plexiglass panel. Users select objects in the environment with the stylus using either ray casting or IntenSelect, a time-dependent, cone-based selection technique that makes

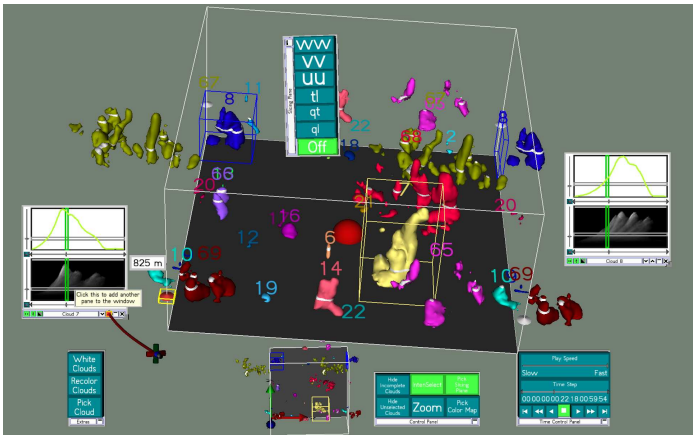


Figure 6.2: An overview of the Cloud Explorer user interface.

selecting smaller objects such as UI widgets easier [39]. Manipulation is done by fixing the distance and orientation of an object relative to the stylus when the user selects it with the stylus button and then keeping this relationship constant as the user moves the stylus.

The user interface is set up with the clouds in the middle of the VE with informational and system control widgets layed out around them. System control widgets are placed in 2D windows, which behave similar to their traditional counterparts. There are also 2D windows, which show statistical plots for the clouds the user has selected. Additional information about widgets in the user interface is provided to the user with tooltips. See Figure 6.2.

The combination of windows, tooltips, and IntenSelect provides for a flexible and compact hybrid interface. By using tooltips, labels on widgets in the UI can be shortened or replaced with icons, which, in turn, allows the widgets to be smaller. Using IntenSelect for selection and manipulation makes it easier for users to interact with small widgets or with widgets that are farther away. Using 2D windows allows for logical groupings of related widgets, which the user can move, show and hide as necessary. De Haan et al. [37] provides a more detailed discussion, but, using these techniques allows for more options to be comfortably presented to the user and for more space in the VE to be reserved for the 3D visualization.

6.2.2 Contextual Information

In Cloud Explorer's virtual environment, it's important for the atmospheric scientists to understand the relations between the different aspects of the visualization. In particular, the relations between time, space, and the current 3D view are important to understand. Cloud Explorer uses two techniques to help make these relationships clear: linked interaction widgets and visual feedback in the form of "decorations".

The linked interaction widgets are sets of widgets that provide the same functionality in different locations. For controlling time, there is a window with a video player control for moving forward and backwards through time, but, in each statistical plot window, there is also a slider that the user can use to browse through time. Similarly, there is a widget on the domain box to measure altitude that is also duplicated on the plot windows. This duplication lets the user adjust visualization parameters without having to shift focus away from the current cloud or plot window.

The decorations are additions that provide visual cues about visualization parameters. In the domain box, a white band is drawn around the clouds at an altitude specified by the user. On the plot windows, a vertical green line indicates the current time step, and a horizontal gray line corresponds with the altitude the user has specified. See Figure 6.3, top. These decorations also come in the form of small pop-up windows. For example, when setting the initial altitude of a plane of particles, a window appears showing the altitude in meters (Figure 6.3, right). The third example (Figure 6.3, center) draws only contour lines for the clouds above the slicing plane. This reveals the scalar values inside the clouds while still giving the user an idea of where the cloud itself is.

6.3 Data Handling

Several techniques for visualizing data that does not fit into main memory are described in the literature. Silva et al. [88] discuss several including geometry simplification, speculative prefetching of data, replacing geometry with imposters, and different forms of culling. Other techniques include using clever data structures or time windows (see e.g. Vrolijk and Post [102]). The common factor among many of these approaches is that they seek to overcome data transfer bottlenecks by reducing the amount of data needed in memory for interactive visualization.

In fact, one of the main challenges when dealing with large, time-dependent data like the LES data is efficiently transferring data from disk to the GPU. There are two important bottlenecks in the transfer process. The first bottleneck is disk transfer speed, which imposes strict limits on the amount of data that can be moved from disk to main memory in a given amount of time. The transfer speed between main memory and the GPU is the second bottleneck. When the visualization data fits into main memory, then the second bottleneck is usually the limiting factor on performance. For out-of-core visualization, then disk transfer speed is the most important bottleneck. When the data is transformed in main memory, e.g. decompressed, then the CPU can also become a performance bottleneck.

6.3.1 Data Cache

In Cloud Explorer, one method employed to mitigate the effects of the disk transfer speed bottleneck is the use of a data cache. The cache is a large segment of main memory that Cloud Explorer manages. As data is loaded from disk, it is stored in the cache. When Cloud

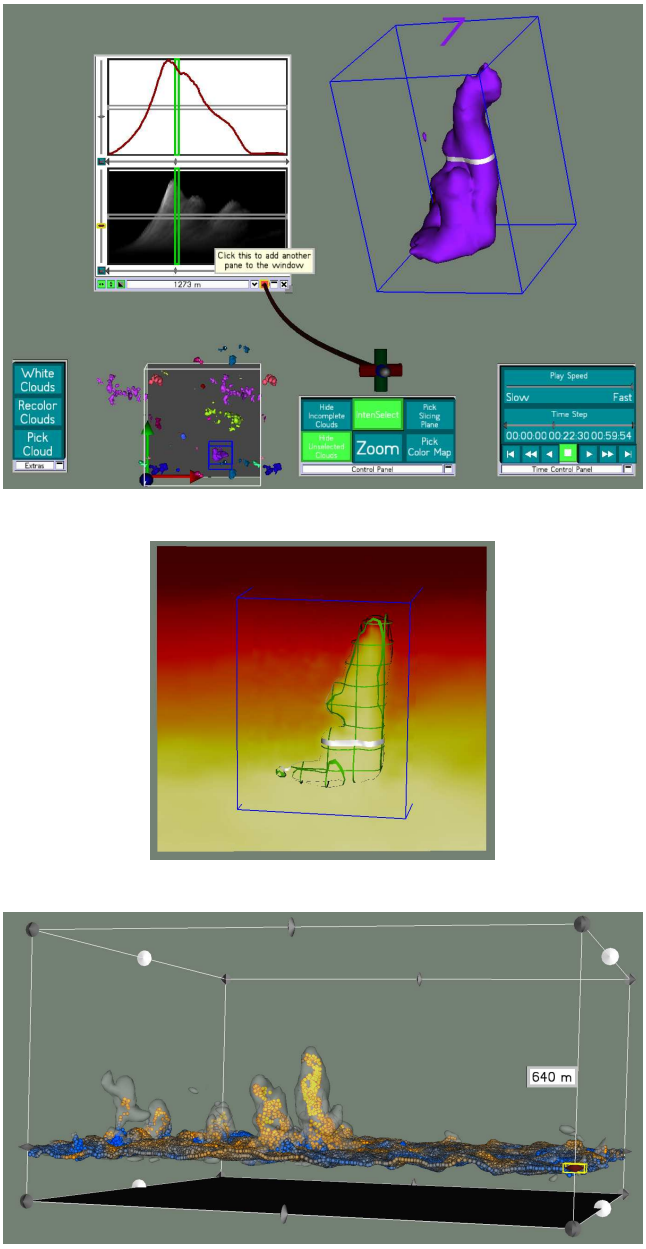


Figure 6.3: Portions of the Cloud Explorer user interface. Top: Displaying additional contextual information relates the selected cloud to plots of its behavior over time, the current time step with the x axes of the plots, and the highlighted altitude on the cloud with the y axes of the plots. Center: Drawing the cloud using contour lines shows the scalar values on the slicing plane inside of the cloud. Bottom: The tooltip displays the altitude of the initial plane of particles.

Explorer needs data for visualization, it first tries to retrieve the data from the cache. If the data is not in the cache, then the data is loaded from disk.

The main memory segment used by the cache is divided into a set of pages. Each page is sufficiently large to hold the largest single piece of visualization data, which for example might be an isosurface or a region of interest in a scalar field. Whenever data is read from a cache page, it is marked as the most recently used cache page and moved to the front of a linked list of cache pages. When data is loaded from disk, Cloud Explorer attempts to store the data on the most recently used cache page. If there is insufficient space in the page, then one of two things happens. If the cache is not yet full, a new cache page is used. If the cache is full, the least recently used page is emptied, and the data is stored in it.

When using the data cache, loading data from disk remains relatively slow, but retrieving data that is already in the cache is comparatively very fast. When visualizing a data set that can fit entirely into main memory, the entire data set will be in the cache after the user has browsed through the data once. Any further exploration will then be much more interactive. When visualizing a data set that does not fit entirely into main memory, the data cache acts as a sort of time window. If the user browses forward through time and sees something interesting in the data, the recent data time steps will be in the cache allowing him to more quickly browse back and forth to study it further.

6.3.2 Data Compression

Another method that Cloud Explorer employs to overcome transfer bottlenecks is data compression. In particular, Cloud Explorer quantizes vector field data and isosurface geometry data during preprocessing.

Isosurface coordinate positions are quantized using three 16-bit integers. With the vector quantization algorithm described in Chapter 4, the surface normal vectors are quantized using one 16-bit integer. The isosurfaces themselves are saved as indexed triangle strips. The list of triangle strip lengths and the list of triangle strip vertex/normal indices are quantized using the smallest unsigned integer type allowed given, respectively, the length of the longest triangle strip and the number of vertices/normals. This approach compresses the vertex/normal information to $\frac{1}{3}$ its size when represented with six 32-bit floating point numbers. It compresses the length and index information by 25% or 50%.

The vector field data is also quantized using the algorithm in Chapter 4. However, in addition to the 16-bit integer quantization of the vector direction, the length of each vector is also quantized using a 16-bit integer. Thus a vector can be represented by two 16-bits integers instead of three 16-bit or 32-bit numbers. While the quantization introduces error, comparisons of particle paths traced using the original vector data and the quantized vector data do not differ very much (See Section 5.7.2).

When Cloud Explorer loads quantized data, the data is stored as-is in the data cache. The data is then sent “as is” to the GPU for rendering or particle tracing. The quantized vectors are decompressed on the GPU by using a texture look up to retrieve the vector direction. The quantized vertex positions are “decompressed” by multiplying them by a constant scaling factor.

6.4 Virtual Reality and Desktop Visualization

Cloud Explorer runs on a variety of Virtual Reality (VR) systems: a responsive workbench (described by Koutek [58]), a PDRIVE system (described by De Haan et al. [38]) and a powerwall system. Figure 6.4 shows the workbench and a PDRIVE. Both the workbench and PDRIVE provide users with a stylus and a plexiglass pad for directly interacting with the data. The powerwall, intended primarily for presentations, offers a stereo view of the data, but provides no direct interaction. Cloud Explorer also runs on standard desktop computers, which also provide no direct interaction with the data. Instead, on these systems, interaction is done with the mouse and keyboard.

Support for mouse and keyboard interaction is provided through a set of keyboard shortcuts, which let the user choose between using the mouse to manipulate the view point and the world. When manipulating the world, the mouse cursor casts an invisible ray into the scene perpendicular to the screen, which allows the user to interact with objects, such as buttons and isosurfaces, under the mouse cursor. This makes common 2D tasks such as clicking on buttons, dragging sliders, and moving objects in the screen plane simple and intuitive, while still also allowing more general 3D interaction.

While Cloud Explorer was initially intended to be a purely VR visualization environment, the utility of having it be a dual VR and desktop visualization environment became increasingly clear. When visualizing the data, the added value that VR provides lies mainly in increased spatial perception of the data and simplified interaction with the data. These advantages are beneficial, but they require using a VR system. For the atmospheric scientists, this requires planning ahead and going to the VR laboratory with suitably prepared data. This added inconvenience makes it less likely that they will use the VR software for casual exploration and experimentation. Letting Cloud Explorer also work on the desktop gave the researchers more freedom to use it at their convenience. It also aided in the development process and made it easier to control the application while giving demonstrations on the powerwall.

In practice, each system was found to have its own advantages. The workbench offered the most immersive experience and could be comfortably used by two to three people simultaneously with one person controlling the visualization and the others observing. The powerwall proved most effective at giving stereo presentations to larger groups (six to eight people). It was also surprisingly effective for particle tracing visualization due to several factors: the particles greatly benefited from stereo visualization, the keyboard gave easy control over the wide variety of visualization options, and the users did not have to interact overly much with the data. The standard desktop environment was most often used during development due to it being both readily available and not requiring the use of extra devices (trackers, projectors, stereo glasses, etc.). The PDRIVE offered the best trade-off between the workbench and the desktop environment: a mouse, keyboard, and development terminal were within easy reach while still offering a full virtual environment with tracked input devices.

Figure 6.4: Top: Illustration of Cloud Explorer running on the responsive workbench. Bottom: A user at a PDRIVE.

6.5 Collaboration with Atmospheric Scientists

Cloud Explorer was developed in close collaboration with atmospheric scientists. It served as a test-bed environment for the visualization tools created for analyzing the Large-Eddy Simulation data generated by the scientists. The visualization tools were then used to help answer important questions about cumulus cloud dynamics.

6.5.1 Developing Cloud Explorer

The atmospheric scientists played an integral role in developing Cloud Explorer. Perhaps most importantly of all, they supplied the data to be visualized. The difficulties they had in processing and visualizing the large, complex, and time-varying data motivated automating the processing and providing interactive visualization. This led to the development of the data processing and data handling portions of Cloud Explorer. In addition to providing the data and its challenges, they also provided direction for developing visualization tools in order to study specific aspects of cloud behavior.

In the early stages of their research, they were focusing on cloud life cycle studies. These studies required the feature detection and feature tracking portions of the data processing. When studying individual clouds, they would use their own software packages to make several statistical plots of different cloud properties over time. It became clear that it would be helpful for them to be able to compute and visualize these types of plots using Cloud Explorer. This led to the idea of data reprocessing and incorporating the 2D plots into the 3D environment.

In the later stages, they focused more on studying the interaction of the clouds with the environment around them. They built particle tracing into their Large-Eddy Simulation to be able to quantify, statistically, how the air in and around the clouds moved around. However, it was difficult for them to visualize the particle motion, which led to creating the GPU particle tracing engine. The dynamic nature of the clouds also led to several of the particle tracing extensions such as ellipsoid particles and viewing particles on their pathlines.

6.5.2 Using Cloud Explorer

Throughout its development, Cloud Explorer was used by the atmospheric scientists for research purposes. It played an important role in helping to answer the cloud research questions posed in Section 1.2.1: what are the defining characteristics of cumulus clouds during the different life cycle stages and how does a cumulus cloud influence and interact with the dry air around it. For both of these questions, Cloud Explorer was helpful both in the research process and also in sharing the results with others.

On the research side, Cloud Explorer played a key role in identifying interesting cumulus clouds to study, as described in Chapter 2. The dynamic, time-dependent nature of the clouds made an automated selection process for identifying interesting clouds difficult. Clouds often break into multiple pieces or collide with each other. By tracking clouds through time

and presenting visually to the atmospheric scientists, they could readily pick the clouds they wanted to study.

During the life cycle research, the scientists created several statistical plots of the interesting clouds to identify and study the different life cycle stages. What they found was that the clouds did not follow the traditional life cycle of birth, maturity and dying out. Instead, the clouds pulsate as they are fueled by pulses at regular intervals of warm, moist air. This behavior was also clearly visible in some of the statistical plots. By incorporating these plots, which could be generated through "reprocessing" the data, as described in Chapter 3, other scientists could readily understand the pulsating behavior.

In the next phase of the research, studying the interaction of clouds with their surroundings, the particle tracing described in Chapter 5 played an important role. Through the use of particle tracing, the scientists were able to verify the existence of the so-called descending shell: a thin layer of downward moving air around a cloud that compensates for the upward air movement in the cloud itself. They were also able to verify that the descending shell was caused by air on the sides of clouds subsiding due to the liquid water there evaporating. While the "hard" evidence was provided by statistical analysis of dispersion shown by massive numbers of particles advected in the large-eddy simulation itself, the insight into the descending shell was gained by visually watching the particles in and around the clouds. This insight guided the statistical analysis. The particle tracing also helped others quickly see the descending shell.

6.5.3 Lessons Learned

Collaboration between atmospheric scientists and visualization experts has proven fruitful for both parties. The atmospheric scientists were a source of interesting and challenging visualization data as well as research directions to follow. They were then able to benefit from using the visualization techniques described in this thesis to help advance their research. The atmospheric research results obtained with the help of visualization can also be seen as a validation of the visualization research. It's important, though, to emphasize the nature of such a collaboration and the risks, from a visualization research perspective, involved with it.

The collaboration described here was primarily two independent lines of research where regular contact, on a monthly basis, was maintained between visualization and atmospheric researchers. The atmospheric scientists made use of the visualization software at certain key times during their research. The lack of regular use of the visualization software can be attributed to several factors: the visualization research started after the atmospheric research, the visualization was often prototype quality requiring expert guidance to use, the VR software could only be run in the VR laboratory, and the visualization software was only a small part of the larger atmospheric research process. Even though the visualizations were infrequently used, they were still very valuable to the atmospheric scientists due to the insight they could gain with them.

While more frequent usage is in some ways desirable, it also has many risks. Visualization research often sits quite close to applications research. Investing the time to develop visualization tools that are sufficiently polished, flexible, and easy to use that they become

embedded in the domain scientists' work flow can easily cross the sometimes fuzzy boundary between applications research and general software engineering. Also, working towards the goal of providing such tools often leads to focusing on tools that are interesting to the domain scientists but are not interesting for visualization research. It's necessary thus to strike a balance between value for domain scientists and value for visualization research when collaboratively developing new visualization tools.

CHAPTER 7

Interactive Simulation and Visualization of Atmospheric Large-Eddy Simulations

Abstract

In this chapter, GALES is presented. GALES is a physically-correct, GPU-accelerated, Atmospheric Large-Eddy Simulation. GALES is a mixed-precision LES implementation using NVIDIA's CUDA to parallelize and accelerate computation while also providing interactive visualization. GALES, with visualization, outperforms an existing LES implementation running on 32 processor cores of a supercomputer. We demonstrate physical correctness by comparing GALES's results to those of other LES implementations. The ability to run interactive, scientific simulations on a desktop computer offers new possibilities for atmospheric researchers.

7.1 Introduction

Cloud dynamics is an important research topic in atmospheric physics. One way to investigate these dynamics is to simulate the turbulent atmospheric boundary layer using Large-Eddy Simulation (LES). LES resolves the dynamic effects of large-scale eddies and uses a statistical model for the dynamics at smaller scales. It can be used at the desired domain size and grid resolution at reasonable computational cost. Typically, LESs are run off-line as batch jobs on supercomputers and generate very large, time-dependent data sets. This makes it difficult to get timely, interactive feedback about running simulations. Gaining insight into simulation results requires extensive post processing and visualization of the simulation data.

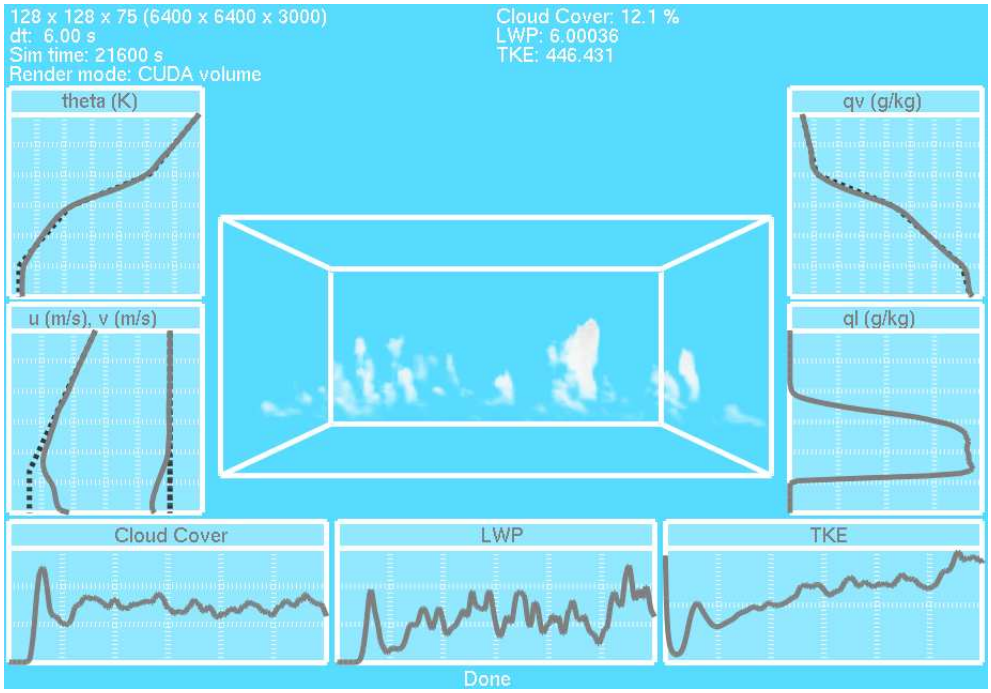


Figure 7.1: GALES's interactive visualization.

To address these issues, we developed GALES: a physically-correct, GPU-accelerated, Atmospheric, Large-Eddy Simulation that integrates interactive visualization (Figure 7.1). GALES is based on the Dutch Atmospheric Large-Eddy Simulation (DALES) [17, 47] and is one of the first GPU-accelerated scientific LES implementations. By using NVIDIA's CUDA [70] to parallelize and accelerate computation and incorporating mixed precision, GALES can outperform DALES running on 32 cores of a supercomputer. We have also verified that GALES and several existing LES implementations produce quantitatively comparable results. The ability to run interactive, scientific simulations on a desktop computer offers new possibilities for atmospheric researchers.

The remainder of this paper is structured as follows. In Section 7.2, we discuss related work and background information on LES, cloud simulation and CUDA. Sections 7.3 and 7.4 describe, respectively, the technical details of GALES and its implementation. Section 7.5 discusses GALES's interactive visualization. In Section 7.6, we present performance results for GALES and compare it with other LES implementations. We conclude in Section 7.7.

7.2 Background and Related Work

A detailed survey of compuational fluid dynamics (CFD) and CFD visualization techniques is outside the scope of this paper. It is important to note, though, that most computer graphics CFD research has placed more importance on performance and appearance than physically-correct simulation. See, for example, Stam [95] and Fedkiw, Stam and Jensen [26]. In this paper, by contrast, we are interested in accurate simulation with interactive performance that can be used in atmospheric research.

7.2.1 CFD for Cloud-Dynamics Studies

Most CFD simulations aim to efficiently solve the Navier-Stokes equations up to a desired level of detail. In turbulent fluid flow, the behavior of small scales, down to the Kolmogorov scale, are important for the flow, but these small scales are not always interesting to researchers.

In atmospheric research, three common approaches to solving the Navier-Stokes equations are Direct Numerical Simulation (DNS), Large-Eddy Simulation (LES) and Reynolds-Averaged Navier-Stokes (RANS). DNS explicitly resolves all turbulent structures, down to the Kolmogorov scale. RANS, on the other hand, models the entire turbulent structure of the flow. LES explicitly resolves the largest turbulent structures but models smaller (typically below 10 m) scales.

Due to the size and turbulent nature of clouds, LES is the most suitable of the three techniques for high resolution modeling of cloud dynamics. LES has been used in modeling of the cloudy atmospheric boundary layer since [92] and has been extensively validated by comparisons with measurement campaigns (see, e.g., [87, 11]).

7.2.2 Cloud Simulation

In computer graphics, research focusing on cloud simulation is most closely related to our work. Due to the computational complexity of cloud-resolving CFD simulations, most early approaches were either not real time (e.g. Kajiya and von Herzen [55]) or used crude approximations of the physics to generate a realistic appearance (e.g.. Dobashi et al. [25]).

As computational power has increased, cloud simulations have focused more on physical correctness. Overby, Melek and Keyser [72] and Harris et al. [43] developed more sophisticated cloud simulations that solved the Navier-Stokes equations using techniques from Fedkiw, Stam and Jensen [26]. These simulations, however, employed some simplifying assumptions and were not yet truly interactive. GALES, on the other hand, offers truly interactive performance while being based on the fundamental equations of cloud physics.

7.2.3 CUDA Overview

We chose NVIDIA's CUDA [70] for GALES's implementation. CUDA supports the stream programming model, where *kernels* operate on elements from *streams* of data. CUDA provides extensions to C and C++ allowing kernels to be written that are executed on the GPU. When a typical kernel executes, a separate GPU thread operates on each output element in a stream. These threads are organized into groups of 32 threads called *warps*. *Blocks* are comprised of one or more warps, and synchronization can occur between threads in a block. Blocks are organized into *grids*. The hierarchy of grids and blocks is used by threads to determine the stream data elements to process.

7.3 Large-Eddy Simulation Details

GALES is closely related to DALES and is similar to other atmospheric LESs. As such, we briefly present the simulation details here and refer the reader the literature on DALES [17,47] and to Moeng [68] for a more in depth explanation.

GALES resolves flow motion using the Boussinesq approximation and after applying the LES filter. The equations of motion that are resolved in GALES, using Einstein's notation, are:

$$
\begin{aligned}
\frac{\partial \widetilde{u}_i}{\partial x_i} &= 0, \\
\frac{\partial \widetilde{u}_i}{\partial t} &= -\frac{\partial \widetilde{u}_i \widetilde{u}_j}{\partial x_j} - \frac{\partial \pi}{\partial x_i} + \frac{g}{\theta_0}(\widetilde{\theta}_v - \theta_0)\delta_{i3} - 2\varepsilon_{ijk}\Omega_j \widetilde{u}_k + F_i - \frac{\partial \tau_{ij}}{\partial x_j}, \\
\frac{\partial \widetilde{\theta}_l}{\partial t} &= -\frac{\partial \widetilde{u}_j \widetilde{\theta}_l}{\partial x_j} + S_{\theta_l} - \frac{\partial R_{u_j,\theta_l}}{\partial x_j}, \\
\frac{\partial \widetilde{q}_t}{\partial t} &= -\frac{\partial \widetilde{u}_j \widetilde{q}_t}{\partial x_j} + S_{q_t} - \frac{\partial R_{u_j,q_t}}{\partial x_j},
\end{aligned}
$$

where the tildes denote the filtered mean variables, averaged over a grid cell. $\vec{u}$ is velocity, g is the gravitational constant, and δ is the size of a grid cell. The thermodynamical state of the system is defined by the scalar values: liquid water potential temperature θ_l, the total liquid water content q_t, and the pressure p. θ_v is the virtual potential temperature. θ_0 is the reference state potential temperature and $\vec{\Omega}$ is the earth's angular velocity. Viscous transport terms are neglected. F_i represents large scale forcings. The large-scale source terms for scalar φ are given by S_φ. The subfilter-scale (SFS), or residual, scalar fluxes are denoted by $R_{u_j,\varphi} \equiv \widetilde{u_j\varphi} - \widetilde{u_j}\widetilde{\varphi}$, i.e. the contribution to the resolved motion from all scales below the LES filter width. The sub-filter scale momentum fluxes are denoted by tensor τ_{ij}. These are the small-scale turbulent fluctuations that need to be modeled. Following DALES, they are modeled using one-and-a-half order closure [21]. The turbulent kinetic energy of the SFS turbulence is included in the modified pressure:

$$\pi = \frac{1}{\rho_0}(\widetilde{p} - p_0) + \frac{2}{3}e,$$

which is determined by solving a Poisson equation:

$$\frac{\partial^2 \pi}{\partial x_i^2} = \frac{\partial}{\partial x_i}(-\frac{\partial \widetilde{u_i}\widetilde{u_j}}{\partial x_j} + \frac{g}{\theta_0}(\widetilde{\theta_v} - \theta_0)\delta_{i3} - 2\varepsilon_{ijk}\Omega_j\widetilde{u_k} + F_i - \frac{\partial \tau_{ij}}{\partial x_j}).$$

7.3.1 Grid and Numerical Schemes

The simulation is discretized on a staggered grid, where the size of a grid cell is $(\Delta x, \Delta y, \Delta z)$ and $\Delta x = \Delta y$. The pressure, SFS kinetic energy, and the scalars are defined at cell centers. Velocity components are centered at corresponding cell faces.

For time integration, we use a third-order Runge-Kutta scheme (see [107]). For advection, we use a second-order central differencing scheme.

7.3.2 Condensation

The condensation scheme is used to calculate the liquid water content q_l from pressure, temperature and total water content. In the model, we assume that there is no liquid water present in an unsaturated grid cell, while all moisture above saturation value q_s is liquid water:

$$q_l = \begin{cases} \widetilde{q_t} - q_s & \text{if } \widetilde{q_t} > q_s \\ 0 & \text{otherwise.} \end{cases}$$

To calculate $q_s \equiv q_s(\widetilde{T}, p)$, where T is temperature, an implicit equation needs to be solved, which is done following [92].

7.3.3 Boundary Conditions

The computational domain has periodic boundary conditions in the horizontal directions. At the top of the domain, we take:

$$\frac{\partial \widetilde{u}}{\partial z} = \frac{\partial \widetilde{v}}{\partial z} = \widetilde{w} = 0; \frac{\partial \widetilde{\theta_l}}{\partial z}, \frac{\partial \widetilde{q_t}}{\partial z} \text{ constant.}$$

Horizontal fluctuations at the top of the domain (for instance gravity waves) are damped out by a sponge layer through an additional forcing/source term:

$$F_{\text{sp}} = -\frac{1}{\tau_r}\left(\overline{\phi}(z) - \phi\right),$$

with $\overline{\phi}$ the slab average value of quantity ϕ, and τ_r a relaxation time scale that goes from $\tau_0 = 360\,\text{s}$ at the top of the domain to infinity at the bottom of the sponge layer.

At the surface, velocities are equal to zero, and either surface values or their subfilter-scale fluxes for $\widetilde{\theta_l}$ and $\widetilde{q_t}$ are prescribed. Monin-Obukhov similarity theory is used to calculate the remainder of the surface conditions, see for example [28].

7.4 Implementation Details

The basic structure of GALES is a loop that computes a series of updates to the velocity field and scalars in the LES and uses these updates to perform integration in time. The basic steps performed in the loop are:

1. Compute updates from advection.
2. Compute updates from SFS terms.
3. Apply Coriolis and gravitational forces.
4. Apply large-scale forcings.
5. Compute pressure.
6. Create a divergence free velocity field.
7. Perform time integration.
8. Enforce boundary conditions.
9. Update the thermodynamic state.
10. Update the LES visualization.

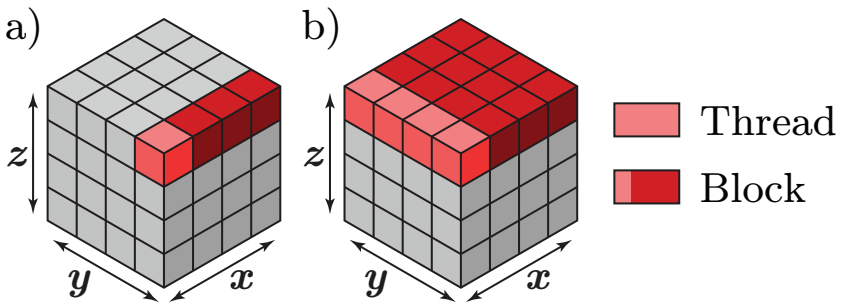


Figure 7.2: CUDA block and thread structure for a) regular kernels and b) reduction kernels.

We have chosen to implement these computations using CUDA kernels because implementing these steps on regular grids is well suited to the stream programming model. With some exceptions, for each grid cell, these updates can be computed independently and require only information from spatially local cells. Once a framework for working with the simulation domain has been established, implementing kernels to compute most of these updates is straightforward. We will now briefly discuss our framework for these kernels and highlight portions of the implementation that required special attention. We will also discuss the use of mixed precision in the simulation.

7.4.1 Kernel Structure and Memory Layout

The simulation grid is arranged in memory using the standard x-major order. In order to achieve the best performance, it is important that warps of CUDA threads read contiguous portions of data from rows in the simulation grid. Therefore, for most kernels, we structure our CUDA thread blocks so that each block processes one entire row in the 3D domain at a time. See Figure 7.2a.

In GALES, two primary types of kernels have been defined: regular kernels and reduction kernels. Regular kernels compute new values for each grid cell in the domain. Reduction kernels compute a mean, minimum or maximum value for an entire horizontal slab in the domain. For regular kernels, with a simulation grid size of $X \times Y \times Z$, we execute a CUDA grid of $Y \times Z$ blocks, where each thread computes values for one cell and each block computes values for one row in the domain (Figure 7.2a). For reduction kernels, we execute a CUDA grid of Z blocks, where each block processes an entire horizontal slab. Each thread in a block first reduces a column in the slab to a single value, resulting in one row of values, which is stored in shared memory. See Figure 7.2b. A standard pairwise reduction is then applied to this row of values to reduce it to a single, final value.

There are two additional memory access patterns of note in GALES. First, whenever threads for a kernel must access data from other cells in a row, the entire row is loaded into shared memory for a block. In this way the memory access is coalesced while providing

random access to the row's data for that thread block. Secondly, whenever threads must access a constant value per horizontal slab, e.g. the average temperature, this value is read from *constant* memory.

7.4.2 Poisson Solver

The Poisson solver is a key part of GALES and it is used to compute pressure, which in turn is used to project a divergence free velocity field. Since our domain is periodic in the horizontal directions, we can use the Fast Fourier Transform (FFT) to solve the Poisson equation in these directions. In the vertical direction, we solve a tri-diagonal matrix system using the well-known Thomas algorithm.

For the FFT portion of our Poisson solver, we opted to implement a GPU version of the power-of-two real-to-real transform from FFTPACK [96]. We chose to do this rather than using the CUDA FFT library, CUFFT, for two reasons. First, DALES makes use of FFTPACK so we could more closely match its LES implementation. Secondly, CUFFT is not optimized for the type of FFTs that GALES needs to perform, which is many, small 2D FFTs.

Our GPU version performs a batch of 2D FFTs simultaneously (one for each horizontal slab in the domain). Each 2D FFT is performed as a 1D FFT first along the rows and then along the columns or vice versa. For each 1D FFT pass, we prepare the data by transposing each slab appropriately with an optimized transpose kernel so that the 3D domain can be seen as a $(Y * Z) \times X$ 2D matrix, where each column will be transformed. We then execute CUDA blocks such that each block performs the 1D FFT on a group of columns in the matrix simultaneously. This ensures that reads and writes during the FFT will be coalesced.

7.4.3 Mixed Precision

GALES can perform its computations in a variety of precision modes, from fully single precision to fully double precision, but it is primarily intended to be used in mixed-precision mode. Mixed-precision offers a good balance between speed and precision by allowing the simulation to take larger time steps without sacrificing much performance.

In mixed precision mode, all computations in GALES are performed in single precision with the following exceptions. The Poisson solver uses double precision because we found it to be the most numerically sensitive step in GALES. We use double precision in our statistics routines since they sum large numbers of small values. Also, the few CPU calculations are done in double precision.

Newer CUDA-enabled NVIDIA GPUs support double precision natively. On these GPUs, GALES uses this for its double precision calculations. On older CUDA-enabled GPUs, however, there is only support for single precision. To allow for higher precision on these GPUs, we implemented emulated double precision using a double-single data type, based on DSFUN90 [6] and the float-float multiplication operator from Da Graça and Defour [18]. This data type achieves approximately 46 bits of precision in the mantissa.

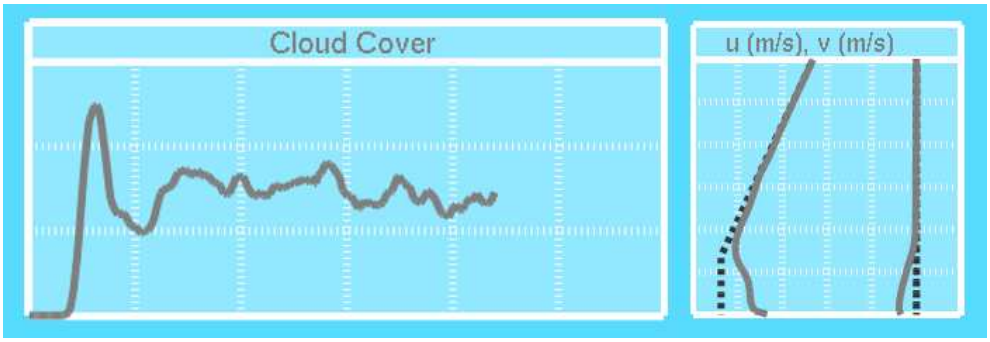


Figure 7.3: Plots from the GALES visualization. Left: A time history of percent cloud cover. Right: An instantaneous profile of *u* and *v* wind speed components at different heights in the domain.

7.4.4 CPU Computation

With two exceptions, simulation computation is performed exclusively on the GPU in GALES. The first exception is some one-off computation during initialization. The second exception is computation performed during the simulation that is based entirely on quantities that are constant within a single horizontal slab in the domain. These computations are: computing some input parameters for the surface routine kernels and computing the pressure and Exner values for each horizontal slab in the domain.

7.5 Interactive Visualization

GALES can provide interactive visualization of running simulations with little impact on performance. The visualizations include both a volume visualization of the clouds in the simulation and various statistical plots. These provide an overview of the current simulation state and an overview of some trends during the simulation run. This interactive insight into the simulation state and behavior is a marked contrast with DALES, where visualizing the results of simulation runs requires storing and extensive processing of very large data sets.

The primary visualization is the volume visualization of clouds in the simulation. This is implemented using simple volume ray-casting, which provides reasonable results. See Figures 7.1 and 7.4. This is coupled with simple interaction so that the user can navigate through the simulation volume to get a better sense of the spatial relation between the clouds in the simulation.

We also incorporate two types of statistical plots into GALES. The plots can be seen in context in Figure 7.1. One type of plot is a time history plot, which shows the progression of a single scalar value, such as percent cloud cover (Figure 7.3, left), over time. These plots provide a high-level overview of the simulation behavior that led to the current state. The

	64x64x80	128x128x80	256x256x80
DALES-SINGLE	6240s	24908s	101910s
DALES-MPI	116s	438s	2092s
GALES-NOVIS	72s	195s	761s
GALES-VIS	104s	225s	791s

Table 7.1: This table lists run times in seconds for DALES and GALES in different configuations.

other type of plot is a profile plot, which shows scalar values, such as wind speed components (Figure 7.3, right), averaged over each horizontal slab in the domain. These provide more quantitative insight into the current state of the simulation and their change over time can highlight interesting trends in the data.

In principle, all parameters of the LES can be made accessible for interactive steering of the simulation. If a truly interactive simulation is available, this can have significant impact on atmospheric research practice. At this time, though, the effect on the workflow of the atmospheric scientists is hard to assess, and design of a true steering interface for GALES would require a large collaborative effort between atmospheric scientists and interaction/visualization researchers.

7.6 Results

GALES can simulate many of the same cases that DALES can simulate. Figure 7.4 shows two of these. The bottom case is the BOMEX case, which is discussed in Section 7.6.2

7.6.1 Performance

We tested the relative performance of GALES, both with and without visualization, and DALES. In our tests, we used the same input for each simulation, altering only the grid size between tests. We tested each simulation on grids of $64 \times 64 \times 80$ cells ($3.2 \times 3.2 \times 3.2\,\mathrm{km}^3$), $128 \times 128 \times 80$ cells ($6.4 \times 6.4 \times 3.2\,\mathrm{km}^3$) and $256 \times 256 \times 80$ cells ($12.8 \times 12.8 \times 3.2\,\mathrm{km}^3$). In each test, we simulated 6 hours of real time with time steps of 10 s. We ran DALES (DALES-MPI) on one supercomputer node with 16 4.7 GHz dual-core processors, for a total of 32 cores. We ran GALES, with (GALES-VIS) and without (GALES-NOVIS) visualization. In both cases, we ran GALES in mixed-precision mode using native single and double precision on a 2.4 GHz Pentium quad-core PC with an NVIDIA GTX 280-based GPU. For comparison purposes, we also ran DALES on one core of the same PC (DALES-SINGLE).

The results of our tests are shown in Table 7.1. The CPU time for DALES increases approximately linearly with the size of the simulation grid. GALES also demonstrates this behavior when moving from the $128 \times 128 \times 80$ grid to the $256 \times 256 \times 80$ grid. However, the slower times at the lowest resolution indicate that GALES makes less effective use of the GPU's processing power at that resolution. The roughly constant difference of 30s between GALES-NOVIS and GALES-VIS is due to the fact that we did not change the user's view

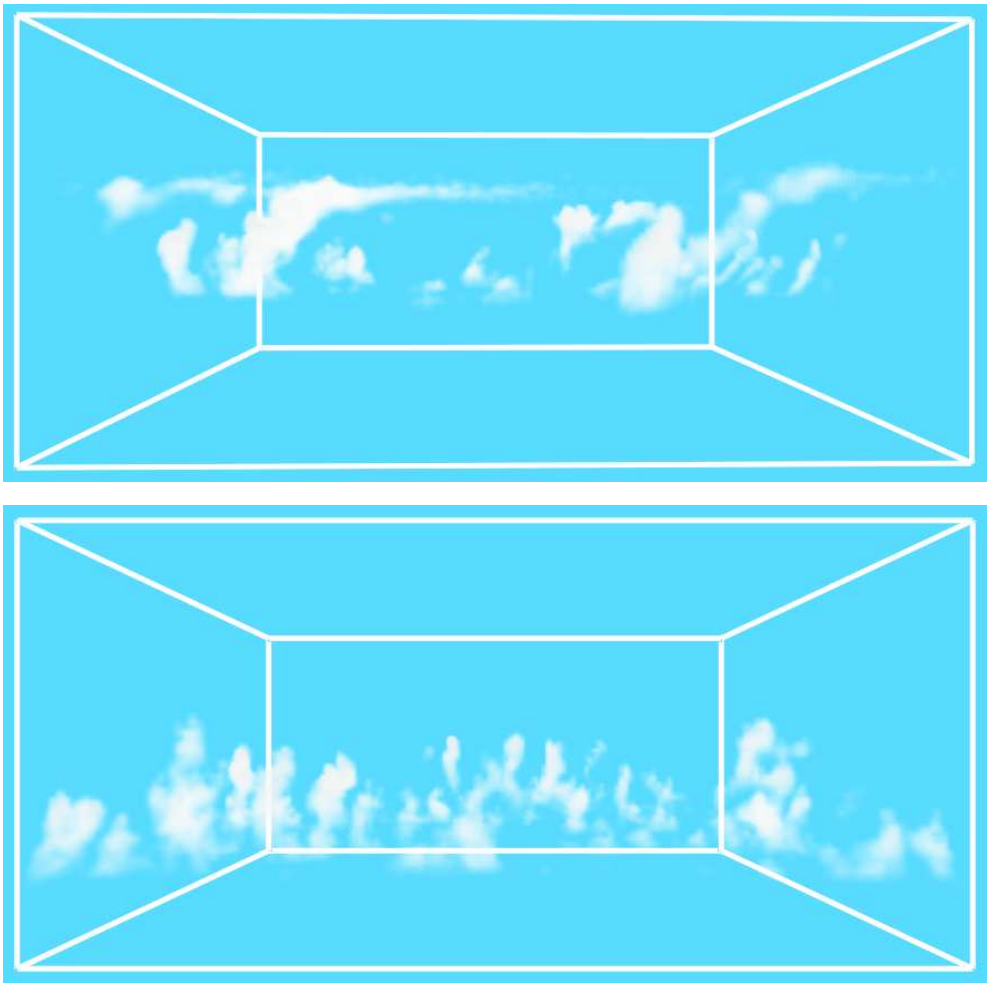


Figure 7.4: Two of the cases GALES can simulate.

	64x64x80	128x128x80	256x256x80
FLT-FLT*	189s	538s	2055s
FLT-DBL	72s	195s	761s
DBL-DBL	147s	451s	-
FLT-DBS	74s	204s	805s
DBS-DBS	237s	651s	-

Table 7.2: This table lists run times in seconds for GALES using various precision modes. FLT-FLT was run with a time step of 3s for stability reasons. The other configurations were run with a time step of 10s.

of the simulation during execution. Thus, we only needed to render one image per simulation time step, and we used the same number of ray-casting iterations regardless of the grid resolution.

We also tested the performance of the different precision modes implemented in GALES. These tests used the same grids as we did in the previous tests, and we ran these tests on the same PC. We tested GALES using single precision (FLT-FLT), mixed single and native double precision (FLT-DBL), native double precision (DBL-DBL), mixed single and emulated double precision (FLT-DBS), and emulated double precision (DBS-DBS). The FLT-DBL results are precisely those from the previous test. The emulated double precision used our implementation of the double-single data type. We were unable to run the DBL-DBL and DBS-DBS tests for the $256 \times 256 \times 80$ grids as they required more than the available 1 GB of memory on our GPU. Also, for the FLT-FLT tests, we used a time step of 3s since that was the largest stable time step.

Table 7.2 lists the results of our tests. In per time step cost, FLT-DBL and FLT-DBS were only about 20% to 30% slower than FLT-FLT. This slight penalty is more than made up for by being able to take larger time steps. Surprisingly, the performance for both FLT-DBS and FLT-DBL was roughly equal. This is likely due to the limited use of (emulated) double precision in mixed precision mode and the performance difference between single and double precision arithmetic on the GPU. Further, all tests followed the same pattern of a roughly 3x increase in execution time between $64 \times 64 \times 80$ and $128 \times 128 \times 80$ grids.

It is important to note that grid sizes of $64 \times 64 \times 80$ and $128 \times 128 \times 80$ are representative grid sizes. While there is a trend towards higher resolution grids for some problems, these sizes are frequently used in a variety of experiments, such as those in the following section.

7.6.2 BOMEX Comparison

This section compares results from GALES with an LES intercomparison study by Siebesma et al. [87]. In this study, Siebesma et al. compared 10 different LES implementations that simulated the BOMEX (the Barbados Oceanographic and Meteorological Experiment [50]) case study. We chose this study since BOMEX is widely studied and relatively straightforward. Also, GALES has all of the functionality necessary to simulate BOMEX, and we have results from DALES for BOMEX. This case can be seen in the bottom of Figure 7.4.

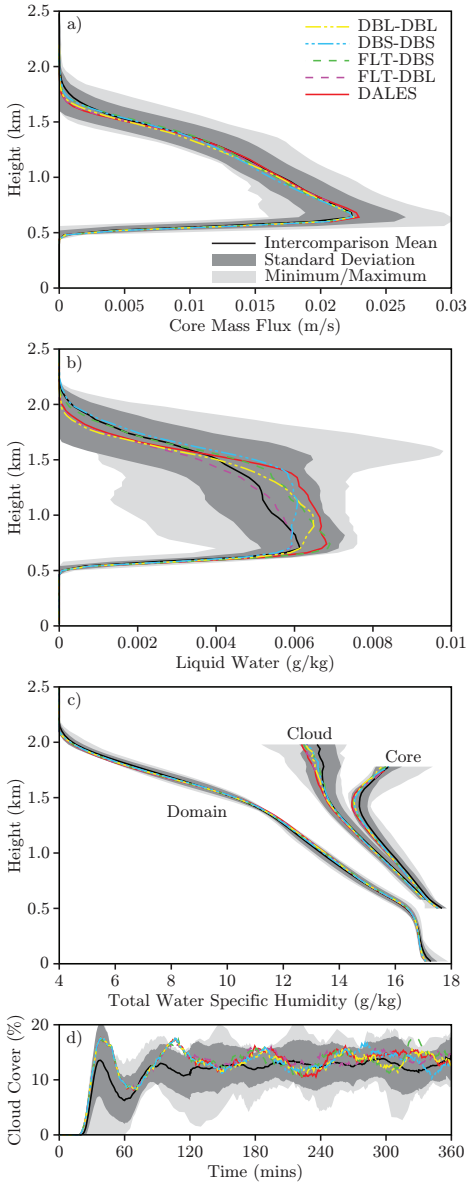


Figure 7.5: Comparisons of our results and DALES with those from the LES intercomparison study by Siebesma et al. [87].

We simulated the BOMEX conditions in a $6.4 \times 6.4 \times 3.0\,\text{km}^3$ domain during a 6 hour time span using a simulation grid of resolution $64 \times 64 \times 75$ and a time step of 10 s, as in the intercomparison study. We repeated the test case for GALES in the FLT-DBL, DBL-DBL, FLT-DBS and DBS-DBS precision modes described in the previous section. We also ran the test case using DALES.

Figure 7.5 shows some of our results superimposed over the results from the original study. Figure 7.5a shows the average mass flux (vertical velocity) by height for positively buoyant, cloudy grid cells. Figure 7.5b shows the average amount of liquid water by height. Figure 7.5c shows the average total water specific humidity by height for the whole domain (Domain), cloudy grid cells (Cloud) and positively buoyant, cloudy grid cells (Core). Figure 7.5d shows the evolution in time of cloud cover (percent of vertical columns in the grid containing liquid water) during the course of the simulation. The liquid water profile and the total water specific humidity domain profile were averaged over the last hour of the simulation. The mass flux profile and the other two total water specific humidity profiles were averaged over the last three hours of the simulation.

A complete listing of our results compared to those of the study is beyond the scope of this paper. However, as in Figure 7.5, our results were generally within one standard deviation of the intercomparison mean, and the results from GALES were always in good agreement with the results from DALES. Given that minor deviations are expected in turbulent flow and that the results from GALES always agreed well with the results from DALES, we have confidence that GALES is physically correct.

One important additional result of this comparison is a validation of our mixed precision implementation. Given how well the results from all precision modes agree with each other, DALES and the other LES simulations, we can conclude that our use of mixed precision does not significantly alter the simulation results.

7.7 Conclusions and Future Work

In this paper, we presented GALES, our physically-correct, GPU-accelerated, Atmospheric Large-Eddy Simulation. Based on the existing DALES, GALES is a mixed-precision, CUDA LES implementation. GALES also includes interactive visualization capabilities, which give immediate insight into running simulations.

In our experiments, we showed that GALES outperforms DALES running on 32 processor cores on a supercomputer while still providing interactive visualization. This translated into a roughly 50x speedup over DALES running on a single processor core on the same standard, desktop computer.

We demonstrated GALES's physical-correctness by simulating the well-known BOMEX case study. We compared our results to those of DALES and the 10 simulations in the case study. Our results agreed well with those from the literature and agreed very well with the results from DALES.

GALES offers many new possibilities for atmospheric scientists. By being physically accurate, GALES can be employed as a research tool. It can be used for its raw simulation

power to run many simulations to explore the simulation parameter space. However, it can also be used as a high-performance, interactive tool. Its interactive visualization, among other things, reduces the need to store large amounts of data and can reduce the time required to set up new case studies.

Beyond some of these possibilities, there are many options for expanding the simulation's visualization capabilities. Additions such as particle-tracing, improved cloud rendering, and an improved user-interface are all planned for the future. GALES's interactive performance also opens the door to computationally steering the simulation, which is another interesting avenue to explore.

CHAPTER 8

Conclusions

This chapter discusses the results of the research presented in this thesis as a whole and how it fits into the larger context of visualization research. The chapter closes with a look at where to go with the research in the future and some unanswered questions raised during the work presented here.

8.1 Thesis Summary

This thesis presented a variety of techniques for visualizing large, multivariate, time-varying, 3D data. The data was generated by Large-Eddy Simulations of fair-weather cumulus clouds. This thesis also introduced a new, GPU-based LES implementation to interactively simulate the cumulus clouds.

Chapter 2 focused on feature tracking. It introduced an automated method for detecting and tracking cumulus clouds in the LES data. The cumulus clouds were then visualized in the initial implementation of Cloud Explorer, which let the atmospheric scientists visually identify interesting clouds. By selecting approximately 40 clouds, they were able to use statistical techniques to study the cloud life cycles.

Chapter 3 focused on data reprocessing. The data processing pipeline was updated to support generating new, derived data based on mathematical expressions supplied by the atmospheric scientists. This data was calculated on a per-feature basis and then included in Cloud Explorer in the form of statistical plots associated with selected clouds. The Cloud Explorer user interface was also extended to support these plots, which introduced more quantitative information into the virtual environment.

Chapter 4 presented a technique for compressing normal vectors using quantization. The technique presented has two advantages: decompression is a matter of a table look up and the angular error introduced by quantization is bounded. Using this technique, normal vectors for cloud isosurfaces can be compressed, reducing their size on disk. By extending this technique to arbitrary vectors by also quantizing the vector length, the vector field data from the LES can also be compressed. Using the compressed data helps to overcome the bottlenecks between disk and main memory and between main memory and the GPU.

Chapter 5 introduced a GPU-based particle tracing system that can advect over a million particles at interactive frame rates. The particle tracing works with both quantized vector fields and uncompressed vector fields, and supports standard particle tracing, the standard flow curves, rendering flow oriented ellipsoids, and drawing particles on their pathlines. The particle tracing engine was integrated into Cloud Explorer, as well as standalone desktop and VR particle tracing applications.

Chapter 6 gave an overview of the final Cloud Explorer system. Cloud Explorer's user interface makes use of IntenSelect together with a hybrid interface for interaction and it provides the user with various forms of contextual information. Cloud Explorer employs data compression through quantization, both of vector information as well as geometry vertices, to improve performance. Cloud Explorer maintains a cache of the most recently used visualization data in memory to support rapidly browsing back and forth through recent time steps. Cloud Explorer's visualization environment can be run in a variety of virtual reality environments, and it can also run as a standard, desktop application using the same user interface.

Chapter 6 also discussed how Cloud Explorer was used by the atmospheric scientists for their research. Using Cloud Explorer, atmospheric scientists can process and interactively visualize LES data both in virtual reality and on a desktop computer. In this project, Cloud Explorer was used to help answer the two atmospheric research questions posed in Section 1.2.1: what are the defining characteristics of cumulus clouds at different stages of their life cycles and how do cumulus clouds interact with and influence the dry air around them. By statistically analyzing several clouds selected using Cloud Explorer, they found that the traditional cloud life cycle did not apply, and that, instead, the clouds were driven by a series of pulses of warm, moist air. By studying the motions of particles both in Cloud Explorer and on a larger scale in the LES, they concluded that cumulus clouds generate a "descending shell" of cooler air around them through lateral mixing, that compensates for the upward buoyant force of the cloudy air. In short, the collaboration with atmospheric scientists yielded fruitful research results for both atmospheric science and visualization even though the visualization techniques presented here were not completely embedded in the atmospheric science workflow.

Chapter 7 presented a new GPU-based LES, GALES, that can run interactively on a desktop computer while visualizing the running simulation. This is a departure from traditional way of running an LES on a remote supercomputer, and it represents an initial step towards a new mode of atmospheric simulation.

8.2 Visualization Challenges

In 2004, the year when this work started, Chris Johnson [52] listed 15 challenges for scientific visualization. These challenges identified important research areas both to improve the science of visualization and to provide insight into complex and varied scientific data. Of the 15 challenges Johnson names, many are relevant here and were addressed by the work presented in this thesis.

- **Think about the science.** In other words, visualization cannot exist in a vacuum. The data being visualized originates from some domain, and it's vital to collaborate with scientists from that domain when developing visualization techniques to study data from the domain.

  Here, one of the important goals was to answer specific questions about cumulus clouds, which are complex, turbulent structures. Answering these questions required specially tailored visualization techniques, and working closely with atmospheric scientists helped provide guidance in choosing suitable visualization techniques to develop. The feature tracking, data reprocessing, particle tracing in Cloud Explorer and GALES itself were all motivated by the atmospheric scientists. Their expert knowledge of the domain was also invaluable in validating these techniques.

- **Efficiently using novel hardware architectures.** Providing efficient processing and interactive visualization of large, multivariate, time-varying, 3D data like the cumulus cloud data is computationally demanding. Making the entire process interactive by integrating the simulation into the visualization is even more computationally demanding. Taking advantage of specialized hardware is currently the only way to make this feasible.

  The graphics processing unit (GPU) has been crucial to many of the techniques presented here. The particle tracing presented in Chapter 5 happens entirely on the GPU, which made it possible to interactively advect over one million particles. The GPU also plays an important role in decompressing quantized data and "decorating" the 3D clouds with contextual information. The increasing capability of GPUs for general purpose computing made the development of GALES possible.

- **Human-computer interaction.** Visualization tools must take ease-of-use into account or else they will be relegated to the status of demonstrations only usable by experts. When visualizing complex data like cumulus clouds, where the user needs access to many visualization tools and options, keeping the visualization environment usable quickly becomes very challenging.

  In Cloud Explorer, a number of steps were taken to address usability issues. For one, Cloud Explorer is a virtual reality (VR) application. As such, it supports direct interaction with the 3D data. For system control tasks, Cloud Explorer uses a set of familiar, 2D widgets such as windows, buttons and sliders. Cloud Explorer also functions as

a desktop application, where these 2D widgets behave as expected. In the VR environment, the IntenSelect [37] technique is used to simplify selection and manipulation tasks for the user.

- **Global/local visualization.** Visualizing a feature of interest in isolation is less useful than visualizing it within some larger context. With cumulus clouds, providing additional spatial and temporal cues helps the user relate features in the data to their spatial surroundings and their behavior over time.

  In Cloud Explorer, this contextual information takes several forms. The 3D data domain is shown both as a world-in-miniature and as a larger container for the 3D visualization. This provides a frame of reference for the clouds, the particles from particle tracing, and the slicing plane. Within the data domain, the clouds provide a reference frame for the features seen on the slicing plane and patterns seen in particle behavior. Statistical plots in 2D windows around the 3D domain show cloud behavior over time. Some of these plots also show behavior over time at different altitudes. Decorations on these plots highlight both the current time step and an altitude. The altitude corresponds with a white band drawn around the cumulus clouds. These different pieces of information can all help the user gain a better understanding of the data.

  GALES also shows the simulation domain as a 3D box, and any clouds in the simulation are visualized within the box. Information about the current state of the simulation is displayed with profile plots, which display the mean values of simulation variables at different altitudes in the domain. Information about the history of the simulation is shown with plots that show the mean value of simulation variables over the whole domain at different points in time.

- **Integrated problem-solving environments.** Understanding complex data through visualization is often an iterative process where scientists generate new data to visualize based on observations they made during previous visualization sessions. Providing scientists with an environment where they can experiment with the data in addition to just visualizing the data can assist with this process. The idea is that, by using such an environment, scientists can spend less time performing custom data processing or running additional simulations, which can reduce the time required for each step in the iterative process.

  The data reprocessing presented in Chapter 3 is meant to be a step in this direction. Cloud Explorer's data processing pipeline was extended to support calculating new derived data from the simulation data. The derived data can then be included in the visualization. This introduces quantifiable data into the visualization, providing a more rich visualization environment. It also allows the scientists to use Cloud Explorer to generate the same kinds of statistical information about the clouds in the data that they would normally generate using their own custom tools. Integrating the functionality into Cloud Explorer lets the scientists spend more time visualizing the cumulus clouds and less time developing custom tools.

- **Multifield visualization.** The cumulus cloud data studied here consists of much more than just clouds. Each time step consists of a 3D grid with six or more variables for each grid cell. Three of these variables comprise the velocity field, which is the wind speed and direction. One of the variables is liquid water, which is a direct cloud indicator. Additional variables include temperature and humidity. To properly understand the cloud behavior, it's necessary to visualize many of these variables in combination, but care must be taken to choose appropriate visualizations for the different variables.

  Here, a variety of techniques for visualizing the variables were presented. The cumulus clouds are visualized as isosurfaces in Cloud Explorer. In GALES, the clouds are visualized using direct volume rendering. Both techniques give a meaningful physical representation to the liquid water variable. In both GALES and Cloud Explorer, scalar quantities, either simulation variables or derived values, are visualized using statistical plots. These plots give insight into the mean temporal and/or spatial behavior of the variables. Cloud Explorer provides a slicing plane for visualizing individual scalar quantities. Cloud Explorer also provides particle tracing for directly visualizing the velocity field. Both the slicing plane and the particle tracing can be used together with the isosurface visualization to understand the relationship between the clouds and their environment. When used together, the clouds are drawn as contour lines so that the particles or slicing plane inside the clouds remain visible. By using each of these techniques and combining them as necessary, the user can develop a more complete understanding of the simulation data.

- **Feature detection.** In order to study features in a data set, they must first be detected. Thus, the first step in studying cumulus clouds in data generated from a Large-Eddy Simulation is to detect the clouds in the data and track them through time. Detecting and tracking clouds offers an interesting challenge: it is nearly trivial to programmatically identify individual clouds, but it is very difficult to automatically determine which clouds are interesting for study. In Cloud Explorer, the user is brought into the loop to resolve this dilemma. By observing the tracked clouds over time, the user can readily select clouds to study further.

- **Time-dependent visualization.** Cumulus clouds are not static; they evolve over time. As such, it is insufficient to study only static snapshots of the clouds. They must be studied in both space and time to understand how they develop and how they interact with their environment. To do this effectively requires a visualization environment that allows the user to interactively and intuitively navigate through both time and space. The environment must also incorporate global/local visualization techniques to help the user relate the instantaneous 3D state of a feature he or she sees with its full behavior in time.

  Here a number of approaches have been taken to help the user understand how the data changes in time. The Cloud Explorer visualization environment allows the user to play forwards and backwards in time through simulation data. Both Cloud Explorer and GALES use statistical plots to provide insight into behavior over time. With particle

tracing, particles can be rendered as flow-aligned ellipsoids to provide more information when the advection is paused. Also, particles can be rendered on their pathlines to connect their current position with their past (and future) positions.

8.3 Future Directions

The work presented in this thesis suggests three broad areas for future research efforts: improving Cloud Explorer, improving GALES, and improving collaboration with domain scientists.

There are several opportunities for extending and improving Cloud Explorer. New visualization techniques can be added such as coloring the cloud isosurfaces with properties from the data or visualizing the data with volume rendering. The idea of reprocessing can be extended by incorporating a data calculator into the visualization environment, which would better support experimentation with the data. Calculating derived data for visualization using the GPU may also be interesting to consider. The performance can be increased by making better use of multi-resolution data and multi-threaded data loading. Another avenue to explore is updating Cloud Explorer to support other types of cloud data, such as stratocumulus clouds, where studying individual features is less interesting than studying the general behavior of the cloud layer, and deep convection, where the simulation grids cover a much higher vertical domain and often have layers of different thicknesses. One additional question to consider is whether realistic cloud rendering can help atmospheric scientists related features from simulated clouds to those observed in nature.

The fast pace at which GPU technology is progressing opens the door for many new research directions. At the time of this writing, GPUs are undergoing major architectural changes to make them more amenable to general purpose computing while continuing to increase in computational power. The first step will be to update GALES to make the most of these next generation GPUs, which should allow it to run larger simulations with better visualizations interactively. Another important step to take is to have GALES run on multiple GPUs simultaneously. Extending GALES to make it more of an experimental platform would also be very interesting. One important option for this is to try to computationally steer the simulations to, which can, for example, let scientists ensure that certain events take place during a simulation. Also here adding a data calculator to the environment to let scientists derive arbitrary contextual data to visualize is another option. A further research area to explore is that of performing parameter studies by running multiple simulations simultaneously. This also brings the challenging comparative visualization problem of how to visualize multiple running simulations and understand their similarities and differences.

Developing a better understanding of how visualization is used by the domain scientists and how visualization can be better integrated into their workflows is important as the field of scientific visualization matures. Learning more about what the domain scientists wish to discover will help visualization scientists develop a cohesive set of visualization techniques that are more tailored to the domain scientists' needs. Studying how they use visualization tools can provide insight into how to make visualization more readily accessible to domain

scientists. Lowering the adoption threshold visualization tools can help domain scientists spend more and higher quality time using visualization tools. This, in turn, can help advance visualization towards one of the ultimate goals: developing a complete system for performing virtual experiments.

8.4 On Visualization in Virtual Reality

When taking a step back from these specifics of the visualization techniques presented in this thesis, there are two important unanswered questions that have come up along the way. When is virtual reality necessary for visualization and what level of immersion is necessary? In the author's anecdotal experience, interacting with 3D data is easier with the direct interaction afforded by VR. However, scientists often work at their own desktop computer, and it's unclear at what point this ease of interaction is significant enough to warrant them leaving their computer and going to a VR system, possibly far away and possibly requiring an appointment, to visualize their data. Similarly, in the author's experience, viewing data in stereo can be a great help in certain cases, such as particle tracing, but it's unclear when the benefits justify the cost and effort. Immersion level is another factor that complicates these questions. Systems with higher immersion are generally more expensive and less accessible to domain scientists. Does higher immersion level improve insight gained from visualization, and, if so, when does it justify the cost and effort?

VR clearly has a role to play in scientific visualization, and it has been successfully employed in industries like the automotive industry and the oil and gas industry. However, VR scientists have thus far shied away from answering these difficult questions. Moving forward, though, it is critical to answer them in order to clarify the role and quantify the value of VR in visualization. This will give VR visualization scientists a better sense of purpose and direction to help guide their research effort, which is currently lacking.

Bibliography

[1] *ISO/IEC 14496-2:2004 Information technology — Coding of audio-visual objects — Part 2: Visual*. International Organization for Standardization, Geneva, Switzerland, 2004.

[2] *ISO/IEC 14496-11:2005 Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*. International Organization for Standardization, Geneva, Switzerland, 2005.

[3] J.-H. Ahn, C.-S. Kim, and Y.-S. Ho. Predictive compression of geometry, color and normal data of 3-D mesh models. *IEEE Trans. on Circuits and Systems for Video Technology*, 16(2):291–299, 2006.

[4] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. Technical report, Los Alamos National Laboratory, 2003.

[5] A. Arakawa and V. R. Lamb. Computational design of the basic dynamical processes of the UCLA general circulation model. In *General circulation models of the atmosphere*, pages 173–265. Academic Press, Inc., New York, 1977.

[6] D. H. Bailey. Dsfun: A double-single floating point computation package. Published electronically at http://crd.lbl.gov/ dhbailey/mpdist/.

[7] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. on Mathematical Software*, 22(4):469–483, 1996.

[8] S. Bony, J.-L. Dufresne, R. Colman, V. M. Kattsov, R. P. Allan, C. S. Bretherton, A. Hall, S. Hallegatte, W. Ingram, D. A. Randall, B. J. Soden, G. Tselioudis, and M. J. Webb. How well do we understand and evaluate climate change feedback processes? *Journal of Climate*, 19(15):3445–3482, 2006.

[9] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proc. EG Rendering Workshop*, pages 53–64, 2002.

[10] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible markup language (XML) 1.0 (third edition). http://www.w3.org/TR/REC-xml/, February 2004.

[11] A. R. Brown, R. T. Cederwall, A. Chlond, P. G. Duynkerke, J. C. Golaz, M. Khairoutdinov, D. C. Lewellen, A. P. Lock, M. K. MacVean, C.-H. Moeng, R. A. J. Neggers, A. P. Siebesma, and B. Stevens. Large-eddy simulation of the diurnal cycle of shallow cumulus convection over land. *Quarterly Journal of the Royal Meteorological Society*, 128(582):1075–1093, 2002.

[12] S. Bryson. Virtual reality in scientific visualization. *Communications of the ACM*, 39(5):62–71, 1996.

[13] K. Bürger, J. Schneider, P. Kondratieva, J. Krüger, and R. Westermann. Interactive visual exploration of unsteady 3D-flows. In *Eurographics/IEEE VGTC Symposium on Visualization*, pages 251–258, 2007.

[14] R. Bürger and H. Hauser. Visualization of multi-variate scientific data. In *Eurographics 2007 State of the Art Reports*, pages 117–134, 2007.

[15] M. M. Chow. Optimized geometry compression for real-time rendering. In *Proc. Visualization*, pages 347–354, 559, 1997.

[16] C. Cruz-Neira, J. Leigh, M. Papka, C. Barnes, S. M. Cohen, S. Das, R. Engelmann, R. Hudson, T. Roy, L. Siegel, C. Vasilakis, T. A. DeFanti, and D. J. Sandin. Scientists in wonderland: A report on visualization applications in the CAVE virtual reality environment. In *Proceedings of IEEE Visualization*, pages 59–66, October 1993.

[17] J. Cuijpers and P. Duynkerke. Large-eddy simulation of trade wind cumulus clouds. *J. Amtos. Sci.*, 50(23):3894–3908, 1993.

[18] Guillaume Da Graça and David Defour. Implementation of float-float operators on graphics hardware. In *Proceedings of Real Numbers and Computers*, pages 23–32, 2006.

[19] A. van Dam, A. S. Forsberg, D. H. Laidlaw, J. LaViola, and R.M. R. M. Simpson. Immersive VR for Scientific Visualization: A Progress Report. *IEEE Computer Graphics and Applications*, pages 26–52, Nov/Dec 2000.

[20] A. van Dam, D. H. Laidlaw, and R. M. Simpson. Experiments in immersive virtual reality for scientific visualization. *Computers and Graphics*, 26(4):535–555, 2002.

[21] J. W. Deardorff. Three-dimensional numerical modeling of the planetary boundary layer. In *Workshop on Meteorology*, pages 271–311, 1973.

[22] M. Deering. Geometry compression. In *Proc. of ACM SIGGRAPH*, pages 13–20, 1995.

[23] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, June 2000.

[24] M. B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM*, 39(2):253–280, 1992.

[25] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of SIGGRAPH*, pages 19–28, 2000.

[26] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of SIGGRAPH*, pages 15–22, 2001.

[27] D. Foulser. IRIS Explorer: a framework for investigation. *ACM SIGGRAPH Computer Graphics*, 29(2):13–16, May 1995.

[28] J. R. Garratt. *The atmospheric boundary layer*. Cambridge University Press, 1992.

[29] J.E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry*. Chapman & Hall/CRC, 2004.

[30] M. Gopi and D. Eppstein. Single-strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forums*, 23(3):371–379, 2004.

[31] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPIX: A framework for high resolution discretization, and fast analysis of data distributed on the sphere. *Astrophysical Journal*, 622(2):759–771, 2005.

[32] C. Gotsman, S. Gumhold, and L. Kobbelt. Simplification and compression of 3D meshes. In *Tutorials on Multiresolution in Geometric Modelling*, pages 319–362, 2002.

[33] E. J. Griffith, M. Koutek, and F. H. Post. Fast normal vector compression with bounded error. In *Proc. Geometry Processing*, pages 263–272, 2007.

[34] E. J. Griffith, M. Koutek, F. H. Post, T. Heus, and H. J. J. Jonker. A reprocessing tool for quantitative data analysis in a virtual environment. In *Proc. ACM VRST 2006*, pages 212–215, 2006.

[35] E. J. Griffith, M. Koutek, F. H. Post, T. Heus, and H. J. J. Jonker. Quantitative data analysis in virtual environments through reprocessing. In *Proc. Thirteenth Annual Conference of the Advanced School of Computing and Imaging*, pages 343–350, 2007.

[36] E. J. Griffith, F. H. Post, M. Koutek, T. Heus, and H. J. J. Jonker. Feature tracking in VR for cumulus cloud life-cycle studies. In Erik Kjems and Roland Blach, editors, *Virtual Environments 2005*, pages 121–128, 2005.

[37] G. de Haan, E. J. Griffith, M. Koutek, and F. H. Post. Hybrid interfaces in ves: Intent and interaction. In Rogger Hubbold and Ming Lin, editors, *Virtual Environments 2006*, pages 109–118, May 2006.

[38] G. de Haan, E. J. Griffith, M. Koutek, and F. H. Post. Pdrive: The projector-based, desktop, reach-in virtual environment, 2007.

[39] G. de Haan, M. Koutek, and F. H. Post. IntenSelect: Using Dynamic Object Rating for Assisting 3D Object Selection. In Erik Kjems and Roland Blach, editors, *Proceedings of the 9th IPT and 11th Eurographics VE Workshop (EGVE) '05*, pages 201–209, 2005.

[40] H. Haase, M. Bock, E. Hergenröther, C. Knöpfle, H.-J. Koppert, F. Schröder, T. Trembilski, and J. Weidenhausen. Meteorology meets computer graphics a look at a wide range of weather visualisations for diverse audiences, 2000.

[41] R. B. Haber and D. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. In *Visualization in Scientific Computing*, pages 74–93, 1990.

[42] M. Harris and A. Lastra. Real-time cloud rendering. *Computer Graphics Forum*, 20:76–84, 2001.

[43] M. J. Harris, W. V. Baxter, III, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of Graphics hardware*, pages 92–101, 2003.

[44] H. Hauser, R. S. Laramee, and H. Doleisch. State-of-the-art report 2002 in flow visualization. Technical Report TR-VRVis-2002-003, VRVis Research Center, 2002.

[45] A. Henderson. *The ParavView Guide*. Kitware, Inc., 2004.

[46] C. Henze. Feature detection in linked derived spaces. In *Proceedings of IEEE Visualization*, pages 87–94, 1998.

[47] T. Heus. *On the Edge of a Cloud*. PhD thesis, Delft University of Technology, 2008.

[48] T. Heus, H. J. J. Jonker, E. J. Griffith, and F. H. Post. Lifecycle analysis of cumulus clouds using a 3D virtual reality environment. In *The 17th conference on Boundary-Layers and Turbulence*, 2006.

[49] W. L. Hibbard, J. Anderson, I. Foster, B. E. Paul, R. Jacob, C. Schafer, and M. K. Tyree. Exploring coupled atmosphere-ocean models using Vis5D. *International Journal of High Performance Computing Applications*, 10(2–3):211–222, 1996.

[50] J. Z. Holland and E. M. Rasmusson. Measurements of the atmospheric mass, energy, and momentum budgets over a 500-kilometer square of tropical ocean. *Monthly Weather Review*, 101(1):44–55, 1973.

[51] Ma. Isenburg and J. Snoeyink. Coding with ASCII: compact, yet text-based 3D content. In *Proc. 3D Data Processing*, pages 609–616, 2002.

[52] C. Johnson. Top scientific visualization research problems. *IEEE Computer Graphics and Applications*, 24(4):13–17, 2004.

[53] C. Johnson, T. Munzner, R. Moorhead, H. Pfister, P. Rheingans, and T. S. Yoo. *NIH/NSF Visualization Research Challenges Report*. IEEE Press, 2006.

[54] C. Johnson, S. G. Parker, C. Hansen, G. L. Kindlmann, and Y. Livnat. Interactive simulation and visualization. *Computer*, 32(12):59–65, 1999.

[55] J. T. Kajiya and B. P. von Herzen. Ray tracing volume densities. In *Proceedings of SIGGRAPH*, pages 165–174, 1984.

[56] P. Kipfer, M. Segal, and R. Westermann. Uberflow: A GPU-based particle engine. In *Proc. ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, 2004.

[57] P. Kondratieva, J. Krüger, and R. Westermann. The application of GPU particle tracing to diffusion tensor field visualization. In *Proc. IEEE Visualization*, pages 73–78, 2005.

[58] M. Koutek. *Scientific Visualization in Virtual Reality: Interaction Techniques and Application Development*. PhD thesis, Delft University of Technology, 2003.

[59] J. Krüger, P. Kipfer, P. Kondratieva, and R. Westermann. A particle system for interactive visualization of 3D flows. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):744–756, November 2005.

[60] D. A. Lane. UFAT: A particle tracer for time-dependent flow fields. In *Proc. IEEE Visualization*, pages 257–264, 1994.

[61] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[62] R. S. Laramee, H. Hauser, L. Zhao, and F. H. Post. Topology-based flow visualization, the state of the art. In *Topology-based Methods in Visualization*, pages 1–19. Springer-Verlag, 2007.

[63] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. Computer Graphics and Interactive Techniques*, pages 163–169, 1987.

[64] B. Lucas, G. D. Abram, N. S. Collins, D. A. Epstein abd D. L. Gresh, and K. P. McAuliffe. An architecture for a scientific visualization system. In *Proceedings of IEEE Visualization*, pages 107–114, October 1992.

[65] G. Mallinson. Cfd visualisation: challenges of complex 3d and 4d data fields. *International Journal of Computational Fluid Dynamics*, 22(1–2):49–59, January 2008.

[66] N. Max, R. Crawfis, and C. Grant. Visualizing 3D velocity fields near contour surfaces. In *Proc. IEEE Visualization*, pages 248–255, 1994.

[67] T. McLoughlin, R. S. Laramee, R. Peikert, F. H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. In *Eugraphics 2009, State of the Art Reports*, pages 73–92, 2009.

[68] C.-H. Moeng. A large-eddy-simulation model for the study of planetary boundary-layer turbulence. *Journal of the Atmospheric Sciences*, 41(13):2052–2062, 1984.

[69] J. D. Mulder, J. J. van Wijk, and R. van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15(1):119–129, 1999.

[70] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.

[71] J. F. Oliveira and B. F. Buxton. PNORMS: Platonic derived normals for error bound compression. In *Proc. ACM VRST*, pages 324–333, 2006.

[72] D. Overby, Z. Melek, and J. Keyser. Interactive physically-based cloud simulation. In *Proceedings of Pacific Graphics*, pages 469–470, 2002.

[73] J. D. Owens, D. Luebke, N. Govindaraju, Ma. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, 2005.

[74] J. Peng, C. S. Kim, and C. C. J. Kuo. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16(6):688–733, 2005.

[75] F. H. Post and T. van Walsom. Fluid flow visualization. In *Focus on Scientific Visualizaton*, pages 1–40. Springer Verlag, 1993.

[76] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualization: Feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[77] B. Preim and D. Bartz. *Visualization in Medicine: Theory, Algorithms, and Applications*. Morgan Kaufmann, 2007.

[78] B. Purnomo, J. Bilodeau, J. D. Cohen, and S. Kumar. Hardware-compatible vertex compression using quantization and simplification. In *Proc. Graphics Hardware*, pages 53–61, 2005.

[79] F. Reinders, F. H. Post, and H. Spoelder. Visualization of time-dependent data using feature tracking and event detection. *The Visual Computer*, 17(1):55–71, 2001.

[80] K. Riley, D. Ebert, C. Hansen, and J. Levit. Visually accurate multi-field weather visualization. In *Proc. IEEE Visualization*, pages 279–286, 2003.

[81] T. Ropinski and B. Preim. Taxonomy and usage guidelines for glyph-based medical visualization. In *Proc. of SimVis*, pages 121–138, 2008.

[82] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proc. ACM SIGGRAPH*, pages 343–352, 2000.

[83] M. Schirski, C. Bischof, and T. Kuhlen. Exploring flow fields with GPU-based stream tracers in virtual environments. In *Eurographics 2006*, pages 115–118, 2006.

[84] M. Schirski, C. Bischof, and T. Kuhlen. Interactive exploration of large data in hybrid visualization environments. In *Virtual Environments 2007*, pages 69–76, 2007.

[85] J. Schpok, J. Simons, D. Ebert, and C. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 160–166, 2003.

[86] W. Schroeder, K. M. Martin, and W. E. Lorensen. *The visualization toolkit: an object-oriented approach to 3D graphics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[87] A. P. Siebesma, C. S. Bretherton, A. Brown, A. Chlond, J. Cuxart, P. G. Duynkerke, H. Jiang, M. Khairoutdinov, D. Lewellen, C.-H. Moeng, E. Sanchez, B. Stevens, and D. E. Stevens. A large eddy simulation intercomparison study of shallow cumulus convection. *Journal of the Atmospheric Sciences*, 60(10):1201–1219, 2003.

[88] C. Silva, Y.-J. Chiang, W. Corrêa, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. Technical Report UCRL-JC-150434-REV-1, Lawrence Livermore National Laboratory, 2003.

[89] D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. *IEEE TVCG*, 3(2):129–141, 1997.

[90] N. J. A. Sloane, R. H. Hardin, and W. D. Smith. Spherical coverings. Published electronically at http://www.research.att.com/∼njas/coverings, 1997.

[91] J. Smagorinsky. General circulation experiments with the primitive equations: I. the basic equations. *Mon. Weather Rev.*, 91(3):99–164, 1963.

[92] G. Sommeria. Three-dimensional simulation of turbulent processes in an undisturbed trade-wind boundary layer. *Journal of Atmospheric Sciences*, 33(2):216–241, 1976.

[93] R. R. Springmeyer, M. M. Blattner, and N. L. Max. A characterization of the scientific data analysis process. In *Proceedings of IEEE Visualization*, pages 235–242, October 1992.

[94] D. Stalling, M. Westerhoff, and H. C. Hege. Amira: A highly interactive system for visual data analysis. In *The Visualization Handbook*, pages 749–767. 2005.

[95] J. Stam. Stable fluids. In *Proceedings of SIGGRAPH*, pages 121–128, 1999.

[96] P. N. Swarztrauber. Vectorizing the FFTs. *Parallel Computation*, pages 51–83, 1982.

[97] G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *Proc. ACM SIGGRAPH*, pages 123–132, 1998.

[98] G. Taubin, W. Horn, F. Lazarus, and J. Rossignac. Geometry coding and VRML. *Proceedings of the IEEE*, 96(6):1228–1243, 1998.

[99] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Trans. on Graphics*, 17(2):84–115, 1998.

[100] C. Touma and C. Gotsman. Triangle mesh compression. In *Proc. Graphics Interface*, pages 26–34, 1998.

[101] C. Upson, T. A. Faulhaber Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The application visualization system: a computational environmentfor scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[102] B. Vrolijk and F. H. Post. Fast out-of-core isosurface extraction and rendering of time-varying data sets. *Computers and Graphics*, 30(2):265–276, 2006.

[103] T. van Walsum. *Selective Visualization Techniques for Curvilinear Grids*. PhD thesis, Delft University of Technology, December 1995.

[104] N. Wang. Realistic and fast cloud rendering in computer games. In *Proc. SIGGRAPH Conference on Sketches & Applications*, page 1, 2003.

[105] C. Weigle and D. C. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *Proc. Symposium on Volume Visualization*, pages 103–110, 1998.

[106] D. Weiskopf. *GPU-Based Interactive Visualization Techniques*. Springer, 2006.

[107] L. J. Wicker and W. C. Skamarock. Time-splitting methods for elastic models using forward time schemes. *Monthly Weather Review*, 130(8):2088–2097, 2002.

[108] J. J. van Wijk. Flow visualization with surface particles. *IEEE Computer Graphics and Applications*, 13(4):18–24, 1993.

[109] H. Wright, R. H. Crompton, S. Kharche, and P. Wenisch. Steering and visualization: Enabling technologies for computational science. *Future Generation Computer Systems*, 2008.

[110] S. Ziegeler, R. J. Moorhead, P. J. Croft, and D. Lu. The metvr case study: meteorological visualization in an immersive virtual environment. In *Proc. IEEE Visualization*, pages 489–492, 2001.

[111] M. Zöckler, D. Stalling, and H.-C. Hege. Interactive visualization of 3D-vector fields using illuminated streamlines. In *Proc. IEEE Visualization*, pages 107–113, 1996.

List of Figures

List of Tables

Summary

Visualizing Cumulus Clouds in Virtual Reality

This thesis focuses on interactively visualizing, and ultimately simulating, cumulus clouds both in virtual reality (VR) and with a standard desktop computer. The cumulus clouds in question are found in data sets generated by Large-Eddy Simulations (LES), which are used to simulate a small section of the atmosphere over a period of several hours. These data sets are large, 3D, multi-variate, and time-varying, which pose several difficult visualization challenges. In order to overcome these challenges and gain insight into such complex data, several techniques are developed and employed together. At a high level, the research presented in this thesis can be divided into four categories: analyzing and visualizing interesting features in the data, giving the user sufficient control over the visualization, keeping the visualization interactive, and interactively simulating the cumulus clouds.

The first step to understanding the data was to find and track the features, i.e. the clouds, in the data. Through the use of a connected component labeling algorithm, individual clouds in the data could be identified. These clouds were then visualized in a VR visualization environment, which allowed atmospheric scientists to visually identify interesting clouds for further study.

The next step along the way was to improve the visualization experience. One aspect of this was to develop data "reprocessing". This allowed the atmospheric scientists to generate derived data from the raw simulation data for inclusion in the visualization environment. Another aspect of this was to improve the user interface with a more intuitive interaction technique and through the use of familiar 2D widgets.

The next challenge to address was the data access bottleneck. In order to move more data from disk to main memory and from main memory to the GPU, a lossy vector compression method based on quantization is developed. By analyzing and bounding the angular error

introduced by quantization, unit vectors can be quantized using 16 bits with less than 0.4 degrees of angular error. This can reduce the size of mesh geometry, and, when also quantizing vector length, can be used to compress vector fields.

In order to help understand the relationship between the clouds and the air around them, interactive particle tracing was the next research area. Using the power of the GPU, millions of particles could be advected interactively. These particles could be visualized as particles in different visual styles or as flow curves such as stream-lines or streak-lines. By combining vector-field compression with a multi-resolution advection scheme, users could interactively seed and advect particles around selected clouds of interested through time.

Most of the techniques developed in this thesis have been integrated into Cloud Explorer, which is an experimental VR visualization platform for interactively visualizing and studying the cumulus cloud data. Several techniques have also been integrated into stand-alone applications.

The final research area addressed in this thesis was interactive simulation of the cumulus clouds. GALES, a GPU-based, atmospheric, Large-Eddy Simulation, was the result of this effort. GALES runs 16x faster than the Dutch Atmospheric Large-Eddy Simulation, which is a Fortran-based LES, while maintaining comparable numerical accuracy. This speedup enables GALES to interactively run and visualize simulations that fit into GPU memory. This opens new and exciting possibilities for future computational steering research.

Eric J. Griffith

Samenvatting

Het Visualiseren van Stapelwolken in Virtual Reality

Dit proefschrift richt zich op het interactief visualiseren en uiteindelijk simuleren van stapelwolken zowel in virtual reality (VR) als met een standaard desktop computer. Deze stapelwolken bevinden zich in data sets die gegenereerd zijn door Large-Eddy Simulaties (LES), waarmee kleine stukjes van de atmosfeer voor een periode van meerdere uren gesimuleerd kunnen worden. Deze data sets zijn zeer groot, 3D, multivariate, en tijdsvariërend, en ze stellen meerdere moeilijke visualisatie uitdagingen. Om deze uitdagingen te overwinnen en inzicht in de data te krijgen, zijn verschillende technieken ontworpen en in combinatie met elkaar toegepast. Op een hoog niveau kan het onderzoek beschreven in dit proefschrift worden onderverdeeld worden in vier categorieën: het analyseren en visualiseren van interessante verschijnselen in de data, de gebruiker voldoende controle geven over de visualisatie, de visualisatie interactief houden, en het interactief simuleren van de stapelwolken zelf.

De eerste stap naar het begrijpen van de data was om de verschijnselen, d.w.z. de stapelwolken, terug te vinden in de data en deze vervolgens te kunnen volgen. Door het gebruik van een algoritme voor het labelen van samenhangende componenten konden de individuele wolken geïdentificeerd worden. Daarna werden deze wolken gevisualiseerd in een VR visualisatie omgeving, waarmee atmosferische onderzoekers visueel interessante wolken konden uitkiezen om verder te bestuderen.

De volgende stap was het verbeteren van de visuele ervaring. Een onderdeel hiervan was het ontwikkelen van een data “herbewerking” waarmee atmosferische onderzoekers afgeleide data konden genereren op basis van de originele simulatie data voor gebruik in de visualisatie omgeving. Een tweede onderdeel was het verbeteren van de gebruikersinterface met een meer intuïtieve interactie techniek en het gebruik van bekende 2D widgets.

De volgende uitdaging om aan te pakken was het knelpunt van toegang tot de data. Om

data sneller te verplaatsen van de schijf naar geheugen en van geheugen naar de GPU is een niet-exact omkeerbare vector compressie methode ontwikkeld op basis van vectorkwantisastie. Door het analyseren en begrenzen van de hoekfout geïntroduceerd door kwantisatie, kunnen eenheidsvectoren gekwantiseerd worden met 16 bits en minder dan 0.4 graden van hoekfout. Dit kan de grootte van de mesh geometrie verkleinen en als de vector lengte ook gekwantiseerd wordt, de grootte van vector velden ook comprimeren.

Om de relatie tussen de wolken en de lucht eromheen te begrijpen werd interactief particle tracing het volgende onderzoeksonderwerp. Met gebruik van de GPU konden miljoenen deeltjes interactief geadvecteerd worden. Deze deeltjes kunnen zowel als deeltjes met verschillende visuele stijlen en als stromingskrommen zoals stroomlijnen of streaklijnen worden weergegeven. Met de combinatie van vector veld compressie en een multi-resolutie advectie schema kunnen gebruikers interactief deeltjes zaaien en laten advecteren in de tijd door wolken.

De meeste van de technieken ontwikkeld in dit proefschrift zijn geïntegreerd in Cloud Explorer. Cloud Explorer is een experimenteel VR visualisatie platform voor het visualiseren en bestuderen van stapelwolk data. Meerdere technieken zijn ook in afzonderlijke toepassingen geïntegreerd.

Het laatste onderzoeksonderwerp dat in dit proefschrift aan de orde komt is het interactief simuleren van stapelwolken. GALES, een GPU-gebaseerde atmosferische Large-Eddy Simulatie was het resultaat van deze inspanningen. GALES draait 16x sneller dan de Nederlandse Atmosferische Large-Eddy Simulatie (DALES), dat een Fortran-gebaseerde LES is. Deze toename in snelheid laat GALES simulaties die binnen het GPU geheugen passen interactief draaien en visualiseren. Dit biedt nieuwe mogelijkheden voor toekomstig computational steering onderzoek.

Eric J. Griffith

Curriculum Vitae

I was born on April 3, 1980 at the Naval Hospital in Long Beach, California, USA, which has since been replaced by a shopping mall. I completed my secondary school education in 1998 at Portsmouth High School in Portsmouth, Rhode Island.

In the fall of 1998, I enrolled in Rensselaer Polytechnic Institute (RPI) to pursue a bachelor's degree in computer science. After taking one year off from my studies in 2000 to work for an internet startup company, I completed my undergraduate education in the spring of 2003 with the expected bachelor's degree in computer science and an unexpected second bachelor's degree in mathematics.

I then immediately began working on my master's degree in computer science at RPI, which I completed in the summer of 2004 with a thesis entitled, "A Design Tool for Specifying Implicit Algebraic Blend Surfaces". During the year I worked on my M.S., I also taught a "Programming in Perl" course for two semesters and developed an interest in the algorithmic challenges posed by digital microfluidic systems.

In August of 2004, I began my PhD studies at TU Delft, the end result of which is this thesis. My PhD research was conducted under the guidance and supervision of Frits Post and with Erik Jansen as my promotor. During my time at TU Delft, I also collaborated with Gerwin de Haan on some papers that fell outside the scope of this thesis. After leaving TU Delft in November of 2008, I began working at Petrotechnical Data Systems in Rijswijk on visualization software for seismic data for use by Royal Dutch Shell.