# การปรับตารางกริดแบบปรับตัวได้โดยใช้ต้นไม้อัฐภาคสำหรับการจำลองควัน

นายรินชัย บรรลุทางธรรม

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

# ADAPTIVE GRID REFINEMENT USING VIEW-DEPENDENT OCTREE FOR SMOKE SIMULATION

Mr. Rinchai Bunlutangtum

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2011

| | |
|---|---|
| Thesis Title | Adaptive Grid Refinement Using View-Dependent Octree for Smoke Simulation |
| By | Mr. Rinchai Bunlutangtum |
| Field of Study | Computer Engineering |
| Thesis Advisor | Assistant Professor Pizzanu Kanongchaiyos, Ph.D. |

---

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

…….. Dean of the Faculty of Engineering
(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

…….. Chairman
(Assistant Professor Thanarat Chalidabhongse, Ph.D.)

…….. Thesis Advisor
(Assistant Professor Pizzanu Kanongchaiyos, Ph.D.)

…….. Examiner
(Nuttapong Chentanez, Ph.D.)

…….. External Examiner
(Chutisant Kerdvibulvech, Ph.D.)

รินชัย บรรลุทางธรรม : การปรับตารางกริดแบบปรับตัวได้โดยใช้ต้นไม้อัฐภาคสำหรับการจำลองควัน. (ADAPTIVE GRID REFINEMENT USING VIEW-DEPENDENT OCTREE FOR SMOKE SIMULATION)
อ. ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ. ดร. พิษณุ คนองชัยยศ, 90 หน้า.

การจำลองควันหรือของไหลประเภทต่างๆซึ่งเป็นปรากฏการณ์ตามธรรมชาติที่มีความซับซ้อนสูงและใช้เวลาในการคำนวนมาก คุณภาพและความสมจริงที่ได้จึงขึ้นอยู่กับเวลาในการคำนวณเป็นปัจจัยควบคุมหลัก ในงานวิจัยก่อนหน้าได้มีการนำเสนอการปรับแต่งตาราง (Grid Refinement) เพื่อลดจำนวนของโหนดในโดเมนจำลอง (Simulation Domain) ลง ผลลัพท์ก็คือการจำลองใช้เวลาประมวลผลสั้นลง และภาพที่ได้มีรายละเอียดสูงขึ้น อย่างไรก็ตามเราพบว่าตารางการจำลอง (Simulation Grid) ยังสามารถลดรูปเพื่อการจำลองที่เร็วยิ่งขึ้นได้ โดยการนำข้อมูลจากมุมมองกล้องมาพิจารณาเป็นส่วนหนึ่งในขั้นตอนการปรับแต่งตาราง (Grid refinement). แนวคิดก็คือ ควันหรือของไหลที่อยู่ไกลจากกล้อง ย่อมมีขนาดที่เล็กลงเสมอ เราจึงสามารถลดความละเอียดของไหลที่อยู่ไกลลงได้โดยที่ไม่ทำให้คุณภาพของภาพผลลัพท์ที่ได้ด้อยลง งานวิจัยนี้จึงได้นำเสนอวิธีการปรับแต่งตารางตามมุมมองกล้อง (View-Dependent Adaptive Grid Refinement) โดยพิจารณาจาก ระยะห่างระหว่างของไหลกับกล้อง (fluid-camera distance), องศารับภาพ (viewing angle), ขนาดภาพผลลัพท์ (output resolution) ซึ่งสามารถนำไปใช้เพิ่มความเร็วในการประมวลผลการจำลองควัน และของไหลประเภทต่างๆ โดยที่ยังสามารถคงคุณภาพและรายละเอียดของภาพผลลัพท์สูงสุดได้ นอกจากนี้เรายังได้นำเสนอการนำอนุภาคมาใช้ใช้ในการแสดงผล เพื่อปรับปรุงภาพผลลัพท์ที่ได้ให้ดียิ่งขึ้น
วิธีการที่เราได้นำเสนอทั้งหมดในงานวิจัยนี้ มุ่งเน้นให้สามารถนำไปประยุกต์ใช้ได้จริงในทางปฏิบัติ ทั้งกับการจำลองควันและการจำลองของไหลประเภทต่างๆ

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต.....
สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก……..
ปีการศึกษา 2554.....

# # 5370477921 : MAJOR COMPUTER ENGINEERING

KEYWORDS : FLUID DYNAMICS , SIMULATION , NATURAL PHENOMENA

RINCHAI BUNLUTANGTUM : ADAPTIVE GRID REFINEMENT USING VIEW-DEPENDENT OCTREE FOR SMOKE SIMULATION. ADVISOR : ASST.PROF. PIZZANU KANONGCHAIYOS, Ph.D., 90 pp.

Computational cost is one of the major problems in animating smoke. Recently, adaptive grid refinement using octree structure has been proposed which is a successful method for reducing the computational cost of detail-preserving fluid simulation. Although octree grid is optimized for details, viewing angle is not addressed. Smoke distant from the viewing screen or beyond the viewing frustum, which usually has less visual attention and is unnecessary for high-detail simulation, can be optimized for speed. However, applying such view-dependent optimization to the octree grid directly might cause animation artifacts and loss in natural fluid behaviors. In this thesis, we have presented a method for view-dependent adaptive grid refinement, extending the traditional octree grid by considering the viewing frustum, as well as the variation in fluid quantities as criteria for grid refinement. In our method, refinement conditions with adaptive thresholds are proposed to optimize the grid for both viewing angle and details. The proposed method preserves visual details and fluid behaviors which allows high-detail smoke animations with a relatively less computational cost. In addition, particles, which are more flexible to conform to obstacle-fluid boundaries, are integrated to enhance animation and reduce artifacts caused by dynamic refinements. Overall, the method provides a flexible framework for optimization that can be applied for various fluid simulations.

Department : Computer Engineering Student's Signature .....

Field of Study : Computer Engineering Advisor's Signature .....

Academic Year : 2011

# Acknowledgements

I would like to thank my thesis advisor, Asst. Prof. Dr. Pizzanu Kanongchaiyos for his advices and assistances, my thesis committee, Asst.Prof. Dr.Thanarat Chalidabhongse, Dr. Nuttapong Chentanez and Dr. Chutisant Kerdvibulvech for their comments and suggessions. We also would like to thank everyone at CUCG for the valuable discussions. I would also like to thank Blizzard for the edutainment. Finally, I deeply wish to thank my parents for their love, understanding and invaluable supports throughout my graduate study.

This research was partially funded by the TRF-Master Research Grants (MAG) MRG-WI535E005, Yodia Multimedia Co.,Ltd and was in part supported by the CUCP Academic Excellence Scholarship from Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University.

# Contents

Page

Page

# List of Tables

Page

## List of Figures

Page

Page

# CHAPTER 1

# Introduction

Figure 1.1 Smoke simulation generated by ®Blender 2.5

## 1.1 Introduction and Problem State

Physically-based fluid simulation is a widely used technique for animating smoke, fire and other fluid phenomena. Even though physically-based simulation can generate physically realistic results as well as stunning effects, which are impossible for the artist to animate manually frame-by-frame, it usually comes with high computational cost as a trade-off. Since the main focus of graphics and realism is on generating plausible visual effects rather than accuracy, a challenging topic is how to minimize the computational cost, while being able to obtain as highly detailed animations as possible.

Grid-based simulation is a common approach for physically-based fluid animations. [1-4] use fixed uniform grid for animating smoke. The method works well for coarse grids, but since animation is becoming more and more in demand in the special effects industry, animating on a fixed uniform grid in a larger domain, or refinement for higher detail is not scalable, because of its high computational cost consumption.

To address this, several adaptive grid refinement were introduced to optimize the simulation. [5,6] replace the traditional fixed uniform grid with an adaptive non-uniform grid using an octree structure. Adaptive grid refinement on an octree structure has been successful in optimizing the simulation. The grid is subdivided only in some specific areas that require higher detail and are merged to save computational

cost when details are no longer necessary. The methods can be speed up by utilizing high performance parallel computing of graphic hardware such as that implemented in [7]. Overall, these adaptive grid techniques allow the capturing of small visual details at a lower computational cost compared to the earlier fixed uniform grid.

Beside an octree grid, [8] propose a view-dependent grid to optimize the simulation based on viewing angle. Instead of being constructed on Cartesian coordinates as usual, the grids are constructed on the transformed polar coordinate that is most fit for the viewing angle. With a view-dependent grid, fluid details gradually decrease proportionally as the distance from the camera increases, thus providing constant screen-space detail across the simulation domain. In addition, fluid that is far beyond the visible scene is not computed. However, unlike the octree grid, view-dependent grid is subdivided uniformly on the transformed coordinates; thus, grid size is fixed and not adaptive for detail optimization.

Up until now, fluid simulations are performed faster with current hardware and technology; nevertheless, they are not fast enough as people always expect animations with higher details. This research proposes an optimization improvement for smoke simulation on an octree grid that allows faster simulations with details preserved in the visual result.

### 1.2 Objectives of Study

In this research, we propose a method to animate smoke that consumes less computational cost but still preserves the visual results. The objective is to minimize unnecessary computational cost for speed while still preserving any small visual details and natural behaviors of smoke that usually disperse when the optimization is applied.

We have realized that a large simulation domain contains lots of distant fluids which usually have less visual attention; thus, small-scale details can be neglected in these regions, as well as hidden or distant smoke. Since the octree grid is optimized for details but not for viewing angle whereas the view-dependent grid is optimized for viewing angle but not for details, we present an improved method for

animating smoke on octree grid in order to additionally refine the octree grid by a view-dependent level-of-detail.

### 1.3 Scope of Study

We have studied only in optimizing a grid-based smoke simulation which assumes smoke as incompressible and homogenous fluids. Our consideration is to reduce unnecessary computational cost while still remaining physically realistic and preserving visual details. The proposed method is based on a view-dependent level-of-detail and a structure of octree grid which is an adaptive structure over space and time.

### 1.4 Expected Benefits

The proposed method should improve the simulation performance e.g., consume relatively lower computational cost and extend the possibility for higher detail and larger domain simulation. The method should be beneficial for various computer graphics applications i.e., game industries, movie special effects and advertisements. In addition, the adjustable thresholds in our method allow the user to flexibly weight between details and computational cost to compromise for various simulation scenarios. Also, the method is fast and easily implemented, yet capable to be integrated as an extension of a current grid-based solver.

### 1.5 Publications

R. Bunlutangtum and P. Kanongchaiyos, Adaptive Grid Refinement Using View-Dependent Octree for Grid-Based Smoke Simulation, The 4th International Conference on Motion in Games, LNCS 7060, (2011):204-215.

R. Bunlutangtum and P. Kanongchaiyos, Enhanced View-Dependent Adaptive Grid Refinement for Animating Fluids, Accepted to be published in The 10th International Conference on Virtual Reality Continuum and Its Applications in Industry, (2011).

R. Bunlutangtum and P. Kanongchaiyos, Smoke Simulation with View-Dependent Adaptive Grid Refinement, Accepted to be published in The 4th International Conference on Computer and Electrical Engineering, (2011).

## 1.6 Definition

1 **Domain**: a defined area for the simulation, containing fluids, obstacles and/or empty spaces. The domain is usually defined as three-dimensional rectangular cuboids.

2 **Cell**: a discrete computational unit that represents continuous quantities of fluid, usually defined as a cubic volume. A domain is composed of cells which are associated with the grid resolution; higher resolution means more number of cells per unit volume. Each cell can be fluid, obstacle or empty cell.

3 **Element**: a single unit of an array. For example, an array with dimensions of 2 x 2 x 2 contains 8 elements.

4 **Grid**: refers to a structure made up of a series of intersecting vertical and horizontal axes used to structure grid-based fluid content. Grids divide space into cubic cells, with each cell storing discrete fluid quantities i.e., pressure, velocity, and density.

5 **Fixed Uniform Grid**: a grid containing cells that all have the same size and with uniformly positioning throughout space and time. Number of cells does not change over time.

6 **Adaptive Grid**: a grid containing cells with non- uniformity in size. Their size and position are usually determined by the octree structure. Each cell has unpredictable adjacent neighboring cells. Number of cells can change adaptively over time.

7 **Grid Refinement**: an optimization process for grid-based simulation consists of two actions i.e., *grid subdivision*: refining the grid to obtain details and *grid merging*: coarsening the grid to reduce the computational cost.

8 **View-Dependent Grid Refinement:** grid refinement that consider a viewing frustum, positions and perspective as criteria for subdivision and merging.

9 **Rendering**: a process for generating an image or motion picture from 3D space, or in another word, 3D fluid volume is projected into 2D via rendering.

1 0 **Frame**: a still image or a snapshot of an animation. Basically, an animation is composed of several frames in a sequence.

1 1 **Viewing Frustum**: the region of 3D space to be displayed on the screen. The shape of the viewing frustum varies depending on what kind of camera lens is being used, but typically a rectangular pyramid is used for perspective view. The viewing frustum is bounded by the field of view (FOV) of the camera, a near clipping plane and a far clipping plane (see Figure 1.2).

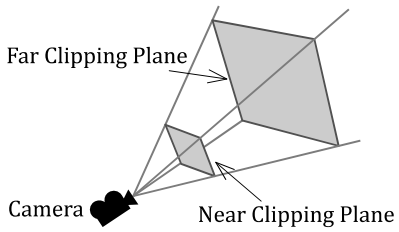



Figure 1.2 Viewing Frustum

1 2 **Field of View (FOV)**: or angle of view, describes the angle of projection by the camera's lens onto the focal plane. A larger degree FOV results in a wider perspective view. A camera's angle of view can be measured horizontally, vertically, or diagonally.

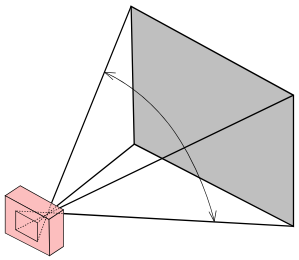


Figure 1.3 Field of View

## 1.7 Research Procedure

Our research is planned for an 8-month period starting in December 2010 and ending in August 2011. The process can be summarized into 7 stages as follows.

| Task | Start | Duration (months) | 12/10 | 1/11 | 2/11 | 3/11 | 4/11 | 5/11 | 6/11 | 7/11 | 8/11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Theory and literature reviews | Dec 10 | 4 | | | | | | | | | |
| Algorithm design | Feb 11 | 3 | | | | | | | | | |
| Application design | Apr 11 | 2 | | | | | | | | | |
| Application Implementation | May 11 | 2 | | | | | | | | | |
| Result evaluation | June 11 | 2 | | | | | | | | | |
| Conclusion | July 11 | 1 | | | | | | | | | |
| Thesis report | July 11 | 2 | | | | | | | | | |

Table 1.1 Research Procedure

# CHAPTER 2

# Literature Reviews

Fluid phenomena such as droplets splashing, rising smoke and fire are some of the stunning natural effects that today can be artificially generated by computer graphics and is hard to distinguish the difference between a “real” one and a “fake” one. There are several techniques to animate such fluid phenomena. In general, fluid simulation can be categorized into physically-based and non physically-based methods. Physically-based methods have benefit over non-physically based methods in several ways i.e., produce physically realistic and allow high-detail fluid simulation which are impossible for the artists to animate manually frame by frame. In this section, we review four different types of physically-based fluid simulation.

## 2.1 Grid Based Method

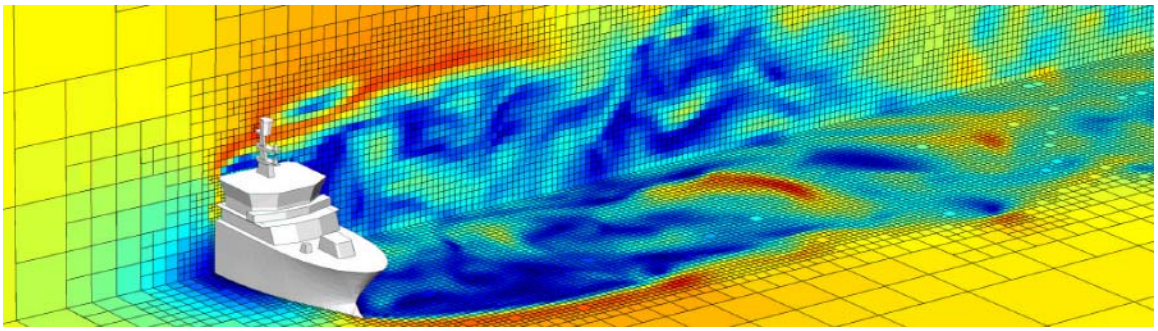


Figure 2.1 Grid-based fluid simulation using octree structure (image from: [9])

Grid based fluid simulation was first introduced in computer graphics by [1], but because their model uses an explicit integration scheme, their simulations are only stable if the time step is chosen small enough; therefore, the simulation is relatively slow. [2] proposed an unconditionally stable simulation by using semi-Lagrangian advection scheme with implicit solvers. However, numerical dissipation was severe in this method. [3] introduced a vorticity confinement term to model the small scale rolling features characteristic of smoke to compensate the numerical dissipation caused by the implicit model. The methods have been extended to other fluid phenomena such as fire [10], explosion [11], viscoelastic materials [12] and bubbles [13]. These previous works

can be categorized as fixed uniform grid as the grid are subdivided uniformly and their cell's position are fixed throughout the simulation domain.

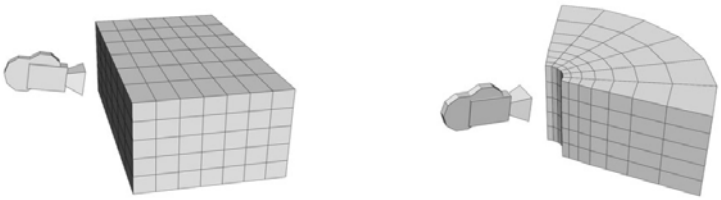


Figure 2.2 Left: Fixed Uniform grid. Right: View-dependent grid. (image from: [8])

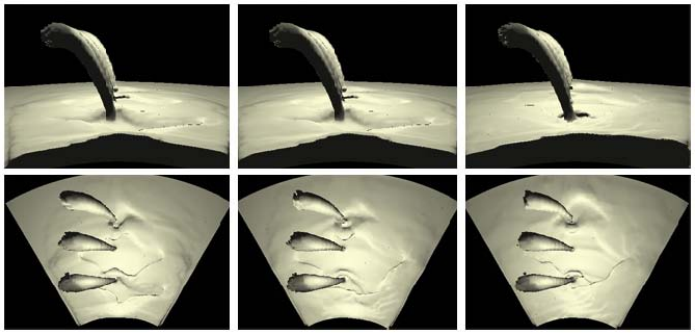


Figure 2.3 A view-dependent water simulation on rendering views (above) and top views (below) (image from: [8])

A fixed uniform grid is simple and straightforward to implement compared to others. It works well for coarse grids but unfortunately, when applied to larger domains or higher grid resolutions, the simulation encounters a scalability problem since it consumes a high computational cost due to its uniform grid size. Adaptive grids, on the other hand, are an alternative method that consumes relatively lower computational cost by dynamically reducing and increasing the number of cells in domain over time. [14,15] introduced adaptive mesh refinement (AMR) for compressible flows while [9] presented an adaptive mesh method using an octree. In computer graphics, the octree data structure has been proposed for adaptive grid refinement by [5] and asymmetric octree by [6], which results in detail optimization, while [8] propose a method that is optimized for viewing angle by using a view-dependent grid, decreasing fluid details as distance from the camera increases, thus providing constant screen-

space detail across the simulation. However, contrary to the adaptive grid using octree, view-dependent grid is subdivided uniformly, thus, lacking detail optimization.

One of the main problems with octree grids is their dynamic and irregular structure, which is contrary to the design of graphics hardware. [7] present a problem decomposition for parallel processing that takes advantage of the graphics hardware while reducing expensive hierarchy traversals on non-uniform and adaptive octree grids.

Figure 2.4 Smoke simulation using octree grid refinement on GPU (image from: [7])

## 2.2 Particle Based Method

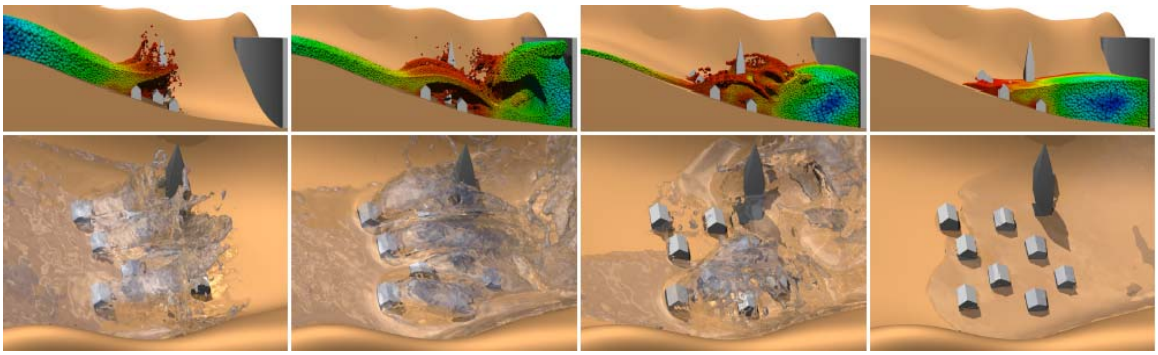


Figure 2.5 Water simulation using SPH with Adaptive Kernel (image from: [16])

Smoothed Particle Hydrodynamics (SPH) is a commonly-used particle based method. With SPH, fluid is simulated on a particle system by sub-sampling a set of elements called particles. Each particle contains fluid attributes i.e., mass, density, velocity, and pressure. Fluid values and derivatives of fluid quantities at arbitrary positions are approximated by a set of neighboring discrete particles with a specified function called a "*smoothing kernel*".

SPH was first introduced in 1977 by [17] for astrophysical simulations i.e., large scale structure in the universe, galaxy formation, supernova and solar formation. Later, many researches on computer graphics [18-22] have used the SPH method to simulate fluid flow. The efficiency of SPH has been improved by [16,23,24] with a technique called "*Adaptive SPH*"; adaptively adjusting kernel size to reduce computational time in dense areas and preserve details in sparse areas.

SPH easily demonstrates the turbulent splashing flows and catches small details of fluid phenomena such as bubbles and foams. Furthermore, the demands of computational resources of SPH with a moderate number of particles are generally less than grid based or LBM counterparts. Thus, several fluid phenomena in games or other interactive system are simulated using the Lagrangian method. However, the stability, accuracy and speed of the SPH method largely depend on the selected smoothing kernel. Also, SPH is hardly guaranteed for its incompressibility.

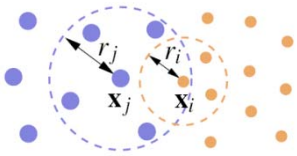



Figure 2.6 Adaptive Kernel (image from: [16])

### 2.3 Lattice Boltzmann Method

[25] introduced the Lattice Boltzmann method (LBM) into the computer graphics community. LBM is a relatively new approach to approximating the Navier-Stokes equations. Unlike traditional CFD methods, which solve the governing equations of macroscopic properties (i.e., mass, momentum and energy), the LBM is based on microscopic models and mesoscopic kinetic equations (the Lattice Boltzmann equation). The fundamental idea is to construct simplified kinetic models that incorporate the microscopic and mesoscopic physical processes so that the macroscopic averaged properties obey the desired macroscopic equations (the Lattice Boltzmann equation converges to the Navier-Stokes equation).

Figure 2.7 Water simulation using Lattice Boltzmann Method (image from: ®Blender)

LBM provides a relatively easy and consistent way to incorporate the underlying microscopic interactions especially for multiphase flows with moving and deformable interfaces. Moreover, LBM has several advantages over other conventional physically-based fluid animation methods, such as in dealing with complex boundaries and parallelizing the algorithm. The major drawbacks of LBM are its poor scalability and small time steps. Moreover, the time step must be small enough to ensure the stability of the simulation.

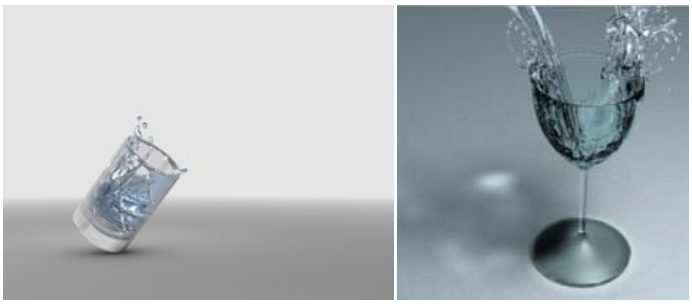


Figure 2.8 Water simulation using Lattice Boltzmann Method (image from: ®Blender)

## 2.4 Tetrahedral Mesh Method

Fluid simulation using an unstructured tetrahedral mesh has been proposed in computer graphics field by [26], [27] with a velocity-based approach and [28] with a vorticily-based approach while [29] presented a two-way coupling of fluid and rigid bodies. The combination of unstructured tetrahedral domains and dynamic remeshing at each time step creates a flexible environment for creating complex scenes. Later, [30] presented a technique called "*isosurface stuffing*" for grading mesh resolution across the fluid boundary, which allows for effective conformability to complex

boundaries. Moreover, they have presented a thickening strategy for reducing volume loss and artifacts of disappearing droplets or sheets during simulation.

In general, tetrahedral meshes conform well to irregular boundaries and their size can be adjusted to optimize the simulation. These benefits make tetrahedral meshes a flexible and effective method for simulation with complex environments. However, simulation usually encounters complications for free surfaces and moving boundaries because the meshes must track the movement of those surfaces. Simulation also needs a specific scheme to prevent volume loss or artificial damping that usually occurs in droplets, filaments or thin sheets.

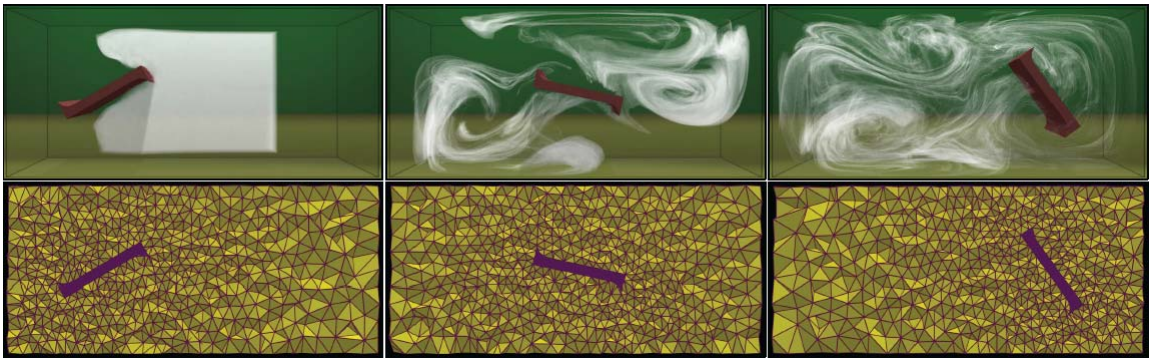


Figure 2.9 Fluid Animation with Dynamic Meshes (image from: [29])

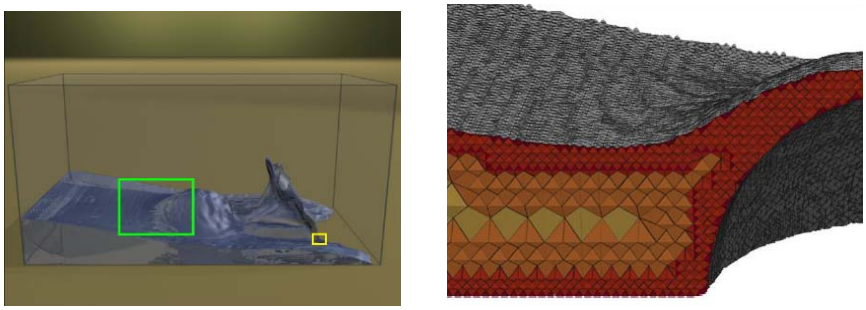


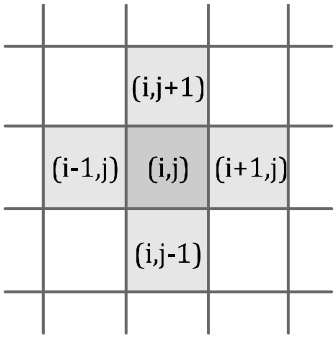
Figure 2.10 A cutaway view showing graded tetrahedral meshes (image from: [30])

# CHAPTER 3

# Theories

In this chapter, we mention some of the physics equations that describe the characteristic nature of fluid motions and their simplified terms for smoke animations, followed by an introduction to the common solving methods for grid-based smoke simulation and an overview of the dynamic grid refinement methods.

## 3.1 Differential operations




Vector Calculus is a branch of mathematics concerned with differentiation and integration of vector fields primarily in three-dimensional Euclidean space ($\mathbb{R}^3$). Vector calculus studies various differential operators defined on scalar or vector fields, which are typically expressed in terms of the "*del operator*" ($\nabla$). The four most important differential operations in vector calculus are summarized in Table 3.1, where $p$ denotes a scalar field and $\boldsymbol{u}$ denotes a two-dimensional vector field: $\boldsymbol{u} = (u, v)$. The meaning of four differential operations in Table 3.1 is described below.

1. **Gradient :** Measures the rate and direction of change in a scalar field.
2. **Divergence :** Measures the magnitude of a source or sink at a given point in a vector field.
3. **Laplacian :** Measures the rate at which the average value of $p$ over spheres centered at $\boldsymbol{x} = (x, y)$ deviates from $p(\boldsymbol{x})$ as the radius of the sphere grows.
4. **Curl :** Measures the tendency to rotate about a point in a vector field.

Note that the Laplacian operator is a composition of the divergence and gradient operations, defined as the divergence of gradient: $\nabla^2 p = \nabla \cdot \nabla p$. If the grid cells are square (that is, if $\partial x = \partial y$, which we assume for this article), the Laplacian simplifies to:

$$\nabla^2 p = \frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}}{(\partial x)^2}$$

| Operator | Definition | Finite Different Form |
|---|---|---|
| Gradient | $\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}\right)$ | $\nabla p = \left(\frac{p_{i+1,j} - p_{i-1,j}}{2\partial x}, \frac{p_{i,j+1} - p_{i,j-1}}{2\partial y}\right)$ |
| Divergence | $\nabla \cdot \boldsymbol{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ | $\nabla \cdot \boldsymbol{u} = \frac{u_{i+1,j} - u_{i-1,j}}{2\partial x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\partial y}$ |
| Laplacian | $\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}$ | $\nabla^2 p = \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\partial x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\partial y)^2}$ |
| Curl | $\nabla \times \boldsymbol{u} = \begin{vmatrix} i & j & k \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ u & v & w \end{vmatrix} = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial z}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial x}{\partial y}\right)$ | |

Table 3.1 Summary of differential operators in two dimensional vector fields

### 3.2 The Navier-Stokes Equations

The Navier-Stokes equations describe the motion of fluid, formulated by French physicist and engineer Claude-Louis Navier and Irish mathematician George Gabriel Stokes. The Navier-Stokes equations are a set of non-linear differential equations in terms of rate of change of fluid quantities over time. These equations are a combination of terms; each term defines individual fluid properties (i.e., advection, pressure, diffusion, viscosity and external forces). For computer graphics, fluid compressibility can be neglected due to its high computational cost and lack of important role in the simulation. This lead to a simpler form of Navier-Stokes equations, called "*The Incompressible Navier-Stokes equations*" or "*Euler Equations*":

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{Equation 3.1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{u} + \boldsymbol{F} \qquad \text{Equation 3.2}$$

Where $\upsilon$ is the kinematic viscosity, $\rho$ is the fluid density, $p$ is the scalar pressure field, $\boldsymbol{F}$ is external forces (or body forces), $\boldsymbol{u}$ is a velocity vector field: $\boldsymbol{u} = (x, v, w)$, and $\nabla$ is the vector operator of spatial partial derivatives : $\nabla = (\frac{\partial}{\partial \mathrm{x}}, \frac{\partial}{\partial \mathrm{y}}, \frac{\partial}{\partial \mathrm{z}})$.

The Navier-Stokes equations are obtained by imposing that fluid conserves both mass (Equation 3.1) and momentum (Equation 3.2). Equation 3.1 is *"Mass conservation"* or *"Continuity equation"*, states that fluid velocity field has *"zero divergence"* which notifies that the velocity flux that flow inward and outward at any infinitesimal volume should be equal. Equation 3.2 is a *"momentum conservation equation"* derived from Newton's second law: $\boldsymbol{F} = m\boldsymbol{a}$. The term on the left hand side, $\partial \boldsymbol{u}/\partial t$, is the rate of change of velocity with respect to time, defined by right hand side components i.e., advection, pressure, viscosity diffusion, and external forces as described below.

1. **Advection:** represents the *"self-advection"* of the velocity field, which means velocity causes the fluid to transport itself as well as its quantities such as density, temperature, pressure and velocity along with the flow. It is a time independent acceleration of the fluid with respect to space.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \ \ldots \ -(\nabla \cdot \boldsymbol{u})\boldsymbol{u} \quad \ldots$$

2. **Pressure Gradient:** states that the fluid should propagate from higher to lower pressure areas by the amount of its pressure difference.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \ \ldots \ -\frac{1}{\rho}\nabla p \quad \ldots$$

3. **Viscosity Diffusion:** determines how fast the fluid diffuses its velocity to surrounding neighbors. The parameter represents a kinematic viscosity of the fluid; thick fluid which has higher kinematic viscosity diffuses its velocity quicker and tends to flow slowly. Viscosity in gases is very low, thus in some case, this term can be negligible.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \ \ldots \ +\nu\nabla^2\boldsymbol{u} \quad \ldots$$

4. **External Forces:** or body forces are any other forces that affect the fluid movement. Gravity force, buoyancy force are external forces.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \ \ldots \ + \boldsymbol{F} \quad \ldots$$

Other fluid quantities are also described with similar Navier-Stokes equations. Here are the Navier-Stokes equations that describe density and temperature respectively.

$$\frac{\partial \boldsymbol{d}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{d} + k_d \nabla^2 \boldsymbol{d} + \boldsymbol{S} \qquad \text{Equation 3.3}$$

$$\frac{\partial T}{\partial t} = -(\boldsymbol{u} \cdot \nabla)T + k_T \nabla^2 T + H \qquad \text{Equation 3.4}$$

Equation 3.3 is a color density equation, where $\boldsymbol{d}$ is a vector field denotes the color density of smoke in alpha (transparency), red, green and blue respectively: $\boldsymbol{d} = \left(d_\alpha, d_r, d_g, d_b\right)$. Color density is advected (transport) along a velocity field and diffuses due to viscosity in a similar manner as the Navier-Stokes Equations that describe velocity. The first term is a density advection term. It states that density should follow the velocity field. The second term is a diffusion term, where $k_d$ denotes a viscosity constant of density. High viscosity means color density diffuses itself to its surroundings quicker. $\boldsymbol{S}$ is a color density added from external sources.

Equation 3.4 is the temperature equation, where $T$ is the fluid temperature, $k_T$ is a viscosity constant of temperature and $H$ is any temperature added from external sources. For the full form of the temperature equation, we refer the reader to [11].

### 3.3 Equations for Smoke Simulations

Smoke has its own characteristics and behaviors apart from other fluids; for instance, smoke does not have an exact boundary as liquid and smoke tends to diffuse away while liquid is clustering together. These unique characteristics and behaviors need a specific scheme for simulation. In this section, we have gathered the essential equations specified for smoke simulation.

For visual simulation, smoke can be assumed as an incompressible and homogeneous fluid. The viscosity term shown in Equation 3.2 can be neglected since the fluid motion is usually below the speed of sound and hence, the appearance of viscosity is dominated by the numerical dissipation [3]. The Navier-Stokes equations can be reduced to Euler equations as follows.

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{Equation 3.5}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \frac{1}{\rho}\nabla p + \boldsymbol{F} \qquad \text{Equation 3.6}$$

Equation 3.5 is a mass conservation equation (or called continuity equation) and Equation 3.6 is a momentum conservation equation. Note that fluid density $\rho$ is constant both in space and time since we assume smoke as an incompressible and homogeneous fluid.

The Navier-Stokes equations that describe density (Equation 3.3) and temperature (Equation 3.4) can neglect their viscosity terms as well. Below are their simplified equations.

$$\frac{\partial \boldsymbol{d}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{d} + \boldsymbol{S} \qquad \text{Equation 3.7}$$

$$\frac{\partial T}{\partial t} = -(\boldsymbol{u} \cdot \nabla)T + H \qquad \text{Equation 3.8}$$

Buoyancy is a fluid behavior that causes smoke to rise due to temperature and fall downwards due to gravity. [3] model these effects by defining an additional force that is directly proportional to the density and the temperature.

$$f_{bouy} = -\alpha\rho\boldsymbol{z} - \beta(T - T_{amb})\boldsymbol{z} \qquad \text{Equation 3.9}$$

Where $z = (0,0,1)$ points in the upward vertical direction, $T_{amb}$ is the ambient temperature of the air and $\alpha$ and $\beta$ are the thermal buoyancy constant for density and temperature respectively. Note that when $\rho = 0$ and $T = T_{amb}$ , the buoyancy force is zero.

## 3.4 Grid Based Fluid Simulation

In this section, we first overview the grid based fluid simulation and its underlying structure. Then we describe the mathematical discretization of fluid equations and a successive approach for solving these equations on a grid structure.

### 3.4.1 Overview

Grid-Based Fluid Simulation is an Eulerian approach. Instead of treating fluid as moving particles and tracking each particle movement as in Smooth Particle Hydrodynamics (SPH), Grid-Based Fluid Simulation uses a grid to represent the fluid quantities. The MAC Grid (MAC stands for Marker-and-Cell), an original grid used in fluid simulation was first introduced by [31]. It stores velocity at cell faces and stores other quantities such as pressure and density at the center of each cell.

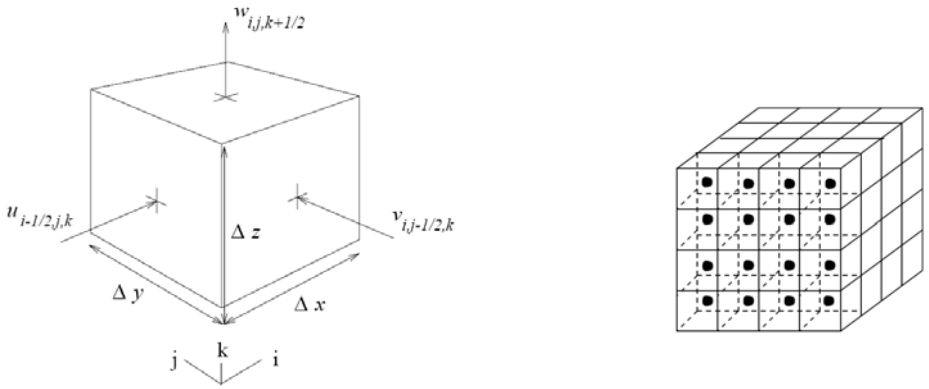



Figure 3.1 (a) Voxel of MAC grid (image from: [1])
(b) Grid-based structure (image from: [2])

[1] used relatively coarse grids to simulate fluid in 3D. [2] modified the MAC grid to store velocity at cell center for simplicity, and introduced a semi-Lagrangian advection scheme with an implicit method, which treats cells as particles during the advection step to achieve a stable simulation. Within a decade, a variety of works have been introduced using grid-based method such as fire [10], explosions [11], viscoelastic materials [12] and bubbles [13].

The grid-based approach has many advantages compare to others, e.g., reliable, stable at large time steps, high quality of smooth liquid surfaces and efficient for parallel computing. Furthermore, it is suitable to represent the fluid volume, especially for smoke that usually spreads itself out over the domain. These make grid-based methods capable for smoke simulations.

### 3.4.2 Discretization of Navier-Stokes Equations

This section shows how to discretize the continuous field of fluid described by the Navier-Stokes equations into a discrete grid space and the step-by-step of how to solve these equations on the grid structure.

The method described here is based on a stable fluids technique proposed by [2]. First of all, equation 2 is simplified by applying the Helmholtz-Hodge Decomposition:

$$\mathbb{P}\frac{\partial \boldsymbol{u}}{\partial t} = \mathbb{P}\left(-(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \frac{1}{\rho}\nabla p + v\nabla^2\mathbf{u} + \mathbf{F}\right) \qquad \text{Equation 3.10}$$

Equation 3.10 is a divergence-free equation, where $\mathbb{P}$ is the projection operator that projects any vector field onto its divergence-free component. The following step is to apply Equation 3.11 and Equation 3.12 into Equation 3.10.

$$\mathbb{P}\frac{\partial \boldsymbol{u}}{\partial t} = \frac{\partial \boldsymbol{u}}{\partial t} \qquad \text{Equation 3.11}$$

$$\mathbb{P}(\nabla p) = 0 \qquad \text{Equation 3.12}$$

The result is a projected Navier-Stokes equation as shown below.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathbb{P}(-(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + v\nabla^2\mathbf{u} + \mathbf{F}) \qquad \text{Equation 3.13}$$

The pressure term is dropped and the equation is in a simpler form. The right hand side of Equation 3.13 has only one unknown variable; velocity $\boldsymbol{u}(\boldsymbol{x})$, where $\boldsymbol{u}(\boldsymbol{x}) = \{u, v, w\}$ and $\boldsymbol{x} = \{x, y, z\}$. To solve the projected Navier-Stokes equation (Equation 3.13) for velocity, which is in a differential form, we split Equation 3.13 into four separate terms and sequentially solve term by term as illustrated below.

$$\boldsymbol{u}_0(\boldsymbol{x}) \xrightarrow{addforces} \boldsymbol{u}_1(\boldsymbol{x}) \xrightarrow{advect} \boldsymbol{u}_2(\boldsymbol{x}) \xrightarrow{diffuse} \boldsymbol{u}_3(\boldsymbol{x}) \xrightarrow{project} \boldsymbol{u}_4(\boldsymbol{x})$$

Before performing these four steps, we first initialize our domain with an empty grid. Also, velocity $\boldsymbol{u}_0$ must be given as an initial value. Then four steps are performed sequentially each iteration. The detail of each step is as follows.

***Adding forces***

External forces $\boldsymbol{F}$ including gravity force, vorticity confinement force, buoyancy force and any user-defined or control forces are added to each cell at this step. The following equation adds external forces to the cell's current velocity.

$$\boldsymbol{u}_1(\boldsymbol{x}) = \boldsymbol{u}_0(\boldsymbol{x}) + \Delta t\boldsymbol{F} \qquad \text{Equation 3.14}$$

Equation 3.14 is derived by a "*Forward Euler Method*" for speed and simplicity. It is a first-order accuracy method but efficient enough for visual simulation. Higher-order accuracy methods such as Runge-Kutta which is a second order accuracy, BFECC method [32] and the MacCormack method [33] can also be used. These higher order accuracy methods might result in a detail improvement but they are relatively slower than the Forward Euler Method.

***Advection***

To solve the advection term of Equation 3.13, instead of directly moving fluid quantities along the velocity field which would be unstable at large time steps, the semi-Lagrangian advection scheme is used for stability as introduced by [2]. The advection scheme using the semi-Lagrangian method is a first-order accurate discretization scheme both in time and space. Cells are treated as particles, each located at cell's center. Each particle is traced back in time using its current velocity to find its previous position. Then the fluid quantities (density, velocity, pressure etc.) at that position replace the fluid quantities at the current position. Here is the mathematical representation of the advection step derived from the Euler Method.

$$\boldsymbol{u}_2(\boldsymbol{x}) = \boldsymbol{u}_1(P(\boldsymbol{x}, -\Delta t)) \qquad \text{Equation 3.15}$$

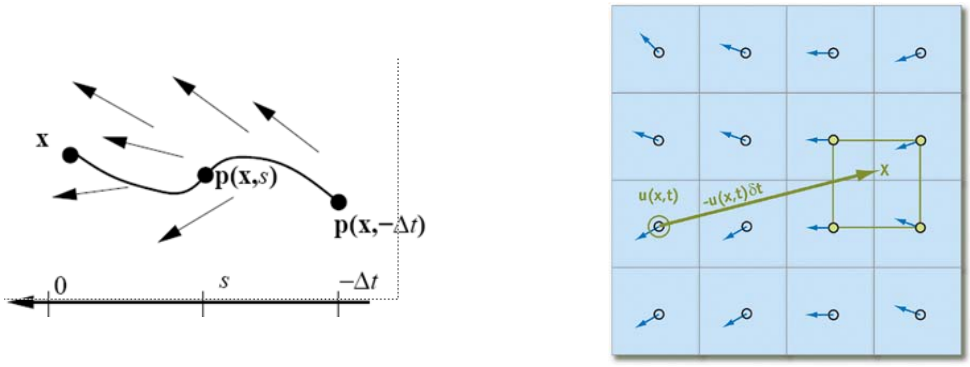



Figure 3.2 (a) Semi-Lagrangian advection (image from [2]).
(b) Fluid velocity interpolation (image from: GPUGems, ®Nvidia 2004).

Since previous position is not always located at the cell's center, interpolation is required. For speed and simplicity, bilinear interpolation is used for two dimensional grids and trilinear interpolation is used for three dimensional grids.

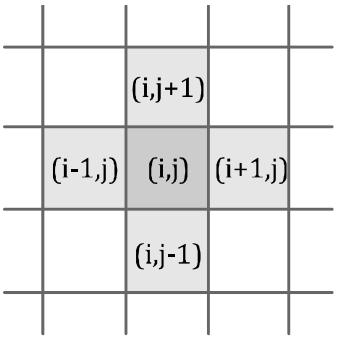
Note that there exist many higher accuracy advection schemes such as back and forth error correction [32], MacCormack [33], QUICK [34], FLIP [35] which can also be used. These higher order advection schemes produce stable yet higher detail simulations, but they also require relatively higher computational cost as well.

***Diffusion***

Diffusion describes the spreading behavior of fluids. Each cell exchanges its quantities with neighbors until the equilibrium is reached. For explicit implementation, using the Forward Euler is straightforward. Unfortunately, this is unstable when fluid propagates further than the neighboring cells. The implicit method by tracing back in time is used. However, unlike the advection step, this term cannot be solved cell-by-cell directly since there are many unknown variables contained within the Laplacian operator. Thus, the only way is to solve as a sequence of linear equations. Here is the implicit equation which can be solved by using matrix operations, where $\boldsymbol{I}$ is the identity matrix.

$$\boldsymbol{u}_2(\boldsymbol{x}) = (\boldsymbol{I} + v\Delta t\nabla^2)\boldsymbol{u}_3(\boldsymbol{x}) \qquad \text{Equation 3.16}$$

By applying a finite difference of $\nabla^2$ along with the fact that cell dimension is cubic: $\partial x = \partial y = \partial z$, Equation 3.16 can be rewritten in two-dimensional discrete form as follow.




$$\boldsymbol{u}_2(i,j) = \boldsymbol{u}_3(i,j) + v\Delta t\left\{\frac{\boldsymbol{u}_3(i+1,j) + \boldsymbol{u}_3(i-1,j) + \boldsymbol{u}_3(i,j+1) + \boldsymbol{u}_3(i,j-1) - 4\boldsymbol{u}_3(i,j)}{\partial x^2}\right\}$$

The solution to the above equation is obtained by solving a matrix of linear equations. The commonly used method is the "*Jacobi Iteration*".

***Projection***

The final step is the projection step which projects back the solution to a none divergence-free term as it was before applying the Helmholtz-Hodge Decomposition. The projection step is done by adding the pressure term back into Equation 3.13. First of all, we need to solve for pressure which is still an unknown variable. Equation 3.18 is a Poisson equation that we solve for the pressure $p$. Once the pressures are known, we then find $\nabla p$ each cell and substitute back into Equation 3.17. The result is the velocity of the current time step which is the final solution of the Navier-Stokes equations.

$$\boldsymbol{u}_4(\boldsymbol{x}) = \boldsymbol{u}_3(\boldsymbol{x}) - \Delta t\frac{1}{\rho}\nabla p(\boldsymbol{x}) \qquad \text{Equation 3.17}$$

$$\nabla^2\, p(\boldsymbol{x}) = \frac{\rho}{\Delta t}\nabla \cdot \boldsymbol{u}_3(\boldsymbol{x}) \qquad \text{Equation 3.18}$$

The Navier-Stokes equations of color density (Equation 3.3) and temperature (Equation 3.4) can be evaluated with the same approach as for velocity.

### 3.4.3 Boundary Conditions

Boundary conditions define fluid behaviors at simulation boundaries and fluid surfaces. There are two types of boundary conditions:

***Neumann boundary conditions***

At the fluid boundary, the pressure gradient in a direction of normal vector to the boundary must equal zero, or in another word, fluid pressure at the boundary must equal to the solid obstacle pressure. This condition is to ensure that no flow passes through solid walls.

$$\frac{\partial p}{\partial n}(\boldsymbol{x}) = \nabla p(\boldsymbol{x}) \cdot \boldsymbol{n} = 0 \qquad \text{Equation 3.19}$$

***Dirichlet boundary conditions***

On a fluid-solid boundary surface, velocity must satisfy *"no-slip condition"* due to the viscous effects. To enforce no-slip condition, the tangential component of the velocity of the fluid must be the same as the tangential component of the velocity of the surface. If we designate the velocity of the rigid surface as $\boldsymbol{V}(\boldsymbol{x})$ and that of the fluid as $\boldsymbol{u}(\boldsymbol{x})$, and select a unit tangent vector to the surface as $t$, the no-slip boundary condition can be stated as

$$\boldsymbol{u}(\boldsymbol{x}) \cdot t = \boldsymbol{V}(\boldsymbol{x}) \cdot t$$

Since there is no mass transfer across the boundary, the normal components of the velocity at the boundary are equal. If $\boldsymbol{n}$ represents the unit normal vector, we have:

$$\boldsymbol{u}(\boldsymbol{x}) \cdot \boldsymbol{n} = \boldsymbol{V}(\boldsymbol{x}) \cdot \boldsymbol{n}$$

As a consequence of the two above conditions, we arrive at the conclusion that the fluid velocity must match the velocity of the rigid surface at every point on it.

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{V}(\boldsymbol{x}) \qquad \text{Equation 3.20}$$

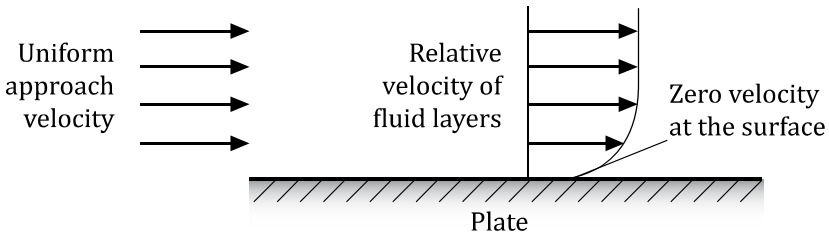
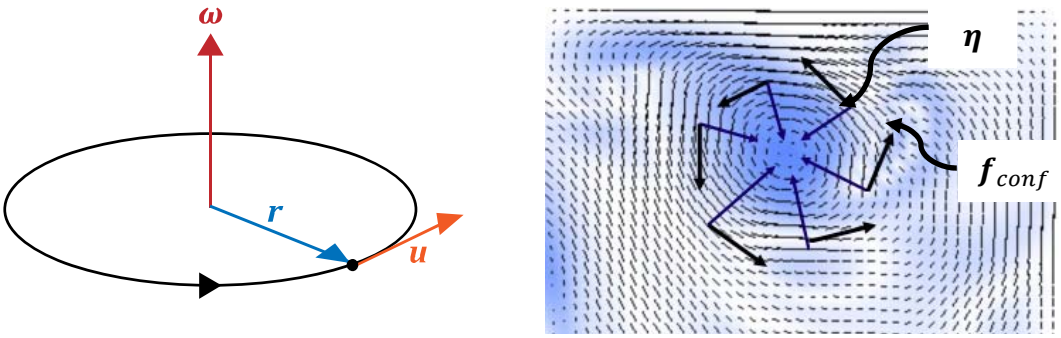



Figure 3.3 Schematic representation of no-slip boundary condition.

### 3.4.4 Vorticity Confinement




(a) (b)

Figure 3.4 (a) Vorticity direction is along the fluid rotation.
(b) Vorticity confinement force increase circulation (image from: [4])

Although using the semi-Lagrangian method for advection term is unconditionally stable even with arbitrary large time steps but when simulated with large time steps or large grid spacing, the fluid animation suffers from high numerical dissipation and results in sticky motions. [3] deal with numerical dissipation using a method called *"Vorticity Confinement"*. Vorticity is the tendency for elements of the fluid to spin or rotate. The idea is to inject some amount of vorticity during simulation to cause fluid to flow swirly. By definition, vorticity is the curl of fluid velocity:

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{u} \qquad \text{Equation 3.21}$$

In Equation 3.21, $\boldsymbol{\omega}$ denotes vorticity and $\boldsymbol{u}$ denotes velocity. Vorticity is a vector with a direction parallel to the axis of rotation. Its direction can be determined by using the right hand rule.

$$\boldsymbol{\eta} = \nabla|\boldsymbol{\omega}| \qquad \text{Equation 3.22}$$

$$\boldsymbol{N} = \frac{\boldsymbol{\eta}}{|\boldsymbol{\eta}|} \qquad \text{Equation 3.23}$$

Equation 3.22 computes the vorticity gradient while Equation 3.23 computes a normalized vorticity location vectors, where $\boldsymbol{N}$ is the normalized vorticity location vector that points from lower to higher vorticity concentrations, the direction of which is perpendicular to the axis of rotation. Then the magnitude and direction of the paddle wheel force is computed by Equation 3.24.

$$\boldsymbol{f}_{conf} = \varepsilon h(\boldsymbol{N} \times \boldsymbol{\omega}) \qquad \text{Equation 3.24}$$

Where $\boldsymbol{f}_{conf}$ is the vorticity confinement force for generating swirl effects. $\varepsilon > 0$ is used to control the amount of vorticity force to add back into the flow while $h$ is a grid scale. Note that vorticity confinement force (Equation 3.24) is treated as one of the external forces $\boldsymbol{F}$ (see Equation 3.14).

### 3.5 Adaptive Grids

Since special effects are getting important in today industry, there is even more demand for larger domains and higher detail fluid animations. Fluid simulation using a fixed uniform grid encounters a scalability problem, since increasing the grid resolution uniformly for entire domain requires a substantial amount of computational cost associated with the increased numbers of cells. Adaptive grids, on the other hand, are the alternative technique that consumes relatively lower amount of computational cost by partially reducing and increasing the grid resolution by means of a level-of-detail approach. This way, the simulation is optimized for any larger domains or higher detail animations. The octree grid and the view-dependent grid are described in the following subsections.

### 3.5.1 Octree Grid

The octree grid is basically a simulation grid constructed by using the octree structure. It has several advantages compared to fixed uniform grid structure; flexible and can be partially change its resolution for detail optimization. Grid resolution is changed by a method called "*grid refinement*". Simulation using octree grid has the following important criteria: where and when to perform grid refinement in order to properly optimize the simulation. Another criterion is how to discretize physics equations and advect fluid quantities on octree grid, which is adaptive and non-uniform structure. Grid refinement, tracing semi-Lagrangian paths and solving the Poisson equation on octree grid are described in the following subsections.

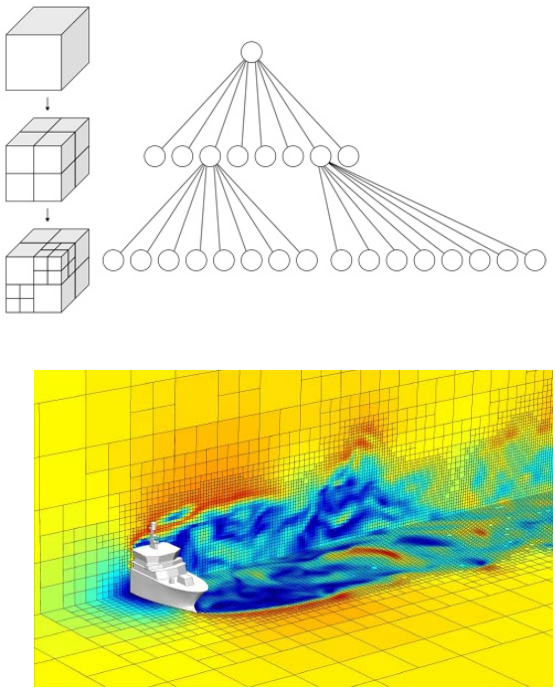


Figure 3.5 Adaptive grid using octree structure (image from: [9])

#### 3.5.1.1 Grid Refinement

Grid refinement is performed at every iteration of the simulation. It consists of two opposite tasks i.e., subdivision and merging. Subdivision is performed in order to increase the grid resolution and allows fine details to be captured while merging is performed to reduce the grid resolution and the computational cost. If subdivision and merging are performed in an appropriate way, the grid should be optimized and result in a low computational cost consumption.

[6] use variations in smoke density to decide where and when to perform subdivision and merging. The idea is that cells should be subdivided for higher resolution if their neighboring cells are relatively different to each other. On the other hand, cells should be merged for lower resolution if the variations of fluid quantities among neighboring cells are no longer significant. The criterion for subdivision can be written as follows:

$$C(x,y,z) = max\left(|\nabla_x^2\rho|, \left|\nabla_y^2\rho\right|, |\nabla_z^2\rho|\right) > T \qquad \text{Equation 3.25}$$

Where $\rho$ denotes smoke density, $\nabla_x^2\rho$ is a finite difference of $\partial^2\rho/\partial x^2$, $\nabla_y^2\rho$ and $\nabla_z^2\rho$ are defined likewise and $T$ is a specified threshold. Once there exists a cell node $(x_i, y_i, z_i)$ in the octree grid such that $C(x_i, y_i, z_i) > T$, which means if the density difference among its neighboring cells exceed the specified threshold, then that cell node should be subdivided. The velocity field, as well as the density distribution of any newly generated child node, is obtained by trilinear interpolation.

On the other hand, if a node has children and they do not have sufficient details any more, they can be merged and removed. To check whether the children nodes should be merged or not, a new density function is computed for the considered node by sub-sampling the smoke density distributions of its children nodes. If the maximum difference between the sub-sampled density and the original densities at the children nodes is smaller than a specified threshold, the children nodes should be merged and removed.

#### 3.5.1.2 Tracing Semi-Lagrangian Paths

Since the octree grid is adaptive and non-uniform over space and time, advection step cannot be performed regularly. The semi-Lagrangian scheme does not need changes as long as the tracing is kept inside an octree node. However, when its path intersects with one of the six bounding faces of the node, we need to find the appropriate neighboring node where the path can continue.

[6] describe six possible cases for semi-Lagrangian path tracing in an octree. In Figure 3.6, solid lines represent the octree partitions. Dashed lines represent the uniform grids inside octree nodes. The voxel at the head of the path is called the source, and the voxel at the tail of the path is called the destination, since fluid quantities are traced from the destination to the source, and the velocity at the source is transferred to the destination. (a) Both the source and destination voxels belong to the grid of the same node. (b) The source and destination voxels belong to two different nodes with the same resolution. (c) The resolution of the destination node is higher than that of the source node. (d) The resolution of the destination node is lower than that of the source node. (e) The destination node is a child of the source node. (f) The source node is a child of the destination node.

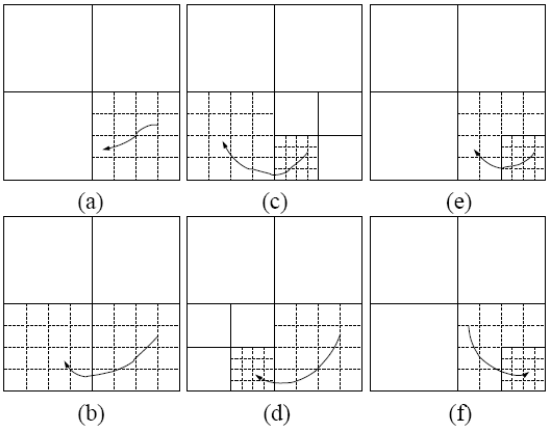



Figure 3.6 Six possible cases for semi-Lagrangian path in octree structure (image from: [6])

### 3.5.1.3 Solving Possion Equation

Soving Equation 3.18 on the octree grid, which is adaptive and non-uniform, requires a particular discretization of the Possion equation. Since the discretization is closely related to the second vector form of Green's theorem that relates a volume integral to a surface integral, [5] developed an adaptive discretization on an octree data structure using the Green's theorem stated as follows.

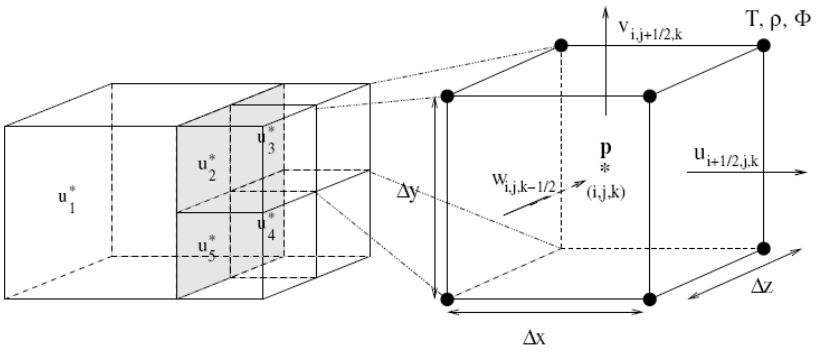



Figure 3.7 Left: one large cell neighboring four smaller cells. Right: zoom of computational cell. (image from: [5])

Consider the discretization of Equation 3.18 for a large cell with dimensions: $\Delta x$ , $\Delta y$ and $\Delta z$ neighboring small cells as depicted in Figure 3.7. First rescale Equation 3.18 by the volume of the large cell to obtain $V_{cell}\Delta t\nabla^2\, p(\boldsymbol{x}) = V_{cell}\rho\nabla\cdot\boldsymbol{u}_3(\boldsymbol{x})$. The right hand side of the equation now represents the quantity of mass flowing in and out of the large cell within a time step $\Delta t$ in $m^3 s^{-1}$. This can be further rewritten as

$$V_{cell}\nabla\cdot\left(\rho\boldsymbol{u}_3(\boldsymbol{x}) - \Delta t\nabla p(\boldsymbol{x})\right) = 0 \qquad \text{Equation 3.26}$$

This equation implies that the $\nabla p$ term is most naturally evaluated at the same location as $\boldsymbol{u}_3(\boldsymbol{x})$, namely at the cell faces, and that there is a direct correspondence between the components of $\nabla p$ and $\boldsymbol{u}_3(\boldsymbol{x})$. Moreover, substituting Equation 3.17 into Equation 3.26 implies $V_{cell}\nabla\cdot\boldsymbol{u}_4 = 0$ or $\nabla\cdot\boldsymbol{u}_4 = 0$ as desired.

Invoking the second vector form of Green's theorem, one can write

$$V_{cell}\,\nabla\cdot\boldsymbol{u}^{*}=\sum_{faces}\left(\boldsymbol{u}_{face}^{*}\cdot\boldsymbol{n}\right)A_{face}$$

where $\boldsymbol{n}$ is the *outward* unit normal of the large cell and $A_{face}$ represents the area of a cell face. In the case of Figure 3.7, the discretization of the $x$ component $\partial u^{*}/\partial x$ of the divergence reads

$$\frac{\partial u^{*}}{\partial x}=\frac{u_2^{*}A_2+u_3^{*}A_3+u_4^{*}A_4+u_5^{*}A_5-u_1^{*}A_1}{\Delta x\Delta y\Delta z}$$

where the minus sign in front of $u_1^{*}A_1$ accounts for the fact that the unit normal points to the left. Then

$$\frac{\partial u^{*}}{\partial x}=\frac{(u_2^{*}+u_3^{*}+u_4^{*}+u_5^{*})/4-u_1^{*}}{\Delta x}$$

The $y$ and $z$ directions are treated similarly. Once the velocity divergence is computed, we solve for the pressure gradient by constructing a linear system of Equation 3.18. Invoking again the second vector form of Green's theorem:

$$V_{cell}\nabla\cdot(\Delta t\nabla p)=\sum_{faces}\left((\Delta t\nabla p)_{face}\cdot\boldsymbol{n}\right)A_{face}\qquad\text{Equation 3.27}$$

The remaining discretization of the pressure gradient $(\nabla p)_{face}$ is carried out in such a manner that the resulting matrix is symmetric. It has been shown that the system still yields a consistent approximation when the gradients are calculated with standard central differences applied to the direct neighbor cells, as long as the perturbation in the pressure location is $\mathcal{O}(\Delta x)$ [7].

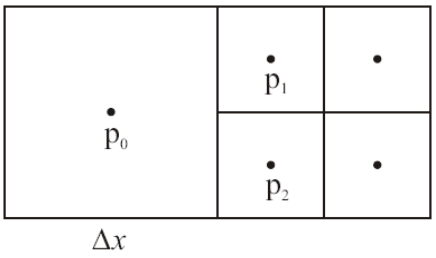



Figure 3.8 Pressure discretization on octree (image from: [7])

Figure 3.8 shows a 2D example of the octree grid where the pressure gradient of the large cell can be obtained by:

$$\sum_{faces} (\nabla p \cdot \boldsymbol{n})\, A_{face} = \left(\frac{p_1 - p_0}{\Delta x} + \frac{p_2 - p_0}{\Delta x}\right)\frac{1}{2}\Delta x = \bar{p} - p_0 \qquad \text{Equation 3.28}$$

In this notation, $\bar{p}$ is the arithmetic average of $p_1$ and $p_2$. The discretization yields a large and sparse linear system with an equation for every cell of the grid.

Once the pressure gradient is found at every face, we obtain the pressure value from Equation 3.17 by carrying out the computation in a manner similar to that of the velocity divergence above.

### 3.5.2 View-Dependent Grid

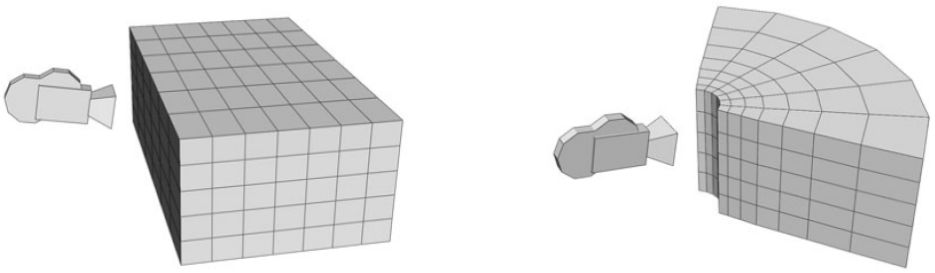


Figure 3.9 Left: Traditional unifrom grid. Right: View-dependent grid (image from: [8])

Departing from the adaptive grid using octree structure, the view-dependent grid explores the use of a non-Cartesian, stable fluid method solver to obtain a view-dependent level-of-detail. A Cartesian coordinate system is not always an optimal fit to some dynamics problems, such as flow around an airfoil or through a pipe. In these cases it is useful to define a new coordinate system that fits the problem at hand, mapping it to a regular grid for the purpose of computation.

For a view dependent simulation proposed by [8], a cylindrical coordinate system with the camera positioned on the central axis is used. In this coordinate system, a grid is built to resemble the camera viewing frustum. The

computational grid provides fine detail close to the viewing position and geometrically reduces detail with distance from the viewer.

To solve the equations for fluid flow in transformed coordinated system other than Cartesian, coordinate transformation must be applied. Transformed coordinates and Grid generation are described in following subsections.

#### 3.5.2.1 Transformed Coordinates

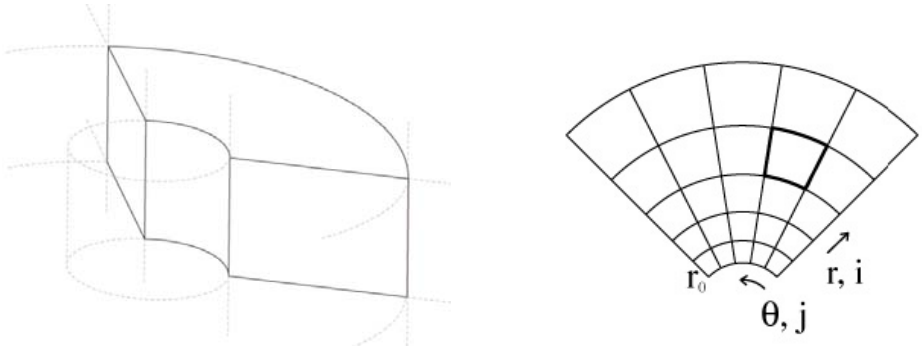



Figure 3.10 Cylindrical coordinate system (image from: [8])

Fluid equations are solved in the transformed coordinate then mapped back to the Cartesian space when needed. There are two approaches to the coordinate transformation: a direct transformation and an inverse transformation. Given a regular, orthogonal grid defined in a Cartesian space, the direct approach requires a set of one-to-one functions that map Cartesian coordinates to a cylindrical coordinate system:

$$r = r(x, y, z) = \sqrt{x^2 + y^2} \qquad \text{Equation 3.29}$$

$$\theta = \theta(x, y, z) = \tan^{-1}\left(\frac{y}{x}\right) \qquad \text{Equation 3.30}$$

$$h = h(x, y, z) = z \qquad \text{Equation 3.31}$$

And the inverse map back to the Cartesian coordinates are:

$$x = x(r, \theta, h) = rcos\theta \qquad \text{Equation 3.32}$$

$$y = y(r, \theta, h) = rsin\theta \qquad \text{Equation 3.33}$$

$$z = z(r, \theta, h) = h$$ Equation 3.34

Moreover, if we have a field $u(r, \theta)$ defined in the computational space, applying the chain rule yields:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial x}$$ Equation 3.35

$$\frac{\partial u}{\partial y} = \frac{\partial u}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial y}$$ Equation 3.36

which provides Cartesian partial derivatives for $u$ in terms of the transformation. Second order derivatives may be found similarly.

#### 3.5.2.2 Grid Generation

For view dependent fluid simulation in the computational space using cylindrical coordinate system $(r, \theta, h)$ , the grid is constructed by placing the viewer at the origin of the cylindrical system. The view depth corresponds to a radial coordinate $r$, viewing direction corresponds to an angle $\theta$, and elevation to $h$.

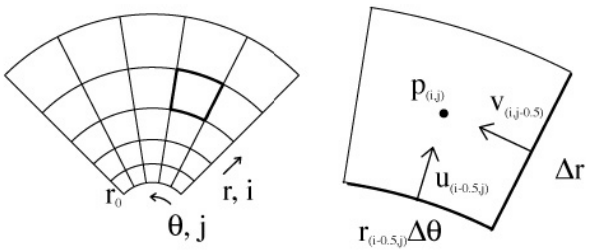
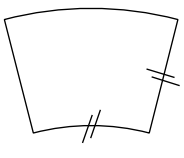



Figure 3.11 View-dependent polar computational grid (image from: [8])

Each cell's dimension has to be cubic-like in order to solve the system without numerical complexity. The proportions of each cell are kept consistent by maintaining a constant ratio of the depth of each cell to the arc length produced by sweeping its angular increment. The height of a cell can simply be kept constant. This produces a convenient, geometrically increasing function to define a view dependent grid.



Assume that the center of the first grid cell is at a radial distance $r_0$ from the origin, and that each cell's angular dimension is $\Delta\theta$. To enforce all cell to have uniform spatial dimensions (i.e. equivalent of a square or cubic), then the radial depth of the cell should be $r_0\Delta\theta$. Here are the constrain equations numbering from n = 0.

$$r(0) = r_0 \qquad \text{Equation 3.37}$$

$$r(n+1) = (1+\Delta\theta)r(n) \qquad \text{Equation 3.38}$$

Equation 35 and equation 36 can be written in a closed form as follow.

$$r(n) = r_0(1+\Delta\theta)^n \qquad \text{Equation 3.39}$$

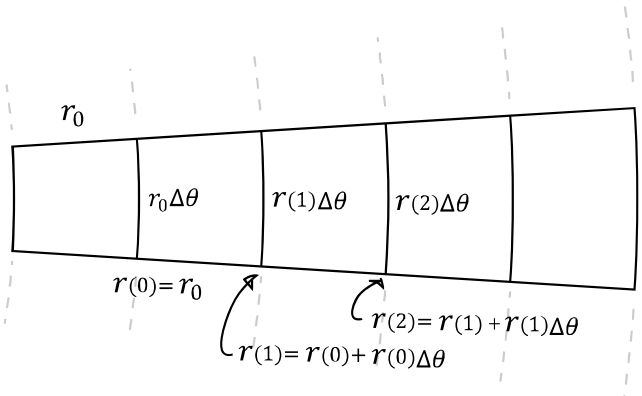



Figure 3.12 Proportion of polar computational grid

# CHAPTER 4

## Proposed Method

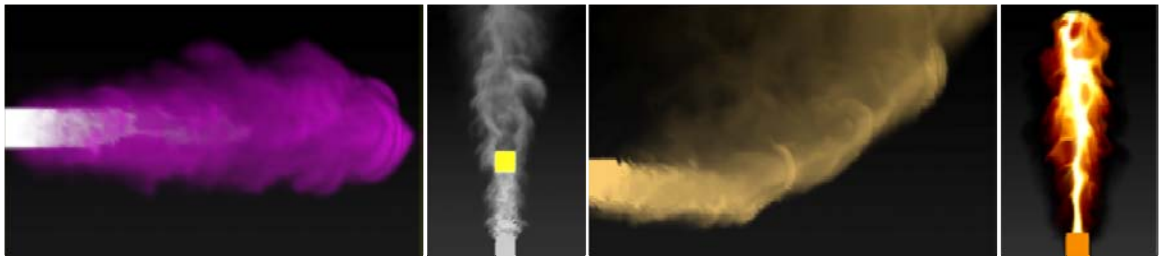


Figure 4.1 Screen Shots of smoke simulation using our method

In this section, we propose a view-dependent adaptive grid refinement which is an improved method to efficiently optimize the simulation grid, since smoke simulation using the octree grid such as [5,6] does not address the viewing angle. Their grid refinements are associated only with variation among the neighboring cells but viewing angle is not considered. To be precise, distant smoke that is far beyond the visible scene, which is unnecessary for fine detail simulation, is poorly optimized in these previous works. Therefore, with a general idea from the view-dependent grid proposed by [8], we propose a method to refine the octree grid associated with the viewing information along with the variation of neighboring cells. Overall, the method optimizes the simulation grid for lower computational cost as well as preserves the visible details.

In addition, particles, which are flexible to conform to obstacle-fluid boundaries, are integrated into our model to enhance the animations and reduce the artifacts caused by dynamic refinements.

We first overview the overall process of the smoke simulation with a brief introduction to each steps to give a general concept of the simulation framework, and to point out where our method is placed in. Then in the following section, we describe the structure of the octree grid. Follow by the details to each step, including our proposed view-dependent adaptive grid refinement method.

## 4.1 Overview

The overall process of smoke simulation can be divided into four major steps i.e., initializing the grid, refining the grid, updating cells and rendering. The simulation starts with the initializing grid step. Other steps are then sequentially executed repeatedly until the simulation reaches its end or is halted. The simulation process flow including our refinement method is summarized in Figure 4.2, where its corresponding steps are detailed in the following subsections.

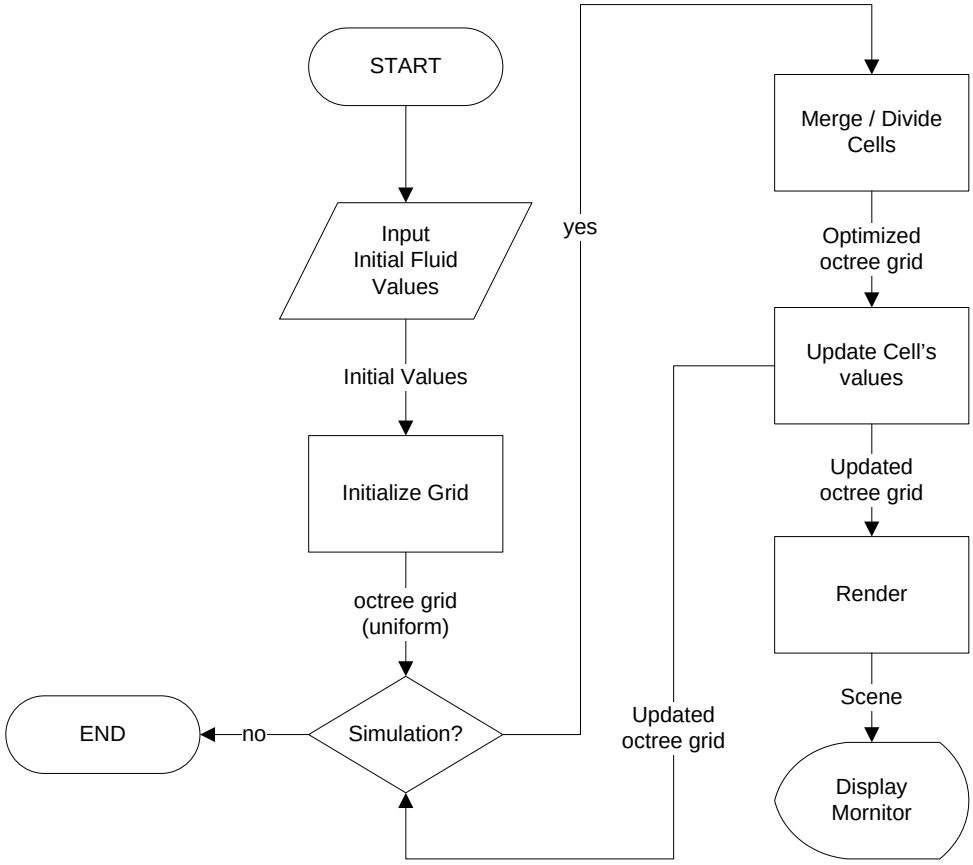



Figure 4.2 A process diagram of smoke simulator with adaptive grid refinement

***Initializing the Grid***

Before any simulation can be performed, the grid, which is an infrastructure of the simulation, must be first constructed and initialized with some initial values. Generally, the grid is initialized using user-given initial fluid values (i.e., velocity,

density, pressure, temperature). The result of this process is the octree-structured grid. But since the grid refinement has not been performed, the grid is structured as a uniform subdivision; same cells' size entire domain.

After the grid is successfully constructed and every time before proceed to the next following steps, a decision must be made; determine whether the simulation will be paused or continued, if so, halt the simulation or otherwise continue on the refinement step.

***Refining the Grid***

The octree grid without refinement, when passed through the fluid solver, spends more computational cost due to its un-optimized structure. Grid refinement is an optional step that optimizes the grid before performing any heavy computation.

The input of this step is an un-optimized octree grid. In this step, cells are partially merged and subdivided. Our view-dependent adaptive grid refinement is applied in order to optimize the octree grid for both viewing and details. Further details for our method is described in section 4.3. The output is an optimized octree grid that is ready for the value updating step.

***Updating Cells***

After applying the refinement method, the grid is well optimized and ready for the simulation step which is the part that takes the most computational time. We have constructed a solver to solve for smoke equations, the details are described in Section 4.4. Once we obtain all the updated values, the grid is then passed to the next step for rendering the results.

***Rendering***

At this step, the computed data is rendered as an animation on screen. The fluid values of every cell, along with their positions, are the input for the rendering process. Several rendering techniques can be applied e.g., billboarding, volumetric rendering or soft particle rendering. Light and shadow can also be applied for a realistic visual result.

In our work, we have presented a coupling between billboarding and particle system which combine their benefits together to enhance animation quality and reduce the artifacts caused by dynamic grid refinements. Details are described in Section 4.5.

### 4.2 Structure of Octree Grid

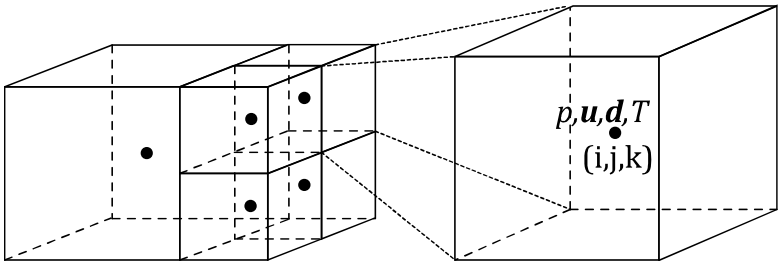



Figure 4.3 Logical structure of octree grid

We have constructed an adaptive grid using the octree structure as illustrated in Figure 4.3. A cell is a fundamental unit of the octree grid. Its size and position are adaptable but constrained by the octree structure. Generally, cell dimensions are a power of 2 (denoted as $2^3$) whereas higher power such as $4^3$ may cause a rapid changing in grid size during the grid refinement, resulting in animation artifacts and discontinuity in fluid values.

Each cell is defined by its position vector $\boldsymbol{x} = (x, y, z)$, velocity vector $\boldsymbol{u} = (u, v, w)$, four-dimensional density vector $\boldsymbol{d} = (d_a, d_r, d_g d_b)$ and a scalar pressure $p$. Note that $d_a, d_r, d_g, d_b$ denote color value of transparency, red, green and blue respectively. All fluid quantities including velocity are stored at cell's node, similar to [2], since storing the velocity at the cell's faces, e.g., traditional MAC grid [31], requires more memory. Moreover, storing all fluid quantities at cell's node is straightforward to implement.

### 4.3 View-Dependent Adaptive Grid Refinement

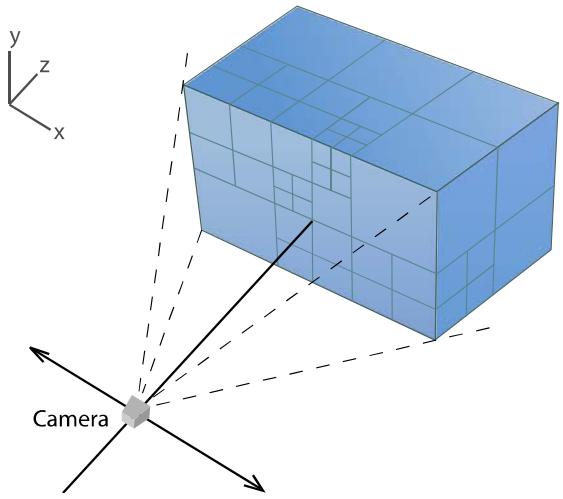



Figure 4.4 Illustration of octree grid using view-dependent adaptive grid refinement

Grid refinement is performed throughout the domain to optimize the grid, i.e., subdivide for acquiring detail in particular regions or merge for saving computational time for faster simulation. Refinement conditions are used to decide for each cell whether to perform subdivision or merging. In general, refinement conditions are defined by comparing only the fluid variation with arbitrary constant thresholds such as the one proposed in [6]. However, this refinement scheme is not sufficient for view-dependent optimization. We have modified these conditions with additional parameters for addressing the viewing angle; thus, not only is fluid detail optimized, but also the viewing angle as well.

The idea to optimize the grid for both viewing angle and details is that cells should be divided for higher resolution if they are too close to the camera or their neighboring cells are relatively different to each other. On the other hand, cells should be merged for lower resolution if their distances to the camera are too far for any fine details to be visualized or the variation of fluid quantities between the neighboring cells are no longer significant. However, not only do the fluid variation and the distance from the camera affect the grid refinements, the dimensions of the viewing frustum and their perspectives are also important parameters that should be properly weighted with the refinement conditions in order to achieve an effective detail-preserved optimization.

We have proposed an improved refinement method by considering both the variation in the fluid quantities and the viewing information to perform additional refinement based on the view-dependent level-of-detail. Briefly, we have presented a *"view-dependent weighting factor"* as a factor to weight the refinement thresholds. Thus, in this manner, thresholds are adaptive and directly proportional to the viewing frustum. Thresholds with view-dependent weighting are called *"adaptive thresholds"* and the refinement with conditions using these thresholds is called *"view-dependent adaptive grid refinement"*.

Measuring the variation in the fluid quantities is described in subsection 4.3.1 while a view-dependent weighting factor is described in subsection 4.3.2., and the adaptive thresholds for merging and subdivision are described in subsection 4.3.3.

### 4.3.1 Measuring Fluid Variation

Fluid variation measures the difference of fluid values among adjacent cells, which roughly indicates the amount of detail to be preserved. High fluid variation implies sharp edges of fluid boundaries where details are usually needed, whereas low fluid variation implies steady flows or empty spaces where details can be neglected.

In general, the fluid variation is indicated by measuring only the variation of density using an equation defined by [6]. However, we found that measuring the variation in velocity is also a good representative inferring the variation of other fluid quantities as well. Since density is always advected (carried) by velocity fields; thus, measuring the velocity itself yields relevant results. Moreover, by measuring the variation of velocity, Laplacian terms shown in Equation 4.1 do not need to be computed, as they can be obtained directly from the diffusion step during solving the Navier-Stokes equations (see Table 4.2). Our equation for measuring the fluid variation by velocity is as follows.

$$C(x,y,z) = max(|\nabla_x^2 u|, |\nabla_x^2 v|, |\nabla_x^2 w|) \qquad \text{Equation 4.1}$$

Equation 4.1 measures the variation of velocity, where $u, v, w$ denote scalar components of velocity vector: $\boldsymbol{u}(\boldsymbol{x}) = \{u, v, w\}$. $\nabla_x^2 u$ is a Laplacian on the $x$ direction: $\nabla_x^2 u = \partial^2 u / \partial x^2$, and $\nabla_x^2 v$ and $\nabla_x^2 w$ are defined likewise.

### 4.3.2 View-Dependent Weighting Factor

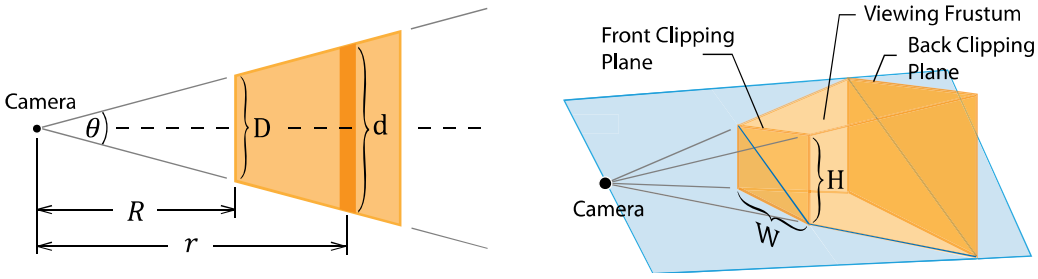



Figure 4.5 Viewing Frustum shown in 3D (right) and its diagonal cross-section (left), shaded areas are visible volume.

In this subsection, we determine any possible factor that should affect the view-dependent grid refinement. These factors are grouped together and rewritten as a mathematical relation called a *"view-dependent weighting factor"*.

Suppose that we are observing an arbitrary object within a rectilinear perspective view. If we move the camera away from the object at constant speed, then the observed size of that object should be decreased hyperbolically. In addition, if we move the camera away and increase the object's actual size accordingly, then with some proportion, the observed size should be preserved as constant. This same proportion is inherited for constructing a view-dependent weighting factor as the grids resolutions over distance should be coarsened away in a manner that the observed grid resolution still remains constant.

Let $\Psi$ represent a view-dependent weighting factor, and $r$ and $R$ be the Euclidean distance measured from camera to an arbitrary cell's node and from the camera to the front clipping plane respectively (see Figure 4.5). $\Psi$ should then be proportional to $r$. Moreover, $\Psi$ should be affected by dimensions of the viewing frustum and its perspective. For instance, $\Psi$ should be greater if $r$ is large with respect to $R$ or the viewing frustum has high perspective. The view-dependent weighting factor is defined as:

$$\Psi = \alpha\left(\frac{r}{R}\right) \qquad \text{Equation 4.2}$$

Where $\alpha$ is the camera's perspective ratio defined as follows:

$$\alpha = \frac{D}{2R} = tan\left(\frac{\theta}{2}\right) \qquad \text{Equation 4.3}$$

Equation 4.3 refers to the viewing frustum (Figure 4.5). $D$ is the diagonal length of the front clipping plane: $D = \sqrt{H^2 + W^2}$, where $H$ and $W$ are height and weight of front clipping plane respectively. $R$ is the Euclidean distance from camera to the center of front clipping plane, $\theta$ is the camera's field of view (abbreviated as FOV, also called angle of view) measured diagonally.

### 4.3.3 Adaptive Thresholds

Grid refinement is performed by means of refinement conditions i.e., merging condition and subdivision condition. In general, refinement conditions comparing the fluid variation with constant thresholds are used. However, to further perform a view-dependent adaptive grid refinement, we propose refinement conditions incorporated with *"adaptive thresholds"* as stated below.

Let $T$ be a constant threshold for grid refinement specified on a viewing frustum's front clipping plane with FOV $90^{\circ}$. Then an adaptive threshold for an arbitrary viewing frustum at distance $r$ and FOV $\theta$ is defined as follows.

$$T^* = (\tau \Psi \phi) T \qquad \text{Equation 4.4}$$

Where $T^*$ is the adaptive threshold, $\Psi$ is the view-dependent weighting factor and $\tau$ is a view-dependent coefficient specifying the weight that the viewing angle should affect the grid refinement.

Image resolution (or output resolution), which is simply a multiplication of height and weight of screen, is another factor that should be considered. The grid resolution does not need to be greater than image resolution, for example, if we lower the image resolution but keep everything else fixed, the grid can be allowed to be coarser (since details are negligible in a low-resolution image). To address this, we define a resolution ratio (Ø) as a square root of grid resolution over image resolution:

$$\phi = \sqrt{\frac{res_{grid}}{res_{image}}} \qquad \text{Equation 4.5}$$

For example, a resolution ratio of four means the grid resolution is twice finer than the image resolution which is unnecessary, thus the computed thresholds are factored to coarsen the grid for faster simulation.

### 4.3.4 Refinement Conditions

Once we obtain both fluid variation and adaptive thresholds for each cell, we compare them cell by cell to decide whether to merge or subdivide the grid using the following refinement conditions.

$$C(x, y, z) > T_s^* \qquad \text{Equation 4.6}$$

$$C(x, y, z) < T_m^* \qquad \text{Equation 4.7}$$

Equation 4.6 and Equation 4.7 are the refinement conditions for subdivision and merging respectively. $C(x, y, z)$ is a fluid variation (see subsection 4.3.1). $T_s^*$ is an adaptive threshold for subdivision and $T_m^*$ is an adaptive threshold for merging. These adaptive thresholds are inherited from Equation 4.4, written as follows.

$$T_s^* = (\tau \Psi \phi) T_s \qquad \text{Equation 4.8}$$

$$T_m^* = (\tau \Psi \phi) T_m \qquad \text{Equation 4.9}$$

Where $T_s$ and $T_m$ are respectively the constant threshold for grid subdivision and merging defined at front clipping plane of a viewing frustum at distance $R$ and FOV $90^{\circ}$.

Refinement conditions categorize each cell in to one of three stages i.e., subdivision, merging and idle. If Equation 4.6 is satisfied then the subdivision stage is assigned. Likewise, if Equation 4.7 is satisfied then the merging stage is assigned. Otherwise, if neither Equation 4.6 nor Equation 4.7 is satisfied, then an idle stage is assigned; therefore, no refinement is performed.

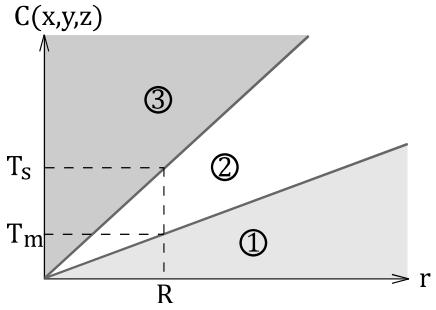



Figure 4.6 Relation between stages on fluid variation versus cell-to-camera distance

Figure 4.6 is a graphical plot of the fluid variation $C(x, y, z)$ versus cell-to-camera distance $r$. The plot shows the relation between these three refinement stages. Shaded area labeled as ① is the subdivision stage,② is the idle stage and ③ is the merging stage. Cells tend to change their current stage from merging to idle and from idle to subdivision stage if their variation $C(x, y, z)$ increases or the distance $r$ is increased or both are increased.

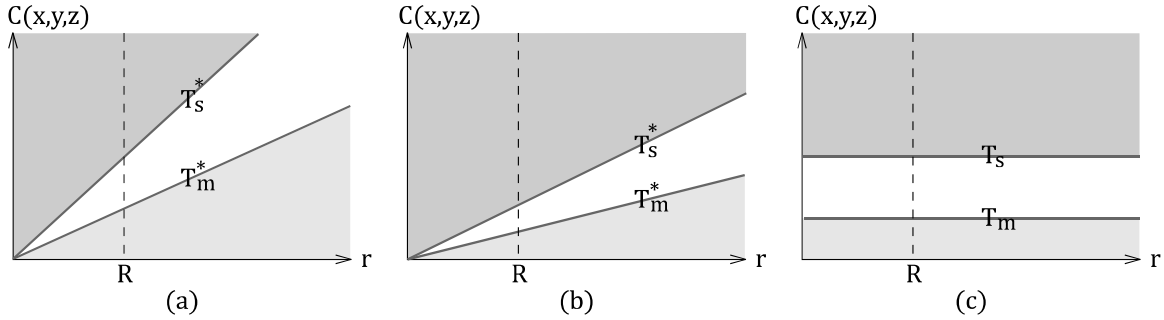



Figure 4.7 Comparison of using different camera's FOV

Figure 4.7 shows the comparison of several parameter adjustments. Steepness of slope indicates tendency of stage transition, or in another word, grid may coarsened faster and merged harder for steep slope (a) and vice versa for flat slope (b). There are many parameters that effect the steepness of slope. For instance, steepness of slope may be increased by applying a greater view-dependent coefficient (increase $\tau$), using a wider FOV camera (increase $\alpha$) or rendering with a lower output resolution (increase $\emptyset$).

Figure 4.7(c) is a case that uses a refinement model proposed by [6]. The refinement is depend only on fluid variation but not on cell-to-camera distance. Grid

refinement has no view-dependent optimization since refinement thresholds $T_s$ and $T_m$ are constant throughout the domain.

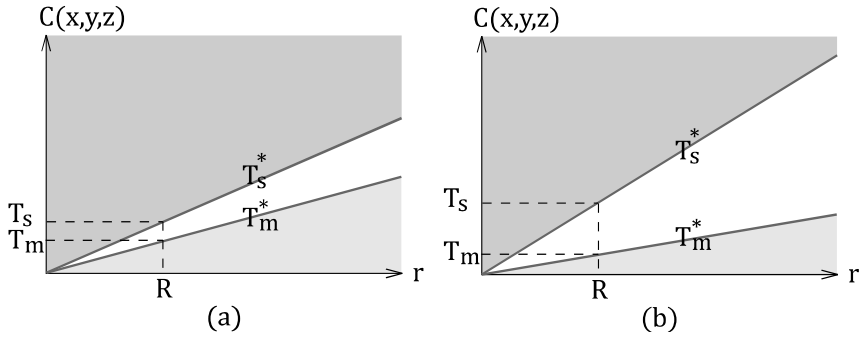



Figure 4.8 Comparison of applying constant thresholds in different values

Sensitivity of stage transitions between subdivision, merging and idle correspond to the differential value of the adaptive threshold $T_s^*$ and $T_m^*$ which can be adjusted by altering the constant threshold $T_s$ and $T_m$. In Figure 4.8 (a), $T_s$ is close to $T_m$ which results in a narrow idle stage area; hence, it is sensitive to stage transition, whereas Figure 4.8 (b), $T_s$ is much different to $T_m$ therefore, it is tolerant to stage transition. As stage transitions consume computational cost, assigning inappropriate thresholds may result in an excessive overhead due to frequent merging and subdivision, or otherwise result in an inefficient optimization; thus, thresholds must be carefully selected. Moreover, to prevent excessive refinement overhead, grid size should be gradually adapted at each time step, which can be done by performing only one operation (i.e., subdivision, merging, or idle) per cell each iteration. This reduces recursive subdivision and merging overhead and also prevents rapid changing in cell size that might cause animation artifacts.

### 4.4 Solving Smoke Equations

Recall from Chapter 3, the Navier-Stokes equations summarized in Table 4.1 are solved cell-by-cell to update their fluid values. We have constructed a solver to solve these smoke equations. Since the input grid is already optimized, the computation cost here should be lower, with respect to the non-optimized grid. Solving the Navier-Stokes equations consists of four sub-stages related to the four terms of the equation, i.e., added force, advection, diffusion, projection (see section 3.4.2 for details).

Each sub-stage as summarized in Table 4.2 is performed with a semi-Lagrangian scheme [2] for unconditional stability. Moreover, a vorticity confinement [3] is applied within the added force stage for nice swirling effects. We use a method proposed by [5] for discretizing the Navier-Stokes equations on the octree grid.

| | | |
|---|---|---|
| Velocity | $\nabla \cdot \boldsymbol{u} = 0$ | Equation 3.1 |
| | $\frac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{u} + \boldsymbol{F}$ | Equation 3.2 |
| Density | $\frac{\partial \boldsymbol{d}}{\partial t} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{d} + \boldsymbol{S}$ | Equation 3.7 |
| Temperature | $\frac{\partial T}{\partial t} = -(\boldsymbol{u} \cdot \nabla)T + H$ | Equation 3.8 |

Table 4.1 Smoke Equations

| | | | |
|---|---|---|---|
| Velocity | $\boldsymbol{u}_0(\boldsymbol{x}) \xrightarrow{addforces} \boldsymbol{u}_1(\boldsymbol{x}) \xrightarrow{advect} \boldsymbol{u}_2(\boldsymbol{x}) \xrightarrow{diffuse} \boldsymbol{u}_3(\boldsymbol{x}) \xrightarrow{project} \boldsymbol{u}_4(\boldsymbol{x})$ | | |
| | Added Forces | $\boldsymbol{u}_1(\boldsymbol{x}) = \boldsymbol{u}_0(\boldsymbol{x}) + \Delta t \boldsymbol{F}$ | Equation 3.14 |
| | Advection | $\boldsymbol{u}_2(\boldsymbol{x}) = \boldsymbol{u}_1(P(\boldsymbol{x}, -\Delta t))$ | Equation 3.15 |
| | Diffusion | $\boldsymbol{u}_2(\boldsymbol{x}) = (\boldsymbol{I} + \nu \Delta t \nabla^2)\boldsymbol{u}_3(\boldsymbol{x})$ | Equation 3.16 |
| | Projection | $\boldsymbol{u}_4(\boldsymbol{x}) = \boldsymbol{u}_3(\boldsymbol{x}) - \Delta t \frac{1}{\rho}\nabla p(\boldsymbol{x})$ | Equation 3.17 |
| | | $\nabla^2 p(\boldsymbol{x}) = \frac{\rho}{\Delta t}\nabla \cdot \boldsymbol{u}_3(\boldsymbol{x})$ | Equation 3.18 |
| Density | $\boldsymbol{d}_0(\boldsymbol{x}) \xrightarrow{addforces} \boldsymbol{d}_1(\boldsymbol{x}) \xrightarrow{advect} \boldsymbol{u}_2(\boldsymbol{x})$ | | |
| | Added Forces | $\boldsymbol{d}_1(\boldsymbol{x}) = \boldsymbol{d}_0(\boldsymbol{x}) + \Delta t \boldsymbol{S}$ | Equation 4.10 |
| | Advection | $\boldsymbol{d}_2(\boldsymbol{x}) = \boldsymbol{d}_1(P(\boldsymbol{x}, -\Delta t))$ | Equation 4.11 |
| Temperature | $T_0(\boldsymbol{x}) \xrightarrow{addforces} T_1(\boldsymbol{x}) \xrightarrow{advect} T_2(\boldsymbol{x})$ | | |
| | Added Forces | $T_1(\boldsymbol{x}) = T_0(\boldsymbol{x}) + \Delta t H$ | Equation 4.12 |
| | Advection | $T_2(\boldsymbol{x}) = T_1(P(\boldsymbol{x}, -\Delta t))$ | Equation 4.13 |

Table 4.2 Discretization of smoke equations

Note that diffusion terms are neglected in Equation 3.7 and Equation 3.8 since the appearance of viscosity is dominated by the numerical dissipation [3], except for Equation 3.6 since the velocity diffusion term can be reused for measuring the fluid variation (Equation 4.1) in the refinement process. Moreover, velocity dominates the effect of other fluid quantities; either color density or temperature is advected by velocity. Thus, it is worth computing the diffusion term of velocity.

Referring to Table 4.2, we solve for velocity first, followed by solving for density then temperature respectively. Solving for the added forces step and the advection step is straightforward. Nevertheless, the most complicated of all part is the diffusion and projection steps. These steps contain Possion equations i.e., equations with Laplacian terms ($\nabla^2$) and gradient terms ($\nabla$) which need a specific discretization specifically on the octree grid which is an adaptive and non-uniform structure. We follow a discretization scheme proposed by [5]. $\nabla \cdot \boldsymbol{u}$ of Equation 3.18 is discretized by using Green's theorem as follows:

$$\nabla \cdot \boldsymbol{u} = \frac{\sum_{faces}(\boldsymbol{u}_{face} \cdot \boldsymbol{n})A_{face}}{V_{cell}} \qquad \text{Equation 4.14}$$

where $\boldsymbol{n}$ is the outward unit normal, $A_{face}$ is the area of cell face, and $V_{cell}$ is the volume of a cell.

In a similar way, the theorem is applied to $\nabla^2\boldsymbol{u}$ of Equation 3.16 and $\nabla^2 p$ of Equation 3.18, where $\nabla^2\boldsymbol{u} = (\nabla^2 u, \nabla^2 v, \nabla^2 w) = \nabla \cdot (\nabla \boldsymbol{u})$ and $\nabla^2 p = \nabla \cdot (\nabla p)$. Their discretizations can be written as follows.

$$\nabla \cdot (\nabla \boldsymbol{u}) = \frac{\sum_{faces}((\nabla \boldsymbol{u})_{face} \cdot \boldsymbol{n})A_{face}}{V_{cell}} \qquad \text{Equation 4.15}$$

$$\nabla \cdot (\nabla p) = \frac{\sum_{faces}((\nabla p)_{face} \cdot \boldsymbol{n})A_{face}}{V_{cell}} \qquad \text{Equation 4.16}$$

Although we have successfully discretized these Poisson equations, we cannot directly solve them cell by cell individually because they are associated with multiple variables. Thus, the only way is to solve these equations at once by

constructing a set of linear differential equations. We use a "*Jacobi iteration*" method since it is a common, fast, and effective one.

Once $\nabla p$ is computed at every faces and substituted back into Equation 3.17, we obtain a velocity $\boldsymbol{u}_4$ which is the solution of the Navier-Stokes equations of velocity (Equation 3.1 and Equation 3.2).

### 4.5 Coupling with Particle System

There are a number of ways to represent the computed result. Since billboarding and particle system are fast and are typically used for modelling fluids such as fire, explosions, and smoke, we couple these two methods to combine their advantages together. The computed density values, composed of transparency, red, green and blue color data are directly used for constructing billboards, which is a fast and an effective way to represent the existence of fluid density within the domain. Meanwhile, we use the computed velocity vector fields to model the motion of particles, since particles can move freely from cell to cell throughout the domain, which yields a better representation of continuous motions. In addition, they conform well to curved and complex boundaries, which can be used to reduce artifacts caused by coarse grids.

Particles are released and advected freely throughout the simulation domain by using the computed velocity field from the view-dependent octree grid to model their motions. There are many available methods to model the particle motion such as the Euler method, Runge-Kutta methods, BFECC method [32] and the MacCormack method [33]. In this work, we prefer to use the Euler method since it is fast and simple and provides a sufficient accuracy for most situations. The method is described as follows.

$$P_i(\boldsymbol{x}_i, t + \Delta t) = P_i(\boldsymbol{x}_i, t) + \boldsymbol{u} \qquad \text{Equation 4.17}$$

Each particle $P_i$ is defined by its position $\boldsymbol{x}_i$. During the simulation, the particle is moved forward by a velocity $\boldsymbol{u}$, where $\boldsymbol{u}$ is obtained by a linear interpolation of velocity stored in the nearest neighboring cells. Figure 4.9 is a schematic view of our hybrid system.

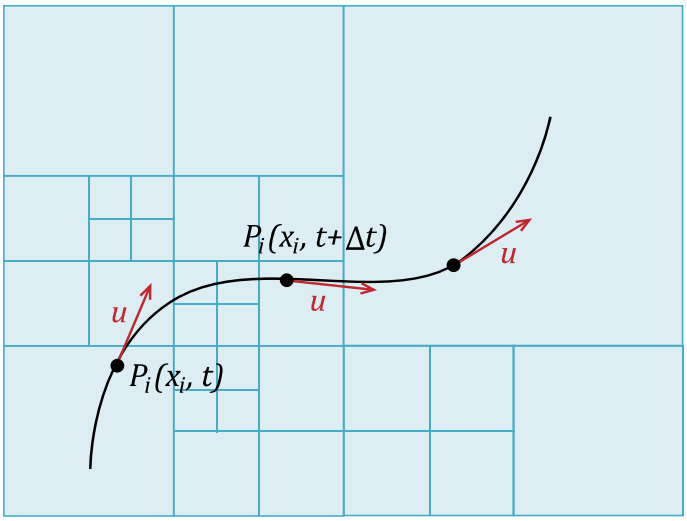



Figure 4.9 A schematic view of our hybrid system

# CHAPTER 5

# Implementation

In this chapter, we introduce an implementation model that derives concepts and general ideas described earlier to a practical simulation. In our implementation, we encapsulate the underlying infrastructure with the hierarchical classes, which provide effective layers of management. Major topics considering implementation on a three-dimensional array are described in this chapter.

## 5.1 Overview

We have implemented the octree grid on three-dimensional array which exploits several benefits over trees or other structures. First, the octree grid can be derived from an array with less modification from the prior model since the simulation using a fixed uniform grid is usually implemented on an array as well. Also, using an array is easily switchable between the adaptive octree grid and the fixed uniform grid by sharing the same array-based infrastructure and coding. Moreover, accessing the memory with arrays is fast (by instance indexing) and uses less overhead for grid refinements. The major drawback of using arrays is the memory usage that relies on its size, not the actual octree grid resolution. Since arrays use predefined memory allocation, therefore, the memory usage is not adaptive during the simulation. Although the grid is refined or optimized for speed; the simulation still uses the same memory as that for fixed uniform grid. However, if the memory usage is not a major constraint then using an array is a preferable choice.

## 5.2 Storing Octree Grid on Array

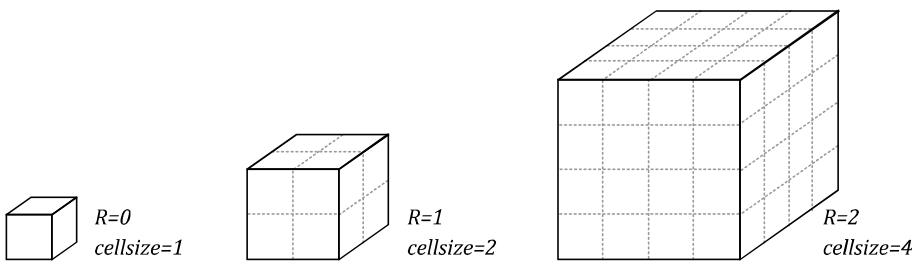



Figure 5.1 Cell size with various ranks

In this section, we introduce a model for storing the octree grid on an array structure and describe a direct mapping between the octree grid and array.

From our definition, a *"cell"* is a fundamental unit of the octree grid while an *"element"* is a fundamental unit of the array. Since the octree grid contains cells with various possible sizes, we define a variable called *"rank"* (denote as $R$) to identify their size:

$$cellsize_R = \Delta x = \Delta y = \Delta z = (cellsize_0)2^R \qquad \text{Equation 5.1}$$

$cellsize_R$ means cell size at an arbitrary rank $R$, where $R$ is an any positive number e.g., 0, 1, 2, 3, … $\Delta x, \Delta y$ and $\Delta z$ are dimensions of cell which should be equal for cubic cells.

Smallest cells are called "*base cells*". Base cells have a size that allocated to an array element. A cell with rank $R = 0$ (denoted as $cellsize_0$) is a base cell. A cell with higher rank has its size a power of 2 of its base size. This procedural produces a hierarchical grid with a subdivision of $2^3$.

A base cell contains only one array element while cells with higher ranks contain multiples (see Figure 5.1). Each cell contains a rank along with its individual fluid values (i.e., pressure, density, velocity); hence, a single array element of each cell is sufficient for storing the cell's fluid values. We define a *"leader element"* as a representative of all array elements in a cell to stores such fluid quantities. In this implementation, the bottom left corner element is chosen to be a leader element. If there exists a cell with rank $R$ that contains an arbitrary element with an index of $\boldsymbol{x} = (x, y, z)$ then the index of a leader unit (denotes as $\boldsymbol{x}^*$) of that cell can be found by:

$$\boldsymbol{x}^* = \boldsymbol{x} - \boldsymbol{x}\, mod\, (2^R) \qquad \text{Equation 5.2}$$

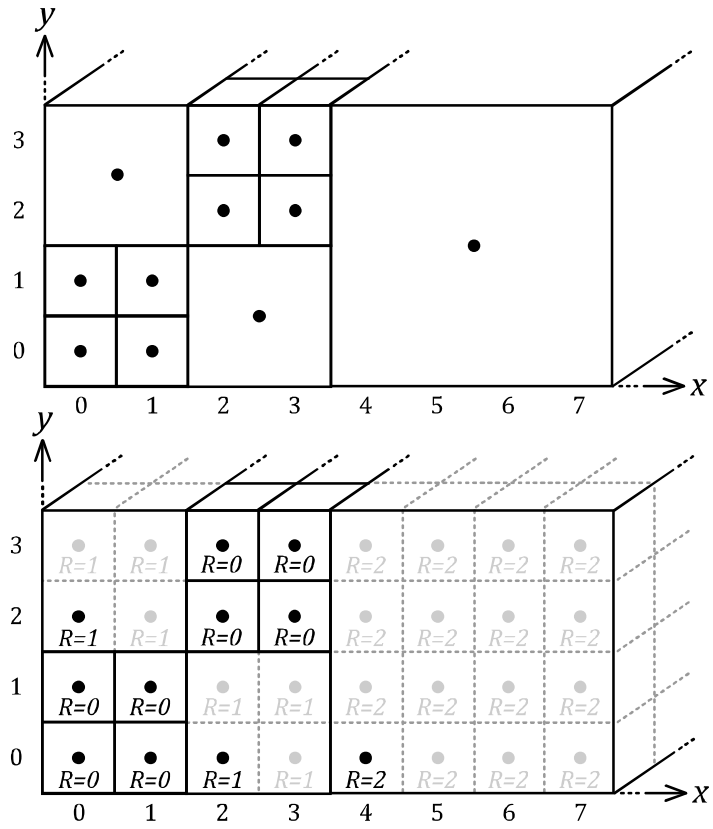



Figure 5.2 Above: logical representation of octree grid. Below: model for storing octree grid on 3D array.

Figure 5.2 (Below) illustrates our model for storing the octree grid on a three-dimensional array. Each cell may contains many array elements depends on its size (defined by its rank) but only one element is selected as a "*leader element*" for storing the fluid quantities of that cell. This model is encapsulated under the infrastructure of layers, hidden array and underlying management from outside direct access. Figure 5.2 (Above) illustrates our encapsulated octree grid which is a common perception for higher level implementation.

### 5.3 Recursive Cell Retrieval

Several simulation steps need cell retrieval as a part of their operations. For example, the added forces step needs to access all cells in a domain in order to update their cell's velocity.

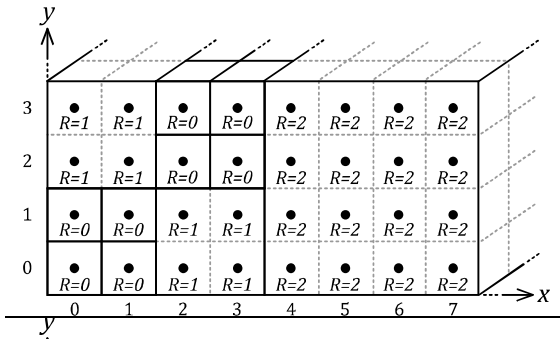
For-looping is a common approach to access all array elements one by one. It works well if most of the elements store data. However, in our array, only a leader element per cell stores valid fluid quantities, which is usually sparse over domain, hence, for looping in this particular grid structure is time wasting.

Instead of accessing elements one by one, we use a recursive function to access only some specific elements those are likely to be leaders ranging from the highest rank to the lowest rank recursively. Equation 5.3 specify cells those are likely to be leaders of rank $N$, where $N$ is any positive number ranging from the highest rank to the lowest rank: $N = \{R_{max}, R_{max} - 1, \dots, 2, 1, 0\}$. Whenever there exist an array element with an index $\boldsymbol{x}$ and a rank $\boldsymbol{R}$ that satisfy both Equation 5.3 and Equation 5.4 then that element is a leader element of that particular cell.

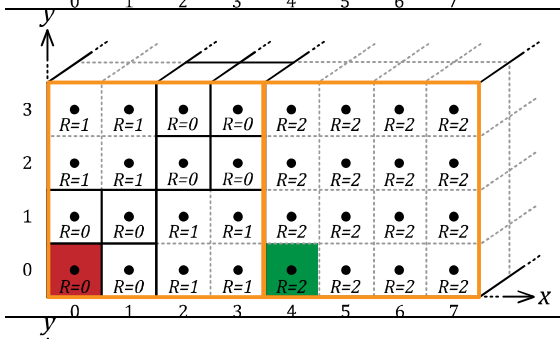
$$\boldsymbol{x}\ mod\ (2^N) = 0 \qquad \text{Equation 5.3}$$

$$R = N \qquad \text{Equation 5.4}$$

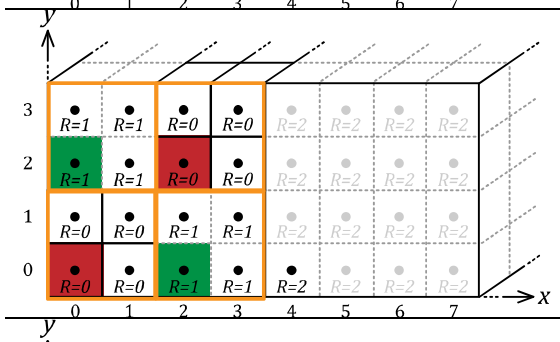
Figure 5.3 is an example scenario to demonstrate the sequence of recursive cell retrieval. The recursive function starts with N=2 and terminate itself whenever a leader element is found.




Initial : octree grid with unknown leader elements



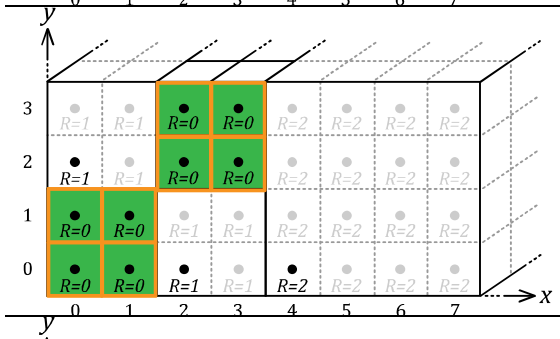

$N = 2$



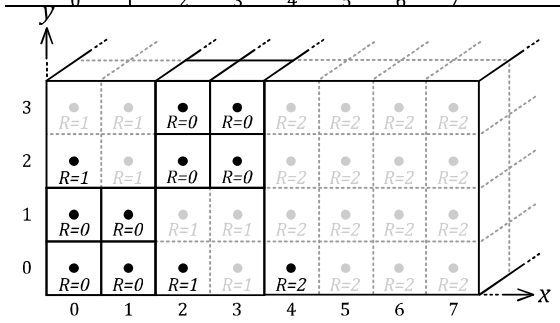

$N = 1$

$N \neq R$

$N = R$




$N = 0$




Final : all leader elements are found (highlighted units)

Figure 5.3 Sequence of finding leader elements using recursive approach

## 5.4 Linear Interpolation

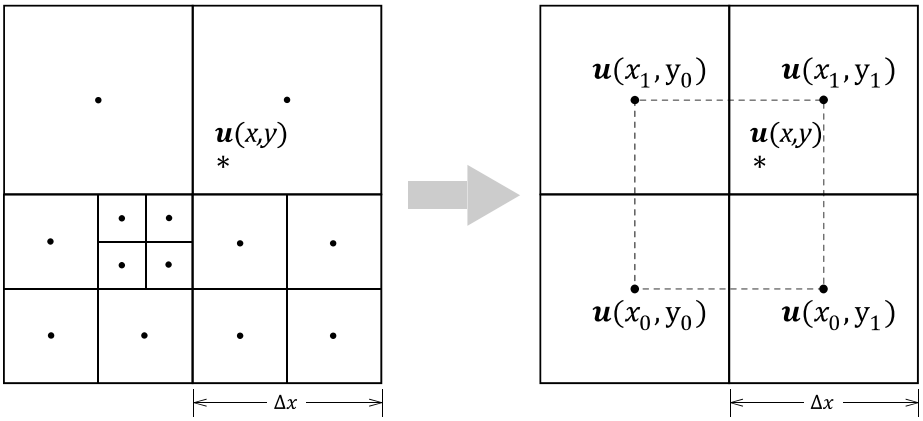



Figure 5.4 Normalizing the grid for data interpolation

Since fluid quantities are stored discretely at each cell center, the interpolation is essential in order to retrieve continuous quantities. There are several interpolation methods. However, in this implementation we use a *"linear interpolation"* since it is fast and simple and have sufficient accuracy for visual simulation. However, the octree grid is adaptive and has non-uniform structure; thus, an interpolating formula cannot be applied directly but must be constructed in an adaptive way for a particular octree region.

Refer to Figure 5.4(b), if we want to obtain a velocity $\boldsymbol{u}(x, y)$ at an arbitrary target point marked as ✱ then velocity of the neighboring cells must be interpolated. In general, bilinear interpolation interpolates between 4 points (for 2D grids) and trilinear interpolation interpolates between 8 points (for 3D grids). In practical, neighboring cells and their positions on the octree grid are adaptive, such as the one shown in Figure 5.4(a); hence, complicated for handling with these typical interpolations. Therefore, we must first mathematically normalize the grid for easily interpolation. Figure 5.4(a) is the original octree grid while Figure 5.4(b) is a normalized one. The processes are detailed as follows.

Let a cell that contains a target point ✱ have a rank $R$. In the case that every neighboring cell have the same rank $R$, then four cells under a rectangular perimeter are chosen as interpolating cells, or eight cells under cubical perimeter for

trilinear interpolation. Interpolation in this manner is straightforward (see below). However, if there exists a cell within the perimeter such that its rank is not equal to $R$ then that cell needs normalization before any interpolation can be performed.

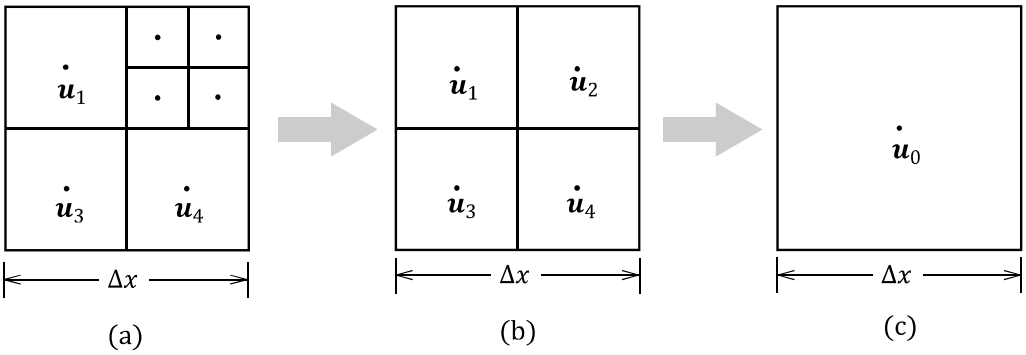



Figure 5.5 Recursively merging cells into a preferred size

Normalization is a process to obtain an imaginary cell with rank $R$. Figure 5.5 illustrate the steps of this process. Suppose that we prefer a cell with rank $R$ (Figure 5.5(c)), but current cells have rank $R-1$ (Figure 5.5(b)), then a normalization is performed. Four neighboring cells with rank $R-1$ are imaginarily merged to form a single preferred rank $R$ cell (Figure 5.5(c)). Its fluid quantities are obtained by arithmetic averaging of four pre-merge neighboring cells. For example, a velocity of a merged cell is:

$$\boldsymbol{u}_0 = \frac{\boldsymbol{u}_1 + \boldsymbol{u}_2 + \boldsymbol{u}_3 + \boldsymbol{u}_4}{4}$$

If there exists a hierarchical subdivision, for example, the upper right corner of grid shown in Figure 5.5(a), then recursive normalization for that level is performed.

Overall, Figure 5.4(a) shows an original octree grid while Figure 5.4(b) is a normalized one. Any cells within a rectangular perimeter are normalized to the same rank of the cell that contains a target point ✱. So far, the grid is ready for interpolation.

Refer to Figure 5.4(b), we obtain velocity $\boldsymbol{u}(x,y)$ at the arbitrary target point ✱ by the following bilinear interpolation.

$$\boldsymbol{u}_{(x,y)} = \frac{\boldsymbol{u}_{(x_0,y_0)}(x_1-x)(y_1-y) + \boldsymbol{u}_{(x_0,y_1)}(x_1-x)(y-y_0) + \boldsymbol{u}_{(x_1,y_0)}(x-x_0)(y_1-y) + \boldsymbol{u}_{(x_1,y_1)}(x-x_0)(y-y_0)}{(\Delta x)^2}$$

or equivalently, in matrix operations:

$$\boldsymbol{u}(x,y) = \frac{[x_1 - x \quad x - x_0]\begin{bmatrix}\boldsymbol{u}(x_0,y_0) & \boldsymbol{u}(x_0,y_1)\\ \boldsymbol{u}(x_1,y_0) & \boldsymbol{u}(x_1,y_1)\end{bmatrix}\begin{bmatrix}y_1 - y\\ y - y_0\end{bmatrix}}{(\Delta x)^2}$$

Note that since the cell's dimension is cubic; thus, we can assume $\Delta x = \Delta y$ for the above equations. For three-dimensional domains, trilinear interpolation is applied instead which can be performed in a similar way.

### 5.5 Discretization of Differential Operations

According to the Poisson equations described chapter 4.4, their implementation can be done in a recursive way. For example, the $x$ components of $\nabla \cdot \boldsymbol{u}$ is obtained by subtraction between left adjacent cells and right adjacent cells and factored by their faces sizes (see Equation 4.14), where left and right adjacent cells are obtained by recursive cell retrieval.

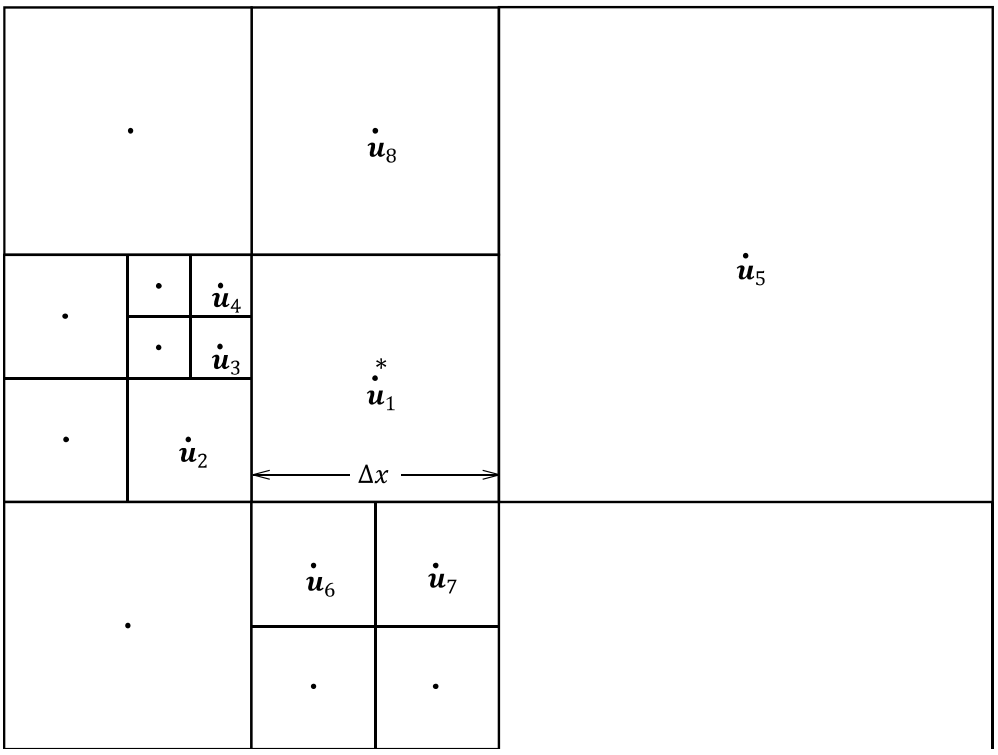



Figure 5.6 An example of quardtree structure

In the case of Figure 5.6, the $\nabla \cdot \boldsymbol{u}$ , $\nabla^2 \boldsymbol{u}$ and $\nabla^2 p$ of a cell marked as * are discretized as follows.

$$\nabla \cdot \boldsymbol{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{u_5 - \left(\frac{u_2}{2} + \frac{u_3}{4} + \frac{u_4}{4}\right)}{\partial x} + \frac{v_8 - \left(\frac{v_6}{2} + \frac{v_7}{2}\right)}{\partial y}$$

$\nabla^2 \boldsymbol{u} = (\nabla^2 u, \nabla^2 v)$ where

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$
$$= \frac{\left(\frac{u_2}{2} + \frac{u_3}{4} + \frac{u_4}{4} - u_1\right) + (u_5 - u_1)}{(\partial x)^2} + \frac{(u_8 - u_1) + \left(\frac{u_6}{2} + \frac{u_7}{2} - u_1\right)}{(\partial y)^2}$$
$$\nabla^2 v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}$$
$$= \frac{\left(\frac{v_2}{2} + \frac{v_3}{4} + \frac{v_4}{4} - v_1\right) + (v_5 - v_1)}{(\partial x)^2} + \frac{(v_8 - v_1) + \left(\frac{v_6}{2} + \frac{v_7}{2} - v_1\right)}{(\partial y)^2}$$

$$\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = \frac{\left(\frac{p_2}{2} + \frac{p_3}{4} + \frac{p_4}{4} - p_1\right) + (p_5 - p_1)}{(\partial x)^2} + \frac{(p_8 - p_1) + \left(\frac{p_6}{2} + \frac{p_7}{2} - p_1\right)}{(\partial y)^2}$$

# CHAPTER 6

## Results and Discussion

In order to evaluate our refinement method, we have constructed and compared results in different simulation scenarios i.e., varying the view-dependent coefficient (Section 6.2), varying the camera's viewing angle (Section 6.3), varying the output resolution (Section 6.4) and finally simulation with and without particles (Section 6.5).

The efficiency of our method is measured by comparing both computational cost and visual result to the simulation without the view-dependent adaptive grid refinement. Computational cost is compared by measuring the simulation time in seconds per frame while visual results are compared by animation quality. Details of our experiments and their evaluation are described in the following subsections.

### 6.1 Simulation Environment

Figure 1.1 illustrates the environment used in our experiments, where the simulation domain is constructed on a $128 \times 80 \times 48$ grid size and a camera is placed nearby for rendering the output scene. All experiment results reported in this article were performed on a machine with dual core CPU 2.40 GHz and 2 GB of RAM. Typical simulation times using our method were about 10 seconds per frame when performed on an octree grid with approximately 100,000 nodes and 340,000 particles.

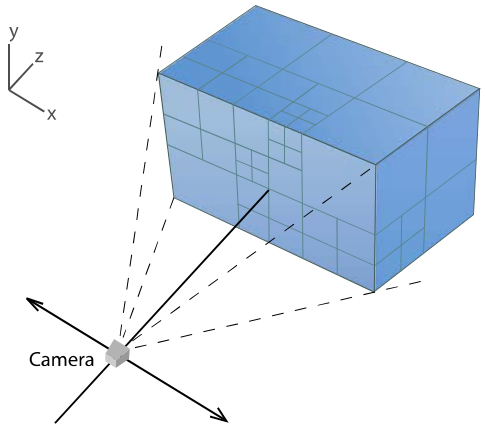
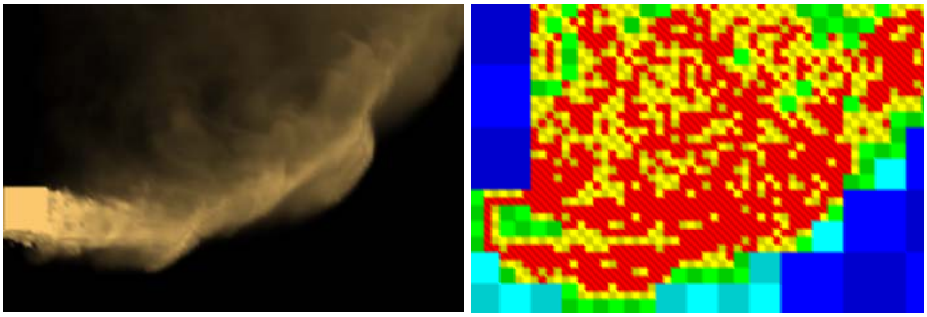


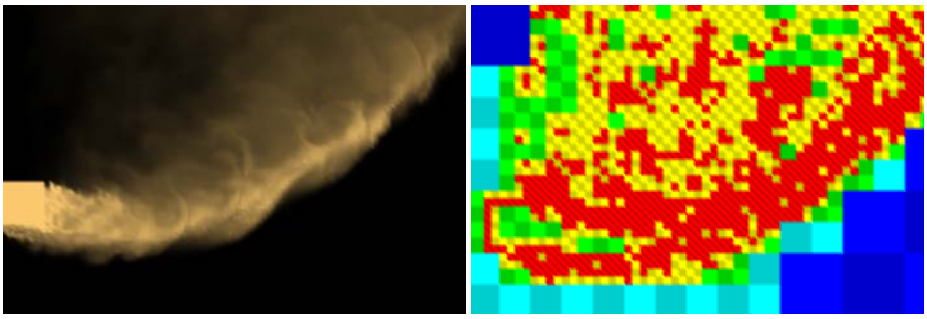

Figure 6.1 Environment for smoke simulation on an octree grid with VD-refinement

## 6.2 Varying View-Dependent Coefficient ( $\tau$ )

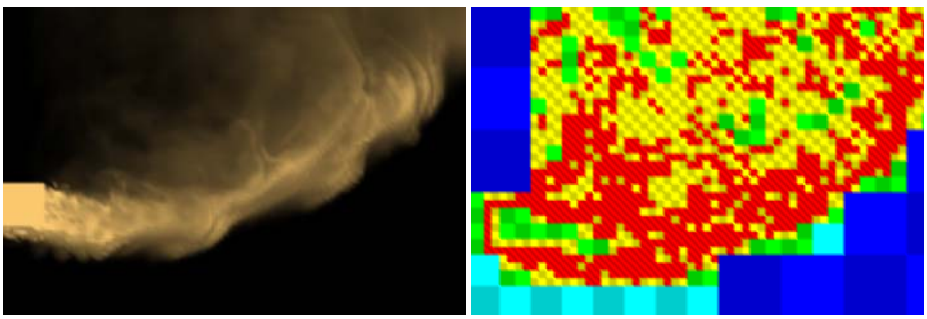
We have demonstrated the effect of the view-dependent adaptive grid refinement by varying only the view-dependent coefficient ($\tau$) while other parameters are fixed. Figure 6.2 are the comparison side-by-side between the visual results (left sides) and the cut-away views (right sides) showing their octree grid sturcture. In this scenario, smoke is injected into a domain vertically with an amount of upward force. By increasing the view-dependent coefficient, the number of large nodes is increased while the number of small nodes is decreased. Table 6.1 shows the timing and the number of nodes of Figure 6.2.



No View-Dependent Refinement



$\tau = 1$



$\tau = 2$

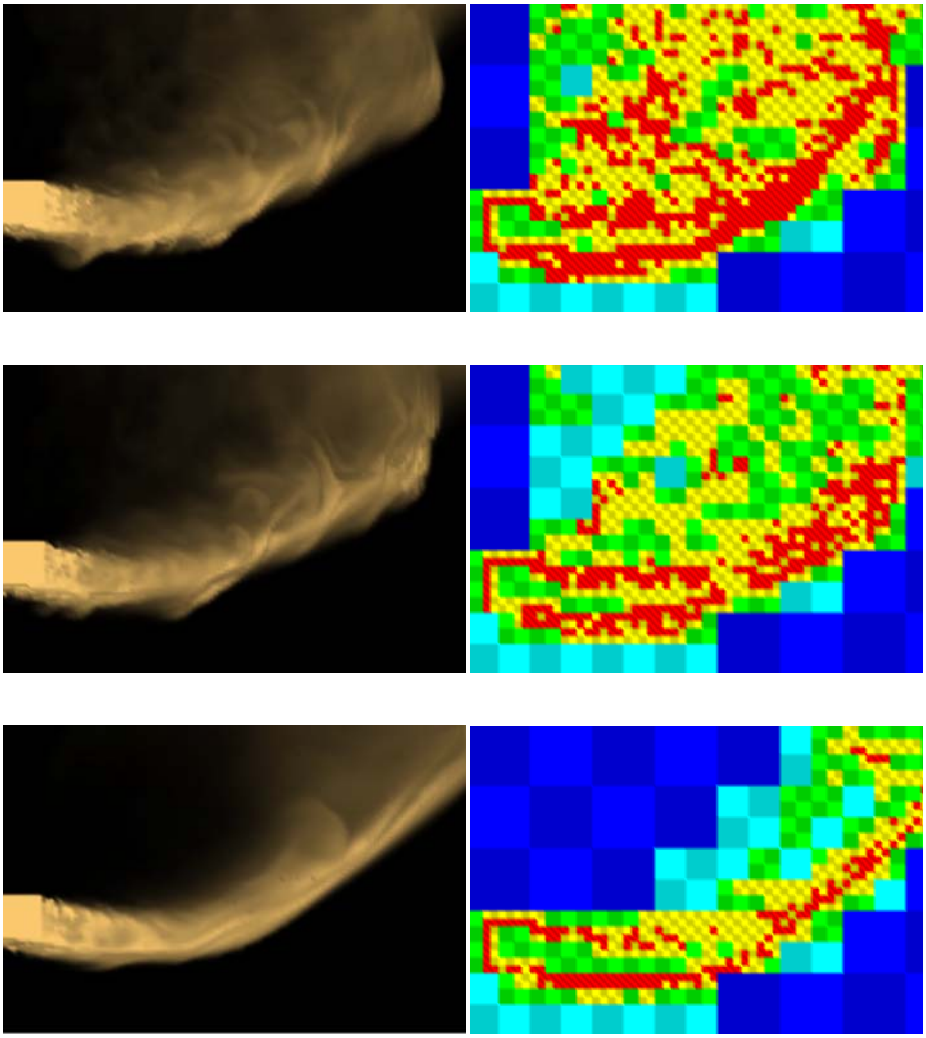



Figure 6.2 Comparison of different VD-refinement

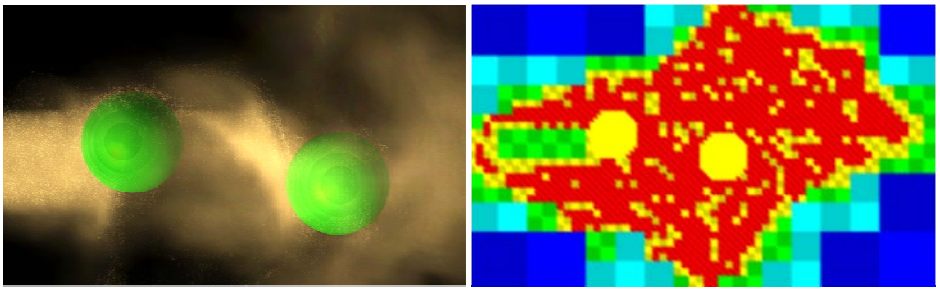
| VD Coefficient ($\tau$) | Number of Nodes | Time per frame (Sec) | Speed-up (%) |
|---|---|---|---|
| No VD Refine. | 111,868 | 9.10 | - |
| 1 | 97,770 | 8.47 | 7.28 |
| 2 | 87,683 | 8.00 | 14.10 |
| 5 | 66,172 | 7.19 | 26.57 |
| 10 | 49,176 | 6.33 | 44.13 |
| 20 | 27,700 | 6.62 | 37.58 |

Table 6.1 Number of nodes and timing in different VD coefficient

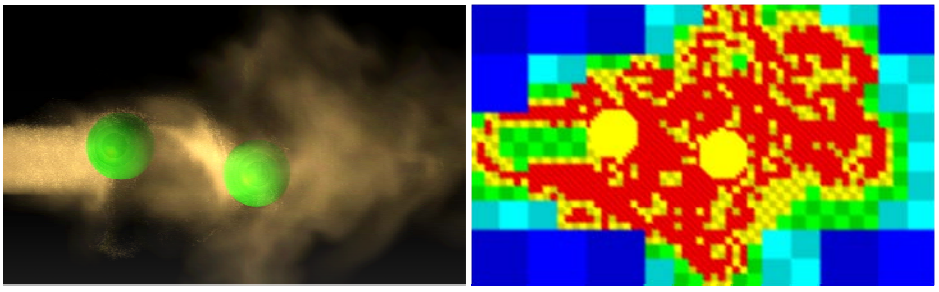
Applying a greater view-dependent coefficient reduces the number of cells in the simulation domain. Simulation achieves more frame rates but detail loss is relatively greater. On the other hand, if a lower view-dependent coefficient is applied, more details are preserved but with higher computation cost as well. Grid refinement with a view-dependent coefficient of 1 is an optimal weighting between details and computational cost since the amount of optimization matches the proportion of grid perspective. However, in our experiment, the view-dependent coefficient can be assigned up to approximately 5 before detail loss becomes noticeable. This is because the foreground smoke, which usually has higher detail, occludes other smoke with lower detail behind.

### 6.3 Varying Camera's Viewing Angle (FOV)

In this scenario, smoke is moving rightward and being distracted by two oscillating spheres as it passes through. In Figure 6.3, we have changed the camera's viewing angle (FOV) from wide to narrow in order to demonstrate the effective of our refinement method on various viewing. According to this experiment, when widen the camera's viewing angle, fluid details are coarsened. However, since all visual scenes accordingly get smaller due to the perspective effect, the detail of the fluid on output screen is still preserved. The corresponding timings of Figure 6.3 are reported in Table 6.2.



FOV = 20°



FOV = 30°

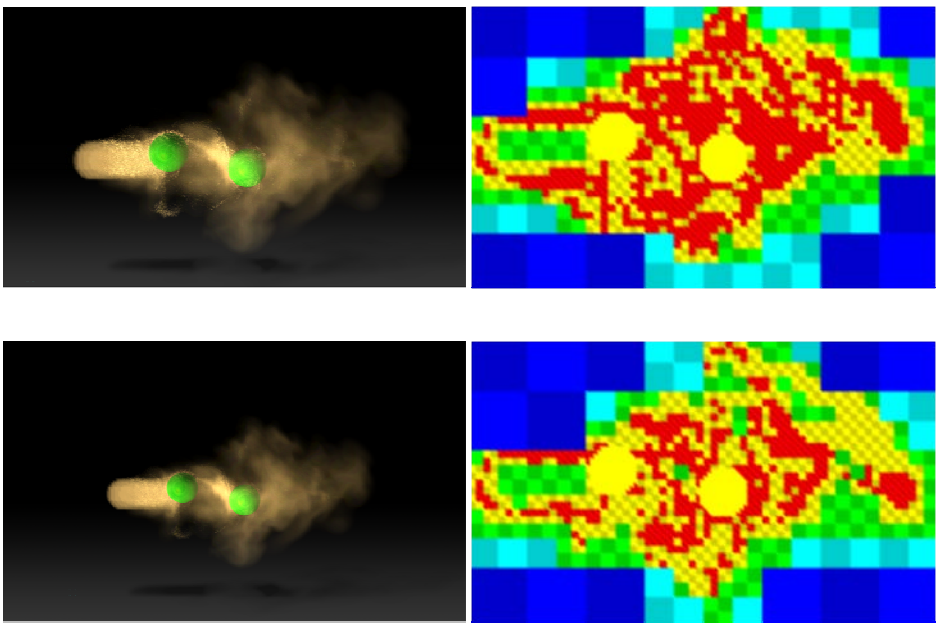



Figure 6.3 Comparison of different camera FOV

| FOV (degree) | Number of Nodes | Time per frame (Sec) | Speed-up (%) |
|---|---|---|---|
| No VD Refine. | 104,075 | 10.34 | - |
| 20 | 100,833 | 9.73 | 5.90 |
| 30 | 84,049 | 8.36 | 19.13 |
| 50 | 68,052 | 7.63 | 26.24 |
| 60 | 42,510 | 6.70 | 35.23 |

Table 6.2 Timing of the simulation using our method with various FOV

### 6.4 Varying Resolution Ratio (Ø)

We have simulated a turbulence flow over a cylindrical rod and vary only the output resolution to demonstrate the result of applying different resolution ratio. Figure 6.4 compares the results of applying different output resolution and the corresponding timings of Figure 6.4 are shown in Table 6.3.

Accoarding the the results, lowering the output resolution (increasing the resolution ratio) results in a coarser grid and speeds up the simulation. However, the

simulation preserves its details whether the output resolution has changed, since with our method, the grid resolution is refined related to the output resolution.

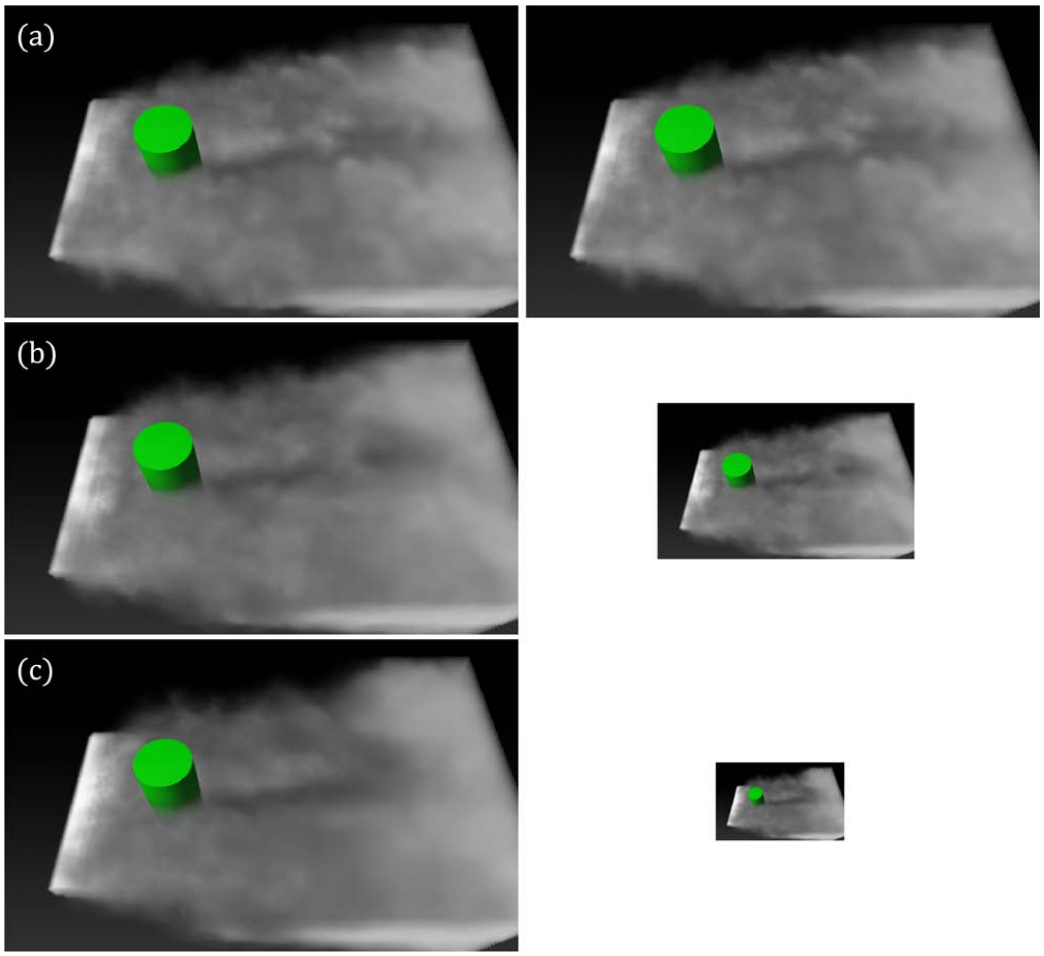



Figure 6.4 Turbulence flows over a cylindrical rods at different output resolutions. Actual output size (right) and enlarged size (left).

| Scene | Resolution Ratio (Ø) | Time per frame (Sec) | Speed-up (%) |
|---|---|---|---|
| - | No VD-Refine. | 10.62 | - |
| Figure 6.4(a) | 0.5 | 9.88 | 6.97 |
| Figure 6.4(b) | 1 | 9.05 | 14.78 |
| Figure 6.4(c) | 2 | 8.14 | 23.35 |

Table 6.3 Timing of results shown in Figure 6.4

## 6.5 Rendering with Particles

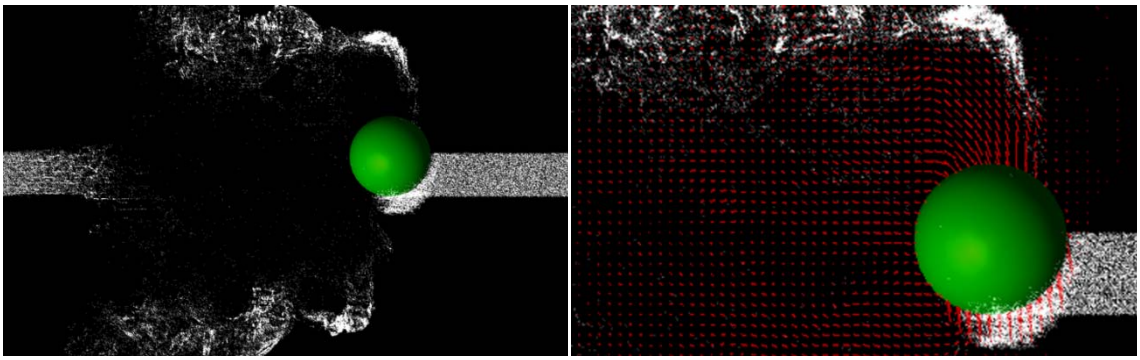


Figure 6.5 Bouncing sphere through a sheet of particles

We have integrated a particle system into our view-dependent adaptive grid and tested in various scenarios. Figure 6.6 compares the difference between using billboards only (right) and coupling billboards with particles (left). Figure 6.7 illustrates that particles and billboards together enhance the fluid motion and visual result. Figure 6.5 is a simulation rendered with particles only to illustrate that particles can conform well to curve and boundaries. Additional results including others without particles are shown in Figure 6.8 and their corresponding timings are shown in Table 6.4. In our experiments, approximately 2,000 particles are generated each frame, taking about 0.8% of all processing time. The timings of simulation using our method in different scenarios are shown separately in Table 6.4. All tests are performed with view-dependent coefficient $\tau = 1$ and camera's FOV=45$^{\circ}$. Timings are measured in second/frame. Note that additional snapshots of our experiment are shown in the Appendix.

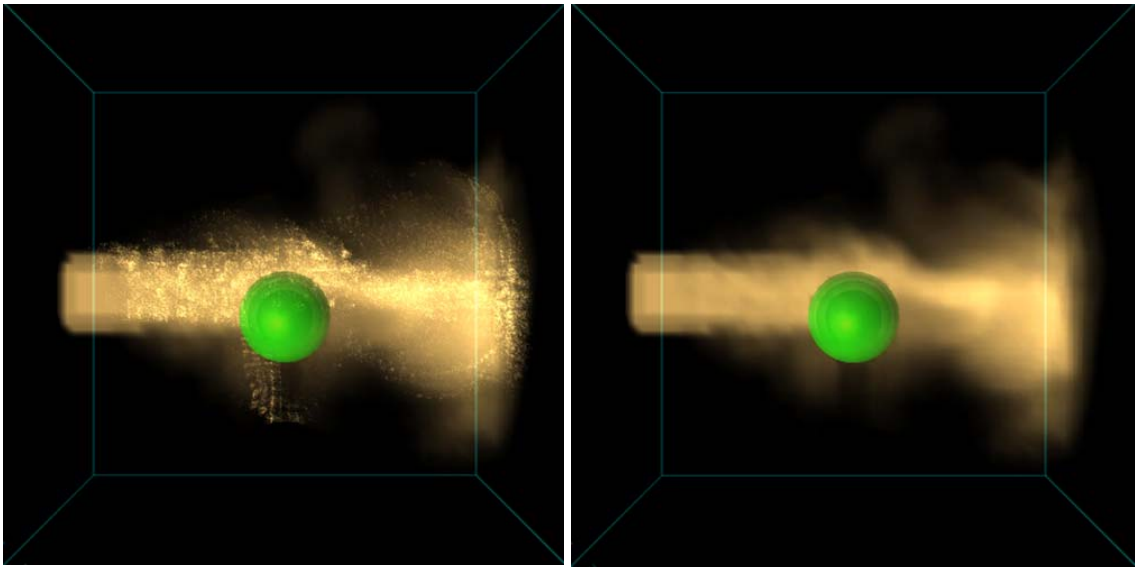


Figure 6.6 Comparison between with and without particles

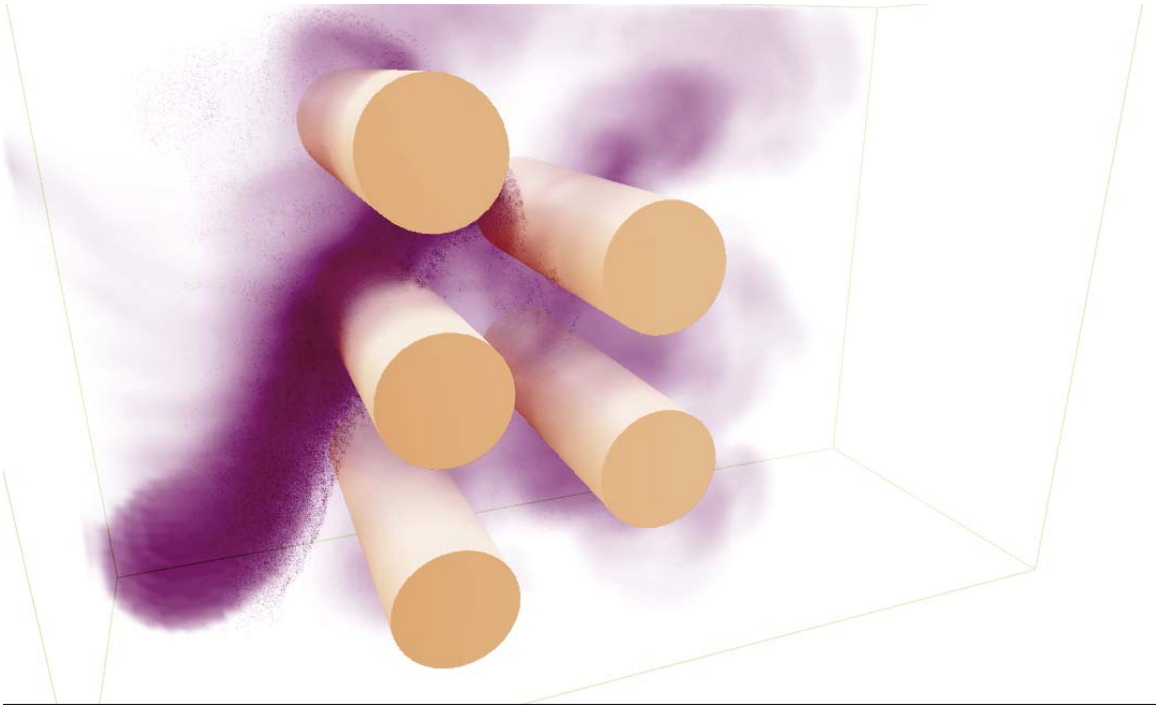


Figure 6.7 Dynamic flows through cylindrical rods.
Result is rendered with approximately 340k particles.

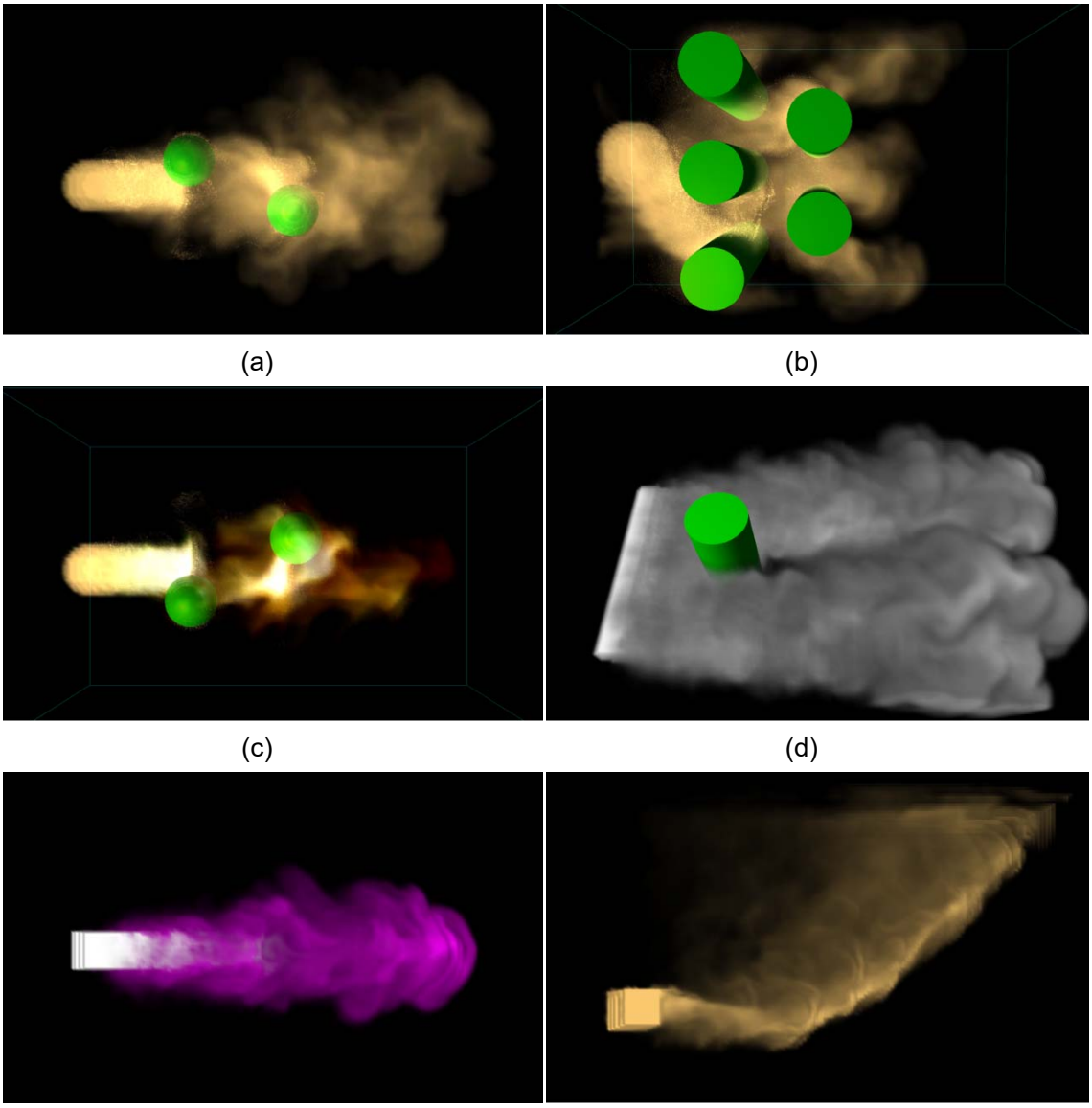



Figure 6.8 Screen shots of our experimental results in various scenes

| Scenes | Refine (s) | Discretize (s) | Particle (s) | Total (s) | Speed-up (%) |
|---|---|---|---|---|---|
| Figure 6.5 | .153 | 3.48 | .053 | 3.69 | - |
| Figure 6.8(a) | .183 | 9.47 | .078 | 9.73 | 5.90 |
| Figure 6.8(b) | .147 | 9.72 | .063 | 9.85 | 5.98 |
| Figure 6.8(c) | .131 | 8.90 | .057 | 9.09 | 5.39 |
| Figure 6.8(d) | .166 | 9.69 | - | 9.86 | 7.38 |
| Figure 6.8(e) | .174 | 8.06 | - | 8.23 | 7.16 |
| Figure 6.8(f) | .181 | 8.29 | - | 8.47 | 7.28 |

Table 6.4 Timing in s/frame in various scenes. Speed-up is compared to another one without VD refinement.

The result shows that particles can significantly enhance the visual result and reduce the motion artifacts caused by dynamic grid refinement. With particles, observers hardly notice that the simulation background is a grid-based approach. In addition, particles are applicable to be integrated into a typical grid-based simulation, since the time for processing the particles is relatively low with respect to the overall simulation time.

# CHAPTER 7

# Conclusion and Future Work

In this section, we have concluded the core idea of our proposed method, the problems we found in this research and the future works.

## 7.1 Conclusion

Since a current smoke simulation using octree grid is optimized for detail but not optimized for viewing; thus, we have presented a “view-dependent adaptive grid refinement” which is an improved refinement method that optimizes the grid for both detail and viewing. The refinement conditions with adaptive thresholds are constructed, incorporating viewing information with fluid variation. With our method, the amount of grid refinement is controlled adaptively by means of cell-to-camera distance ($r$), view-dependent coefficient ($\tau$), camera’s viewing angle ($FOV$) and the resolution ratio ($\emptyset$).

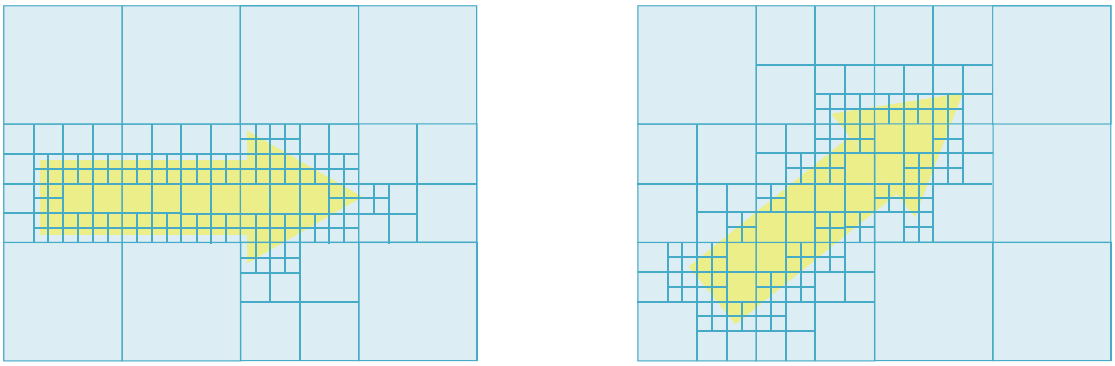
We have shown that optimizing the grid with the view-dependent adaptive grid refinement speeds up the simulation as well as preserve fluid details. Several parameters have been adjusted and tested on different scenarios to demonstrate the efficiency of the proposed grid refinement in various environments. According to the results, the method has successfully optimized the grid for both viewing angle and details. Visual details are decreased corresponding to the reduction of the total number of octree nodes within a domain, which results in lower computational cost consumption. We have also shown that the method can be integrated with a particle system to enhance visual results and reduce motion artifacts.

Overall, this approach provides a flexible framework for fluid simulation optimization that can be applied for variety of simulation environments and real-life applications such as special effect in games, movies and advertisement.

## 7.2 Future Works

For future work, we plan to further speed up the simulation by culling the occlusion regions and invisible areas such as smoke behind the obstacles and smoke behind the dense smoke. The invisible areas do not have to contain fine details; therefore, unnecessary processing time can be reduced.

We have found that the simulation speed-up is depends on the grid orientation. If fluid flows through the domain in a direction that is orthogonal to the grid orientation (i.e., horizontally or vertically to the grid orientation) then fewer cells are subdivided and the simulation is performed faster but if fluid flow diagonally across the domain, more cells are subdivided and the simulation is performed slower. See Figure 7.1 for example.



(a) (b)

Figure 7.1 (a) Fluid flow along the grid orientation.
(b) Fluid flow diagonal to the grid orientation.

To address this, we plan to replace the octree grid with any other structure that is independent to the grid orientation such as unstructured tetrahedral meshes or unstructured grid.

In addition, we plan to extend our method to larger scenes with an adaptive level-of-detail for dynamic viewing instead of a single camera view. We have found that the grid needs latency for iteratively refine to the appropriate optimized structure; thus, rapidly changing the camera's viewing e.g., translation, rotation or even zooming might cause discontinuity artifacts and detail loss in a period of time. Therefore, view can be changed with a limited speed relative to the simulation frame rates. We have planned to address this limitation by predicting the camera movement and refine the grid in advance by using the camera's velocity.

## References


[1] Foster, N., Realistic Animation of Liquids, Graphical Models and Image Processing 58 (September 1996): 471-483.

[2] Stam, J., Stable fluids, Proceedings of the 26th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., p. 121–128.

[3] Fedkiw, R., Stam, J., and Jensen, H. W., Visual simulation of smoke, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, Citeseer, p. 15–22.

[4] Stam, J., Real-time fluid dynamics for games, Proceedings of the Game Developer Conference, Citeseer.

[5] Losasso, F., Gibou, F., and Fedkiw, R., Simulating water and smoke with an octree data structure, ACM Transactions on Graphics, 23 (August 2004): 457.

[6] Shi, L. and Yu, Y., Visual smoke simulation with adaptive octree refinement, Computer Graphics and Imaging, Citeseer, p. 13–19.

[7] Ament, M. and Straßer, W., Dynamic Grid Refinement for Fluid Simulations on Parallel Graphics Architectures, EUROGRAPHICS SYMPOSIUM ON PARALLEL GRAPHICS AND VISUALIZATION, Citeseer.

[8] Barran, B. A., View dependent fluid dynamics, Texas A&M University, 2006.

[9] Popinet, S., Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries, Journal of Computational Physics 190 (September 2003): 572-600.

[10] Nealen, A., Physically Based Simulation and Animation of Gaseous Phenomena in a Periodic Domain, I Can, (2001).

[11] Feldman, B. E., O'Brien, J. F., and Arikan, O., Animating suspended particle explosions, ACM Transactions on Graphics 22 (July 2003): 708.

[12] Goktekin, T. G., Bargteil, A. W., and O'Brien, J. F., A method for animating viscoelastic fluids, ACM Transactions on Graphics 23 (August 2004): 463.

[13] Zheng, W., Yong, J.-H., and Paul, J.-C., Simulation of bubbles, Graphical Models 71 (November 2009): 229-239.

[14] BERGER, M. and OLIGER, J., Adaptive mesh refinement for hyperbolic partial differential equations☆, Journal of Computational Physics 53 (March 1984): 484-512.

[15] BERGER, M. and COLELLA, P., Local adaptive mesh refinement for shock hydrodynamics, Journal of Computational Physics 82 (May 1989): 64-84.

[16] Adams, B., Pauly, M., Keiser, R., and Guibas, L. J., Adaptively sampled particle fluids, ACM Transactions on Graphics 26 (July 2007): 48.

[17] Lucy, L. B., A numerical approach to the testing of the fission hypothesis, The Astronomical Journal 82 (December 1977): 1013.

[18] Mathieu Desbrun and Marie-paule Gascuel, Smoothed Particles: A new paradigm for animating highly deformable bodies, Proceedings of EG Workshop on Animation and Simulation, Springer-Verlag, pp. 61-76.

[19] Simon Premzoe, Tolga Tasdizen, James Bigler, Aaron Lefohn, R. T. W., Particle-Based Simulation of Fluids, Computer Graphics Forum 22 (September 2003): 401-410.

[20] Enright, D., Marschner, S., and Fedkiw, R., Animation and rendering of complex water surfaces, ACM Transactions on Graphics 21 (July 2002).

[21] Matthias Müller, David Charypar, M. G., Particle-Based Fluid Simulation for Interactive Applications, Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, pp. 154-159.

[22] Müller, M., Solenthaler, B., Keiser, R., and Gross, M., *Particle-based fluid-fluid interaction*, ACM Press.

[23] Hong, W., House, D. H., and Keyser, J., Adaptive particles for incompressible fluid simulation, The Visual Computer, 24 (May 2008): 535-543.

[24] He Yan, Zhangye Wang, Jian He, Xi Chen, Changbo Wang, Q. P., Real-time fluid simulation with adaptive SPH, Computer Animation and Virtual Worlds, 20 (2009): 417-426.

[25] W Li, X Wei, A. K., Implementing Lattice Boltzmann Computation on Graphics Hardware, The Visual Computer, 19 (2003): 444-456.

[26] Feldman, B. E., O'Brien, J. F., and Klingner, B. M., Animating gases with hybrid meshes, ACM Transactions on Graphics, 24 (July 2005): 904.

[27] Feldman, B. E., O'Brien, J. F., Klingner, B. M., and Goktekin, T. G., Fluids in deforming meshes, Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '05, (2005): 255.

[28] Elcott, S., Tong, Y., Kanso, E., Schröder, P., and Desbrun, M., Stable, circulation-preserving, simplicial fluids, ACM SIGGRAPH ASIA 2008 courses on - SIGGRAPH Asia '08, (2008): 1-11.

[29] Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F., Fluid animation with dynamic meshes, ACM SIGGRAPH 2006 Papers on - SIGGRAPH '06, (2006): 820.

[30] Chentanez, N., Feldman, B. E., Labelle, F., O'Brien, J. F., and Shewchuk, J. R., Liquid simulation on lattice-based tetrahedral meshes, Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association, p. 219–228.

[31] Harlow, F. H. and Welch, J. E., Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface, Physics of Fluids, 8 (1965): 2182.

[32] Kim, B., Liu, Y., Llamas, I., and Rossignac, J., Advections with significantly reduced dissipation and diffusion., IEEE transactions on visualization and computer graphics, 13 (2007): 135-44.

[33] Selle, A., Fedkiw, R., Kim, B., Liu, Y., and Rossignac, J., An Unconditionally Stable MacCormack Method, Journal of Scientific Computing 35 (November 2007): 350-371.

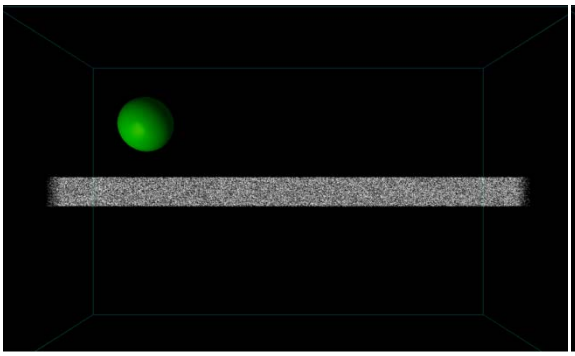
[34] Molemaker, J., Cohen, J. M., Patel, S., and Noh, J., Low Viscosity Flow Simulations for Animation, Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, (2008): 9-18.

[35] Zhu, Y. and Bridson, R., *Animating sand as a fluid*, ACM Press.

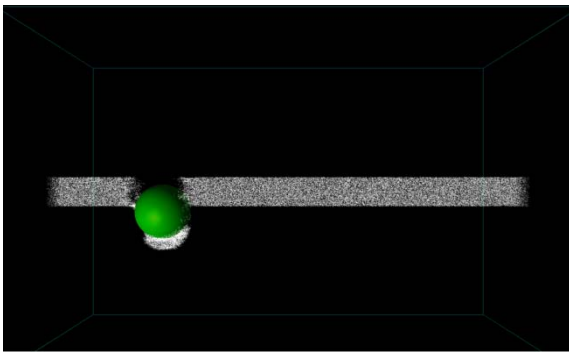
# Appendix

In this chapter, our experimental results in different scenarios are shown as a sequence of snapshots. All experiments are performed on an octree grid with a $128 \times 80 \times 48$ resolution. The control variables are VD-coefficient $\tau = 1$, camera's viewing angle FOV=45$^{\circ}$ and resolution ratio $\emptyset = 1$. Typical simulation times were about 10 seconds per frame, with approximately 100,000 nodes and 340,000 particles. The corresponding timings are shown in Table 6.4.

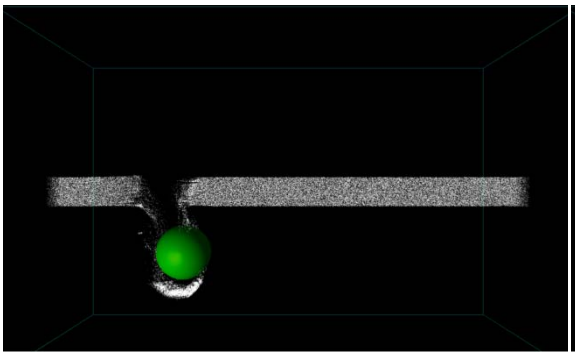
***Bouncing sphere through a sheet of particles***



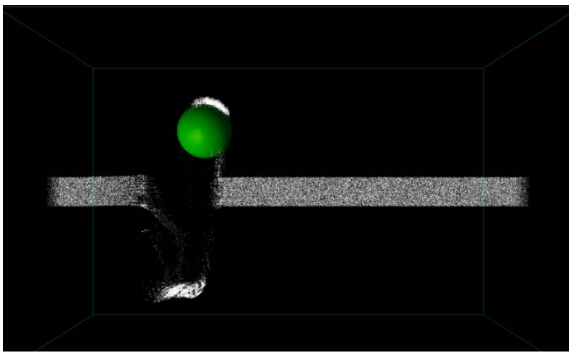
Frame =0



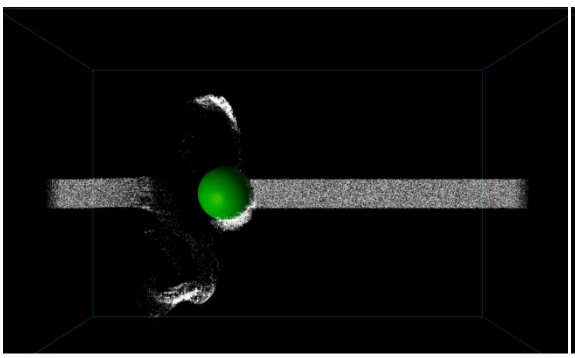
Frame =10



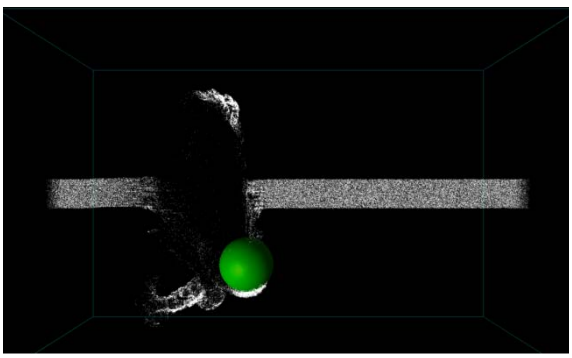
Frame =20



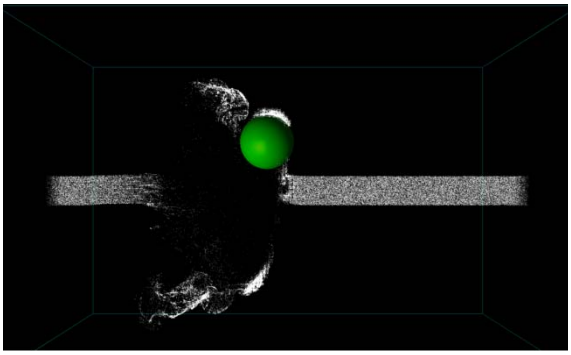
Frame =30



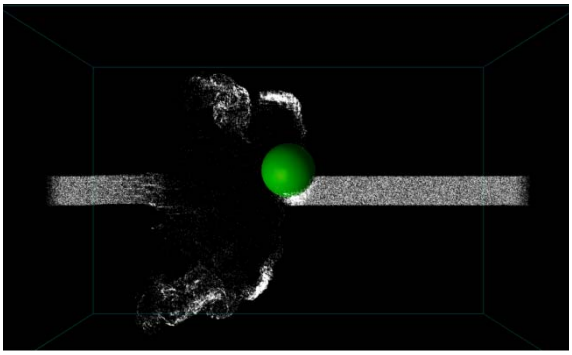
Frame =40



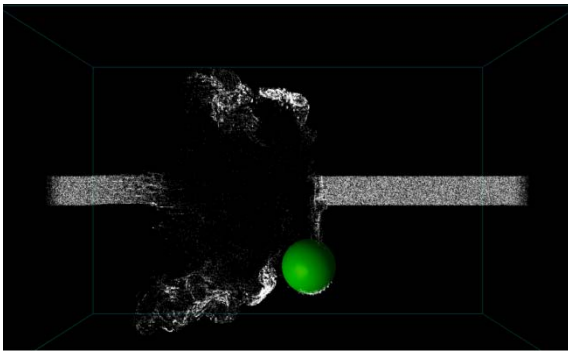
Frame =50



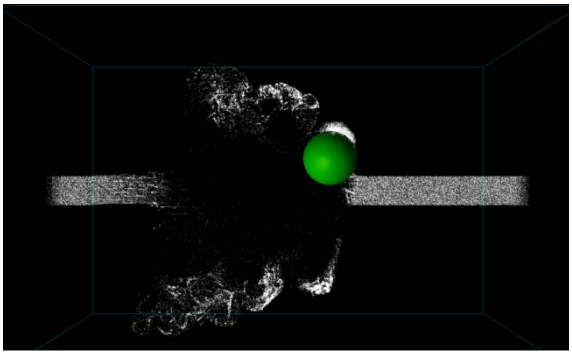
Frame =60



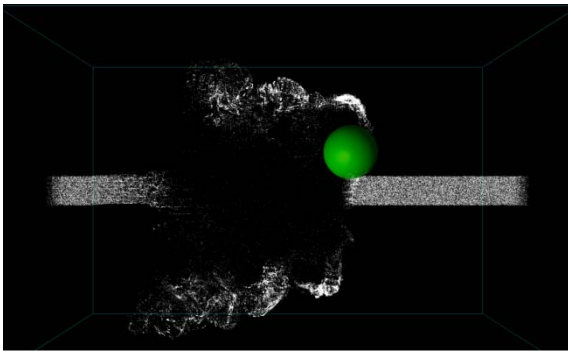
Frame =70



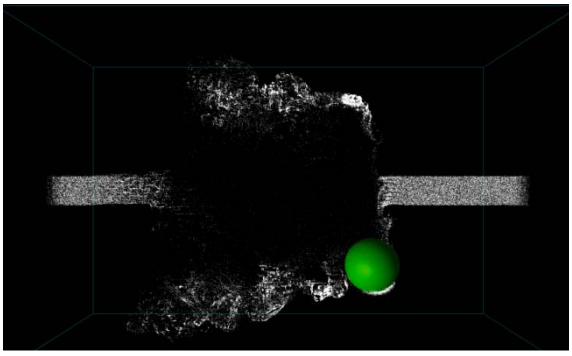
Frame =80



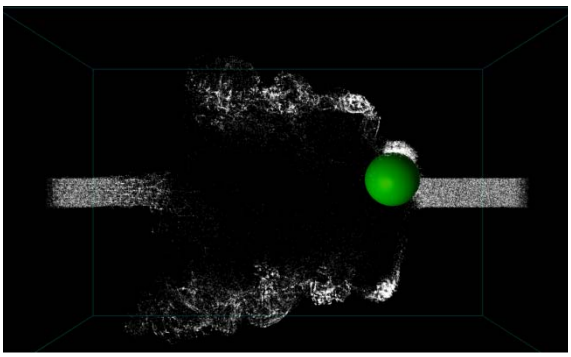
Frame =90



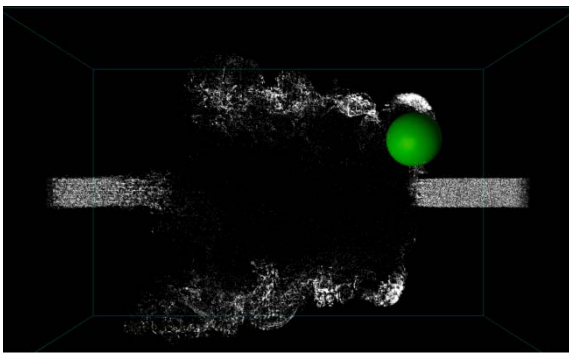
Frame =100



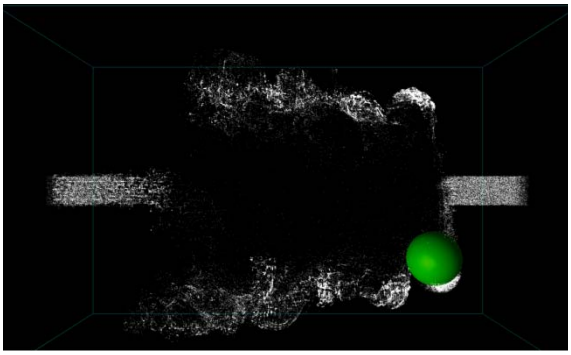
Frame =110



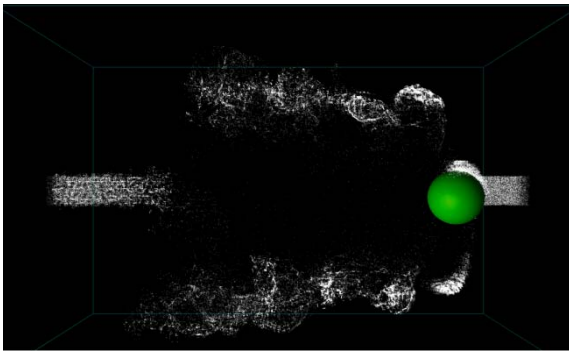
Frame =120



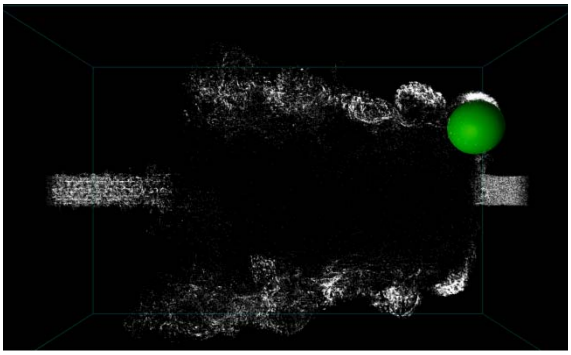
Frame =130



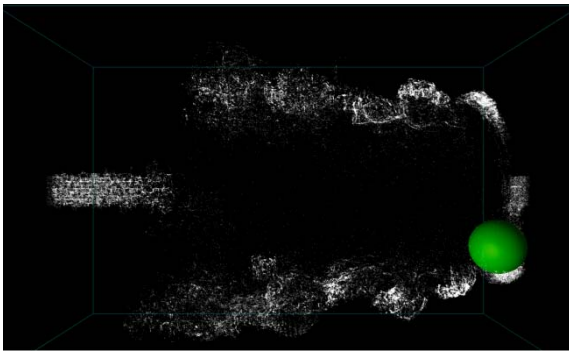
Frame =140



Frame =150

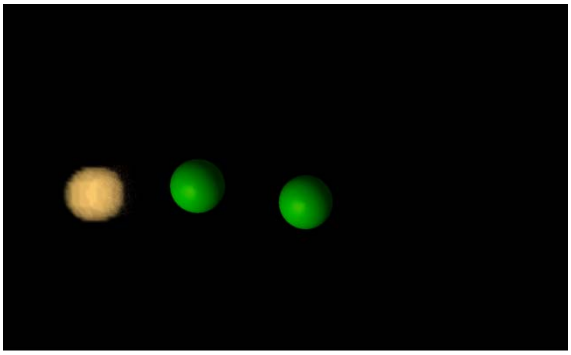


Frame =160

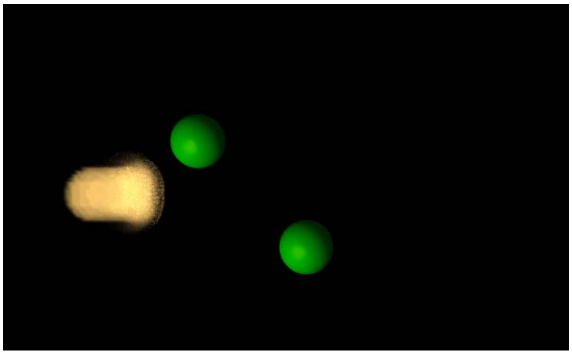


Frame =170

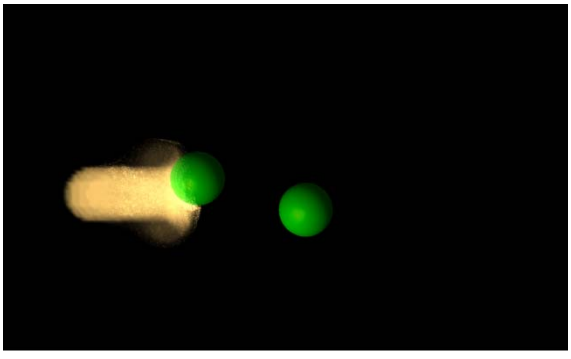
***Smoke animation with oscillating spheres.***



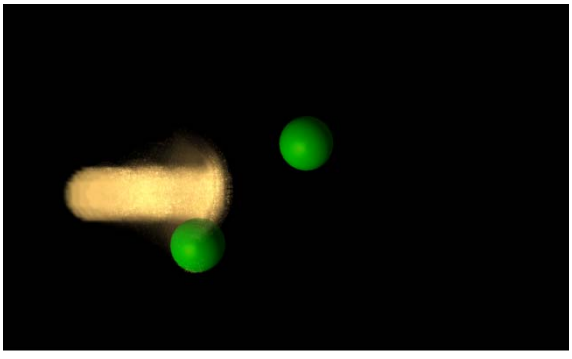
Frame =0



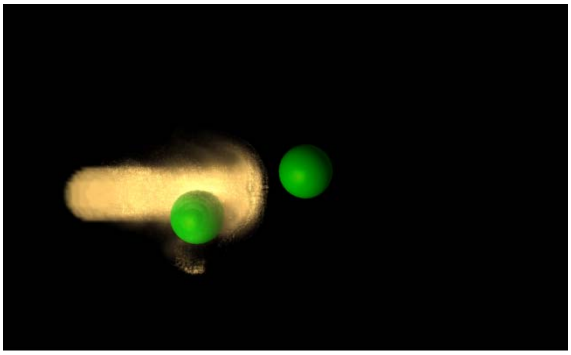
Frame =10



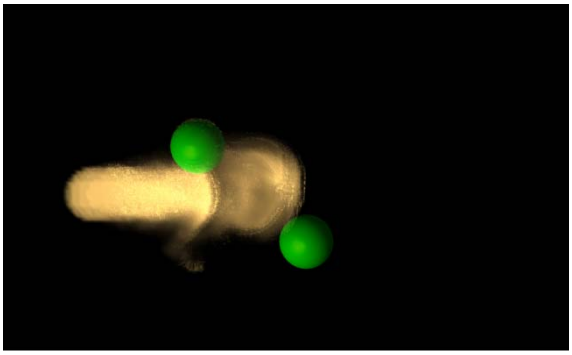
Frame =20



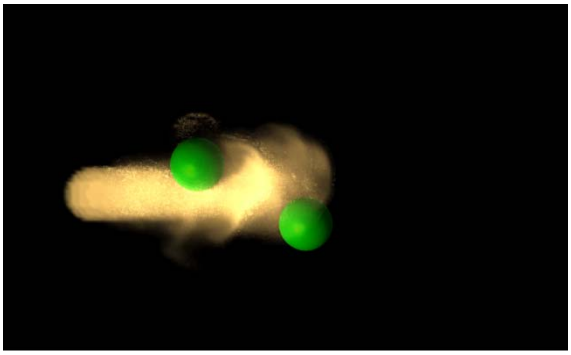
Frame =30



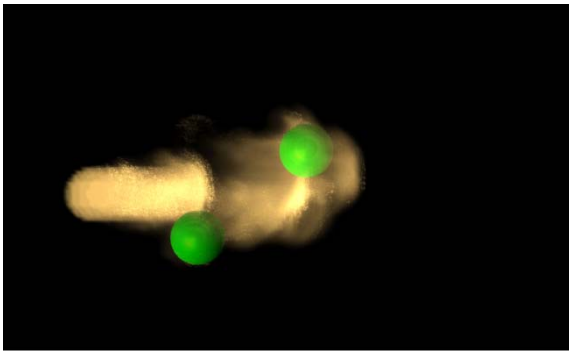
Frame =40



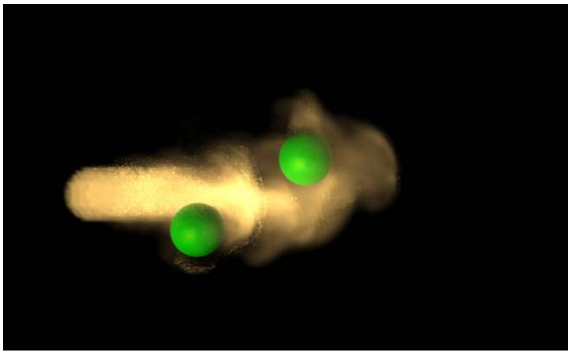
Frame =50



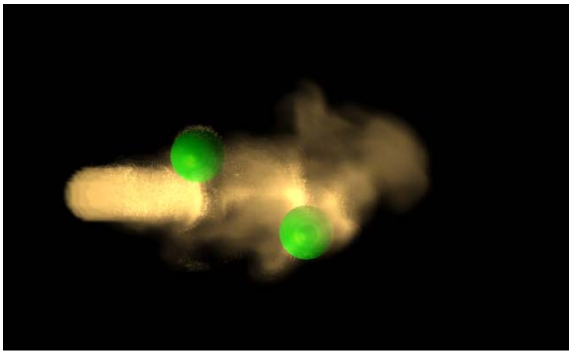
Frame =60



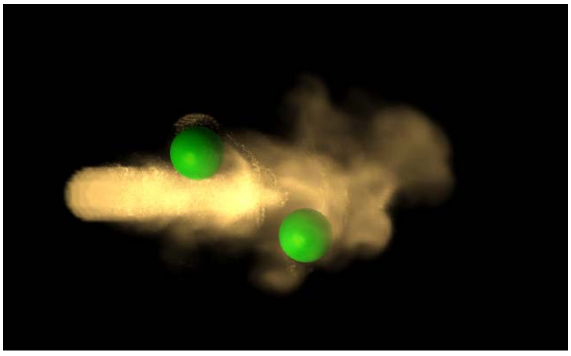
Frame =70



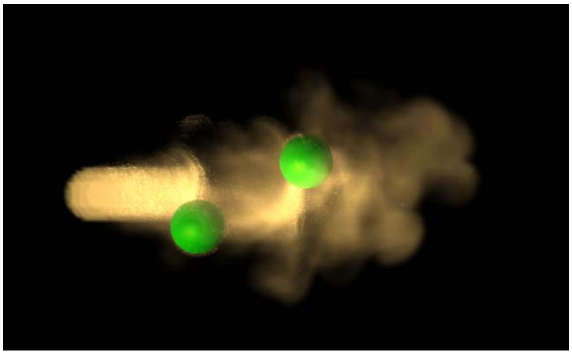
Frame =80



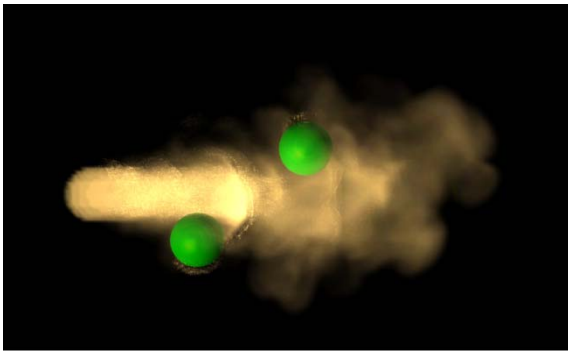
Frame =90



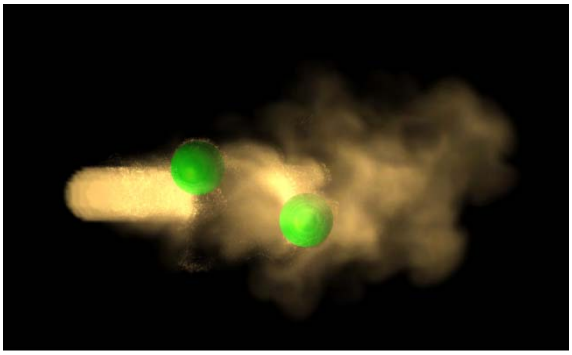
Frame =100



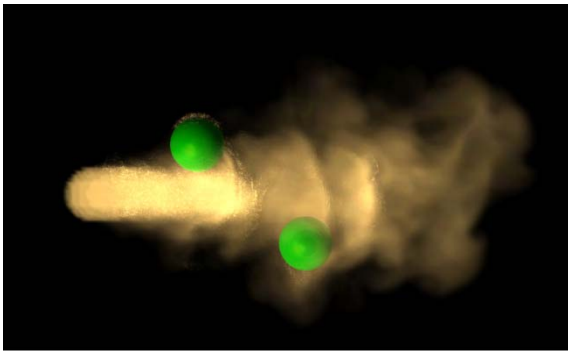
Frame =110



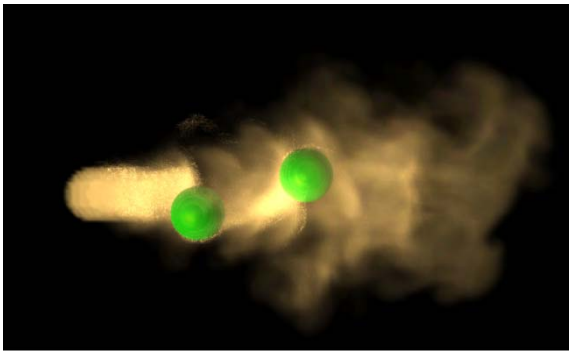
Frame =120



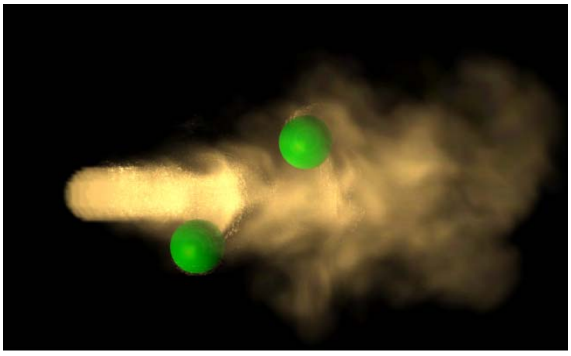
Frame =130



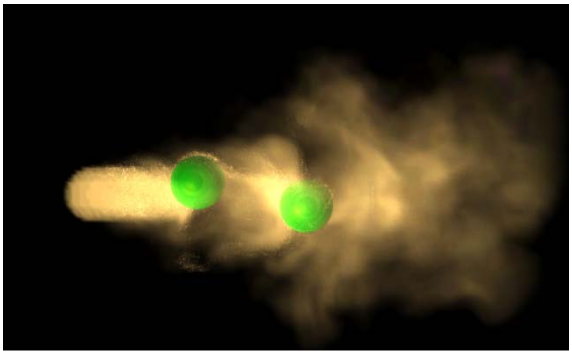
Frame =140



Frame =150

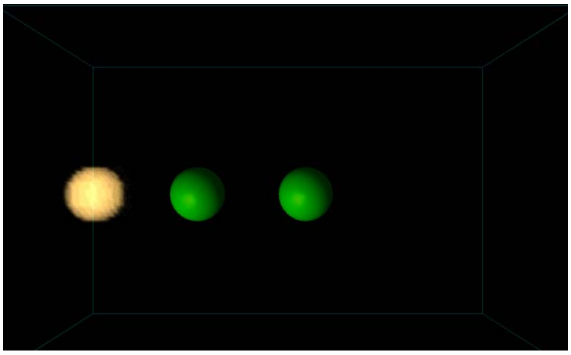


Frame =160

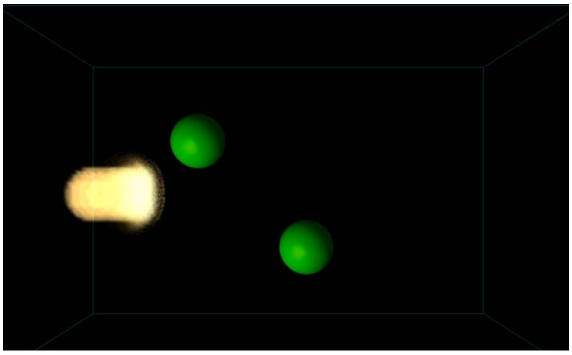


Frame =170

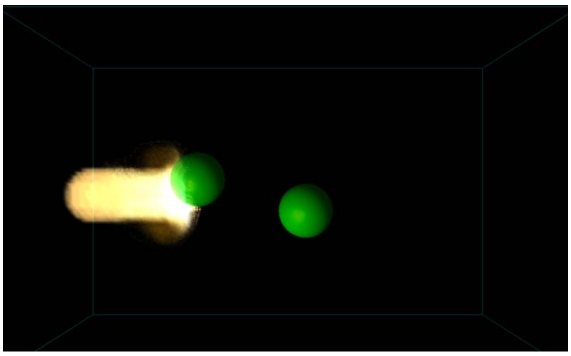
***Flamethrower with two oscillating spheres***



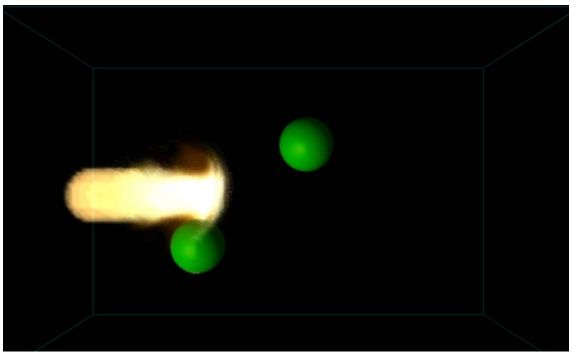
Frame =0



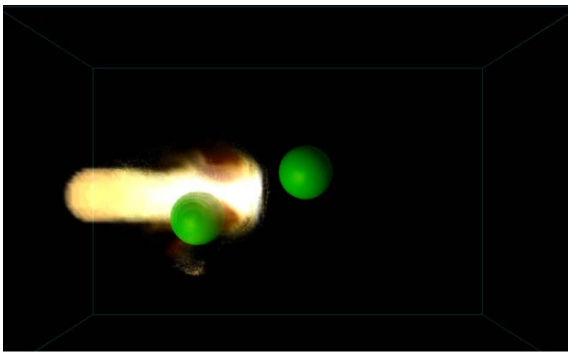
Frame =10



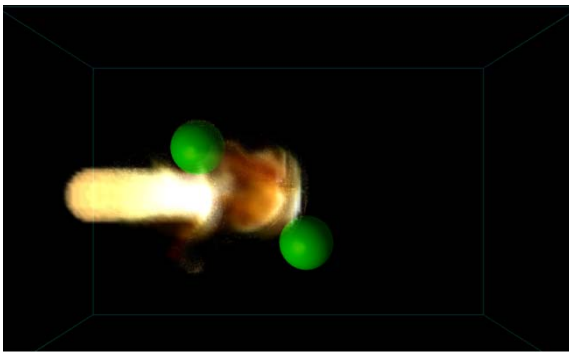
Frame =20



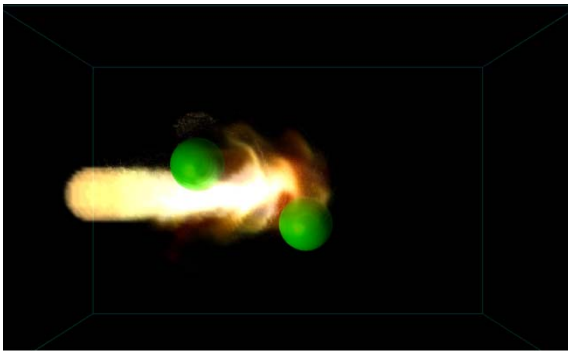
Frame =30



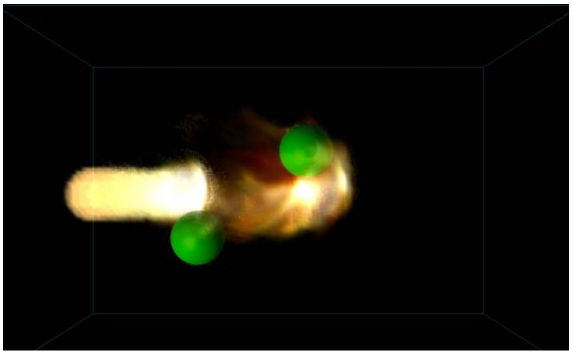
Frame =40



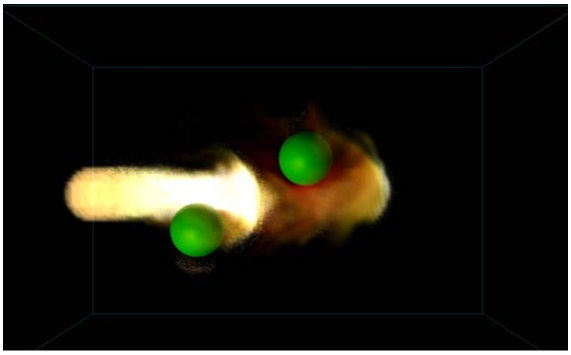
Frame =50



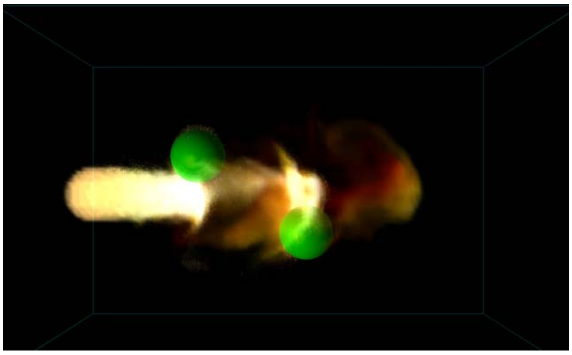
Frame =60



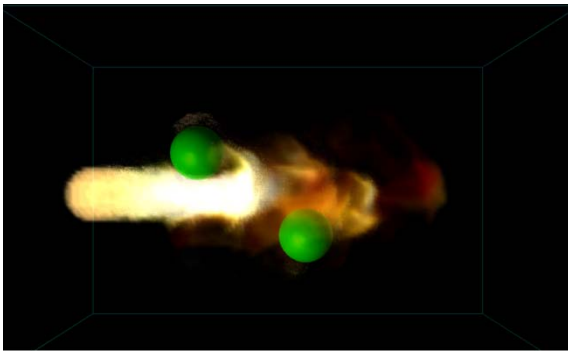
Frame =70



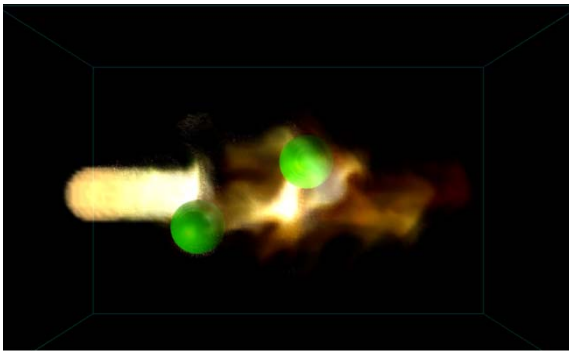
Frame =80



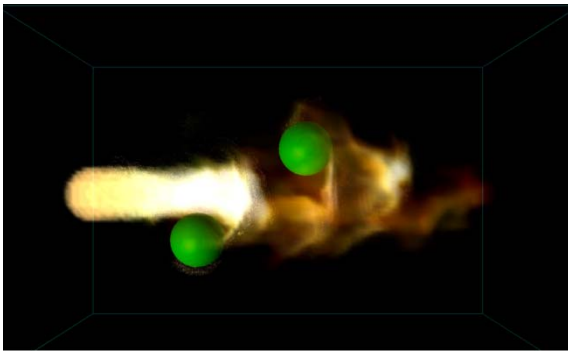
Frame =90



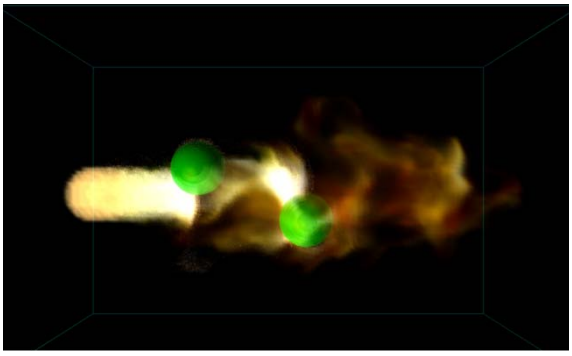
Frame =100



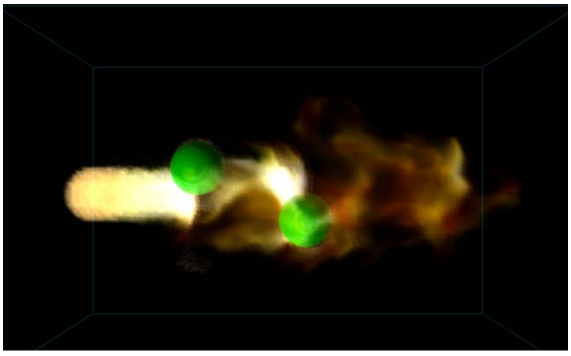
Frame =110



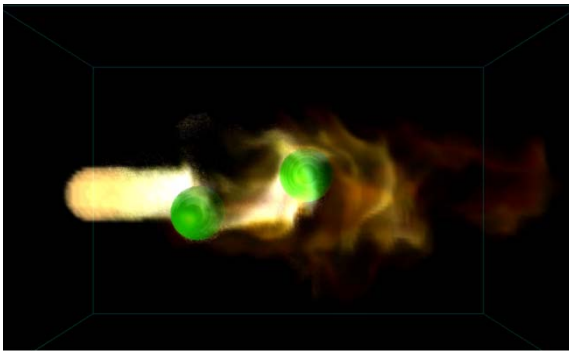
Frame =120



Frame =130

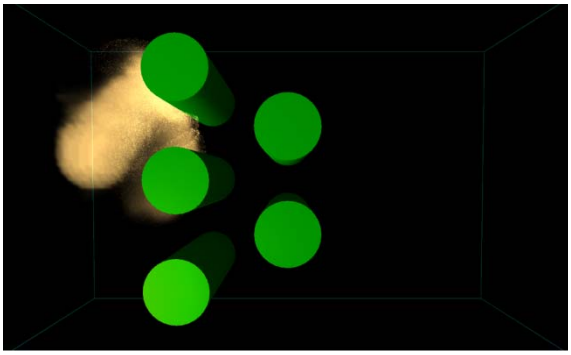


Frame =140

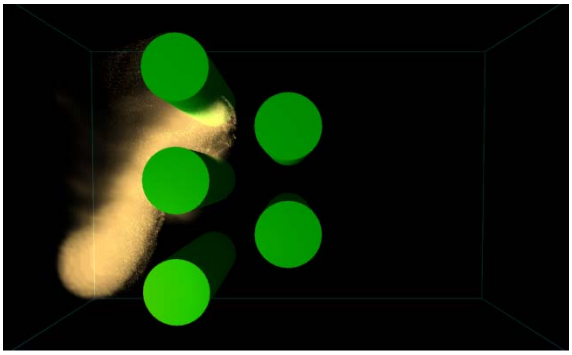


Frame =150

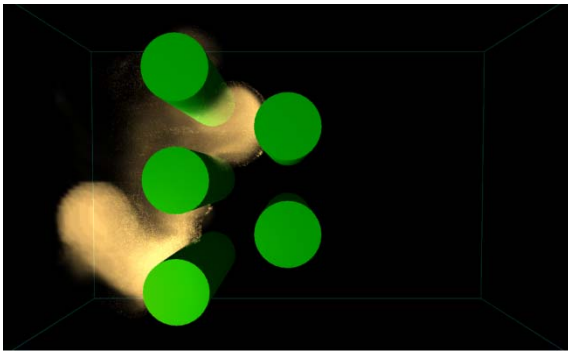
***Moving flows through a set of rods.***



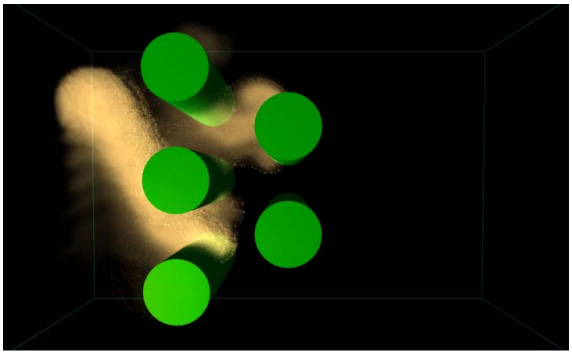
Frame =20



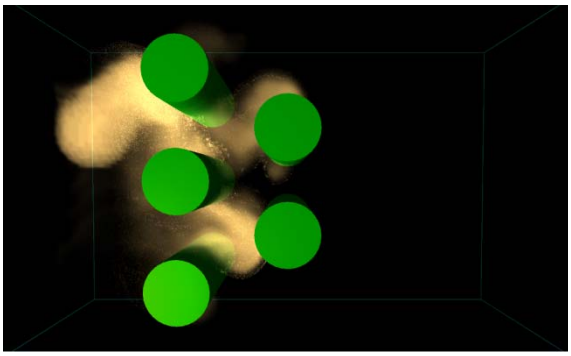
Frame =30



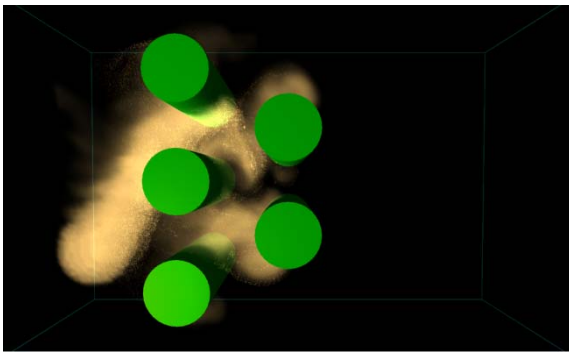
Frame =40



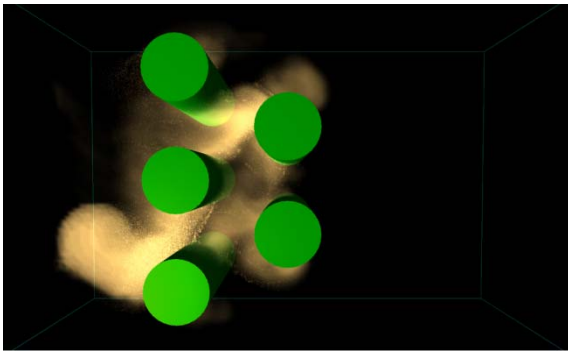
Frame =50



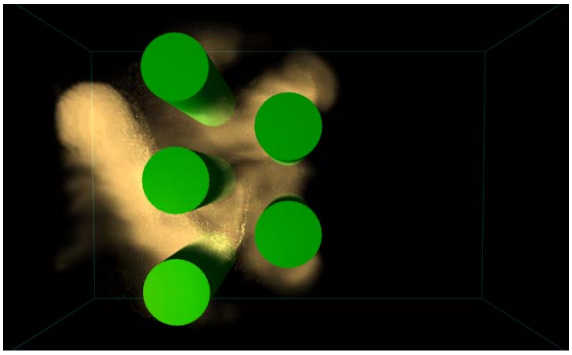
Frame =60



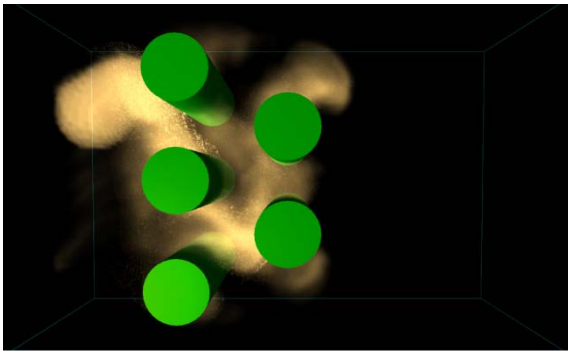
Frame =70



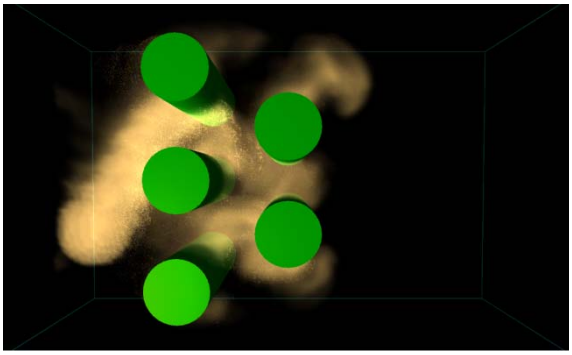
Frame =80



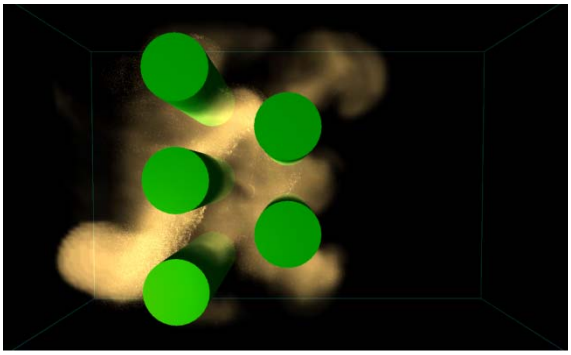
Frame =90



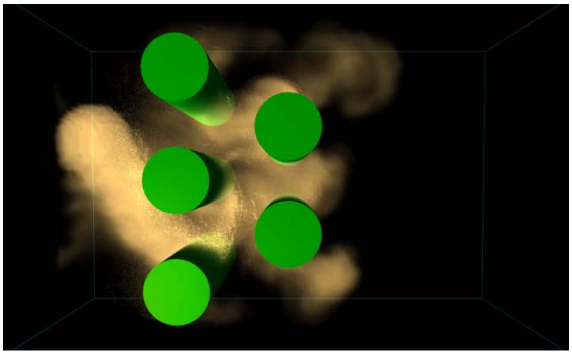
Frame =100



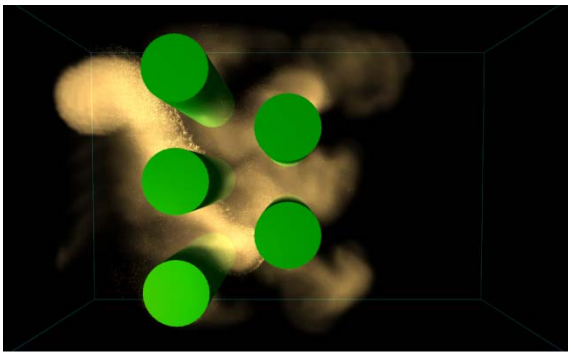
Frame =110



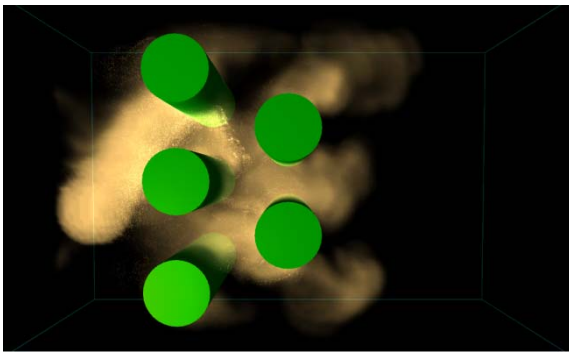
Frame =120



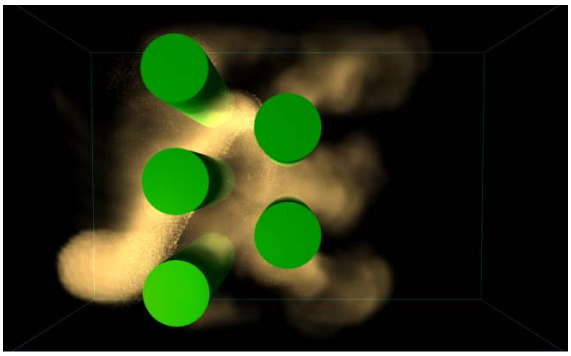
Frame =130



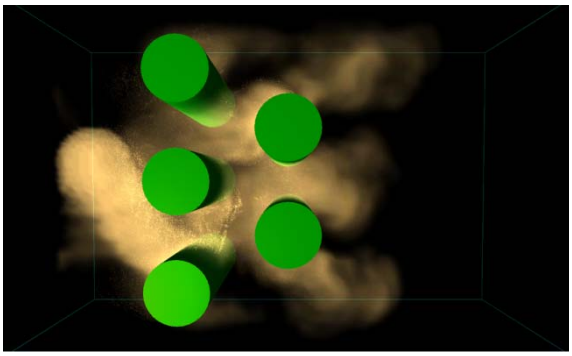
Frame =140



Frame =150

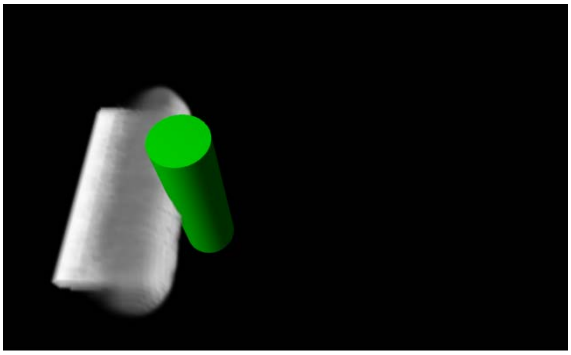


Frame =160

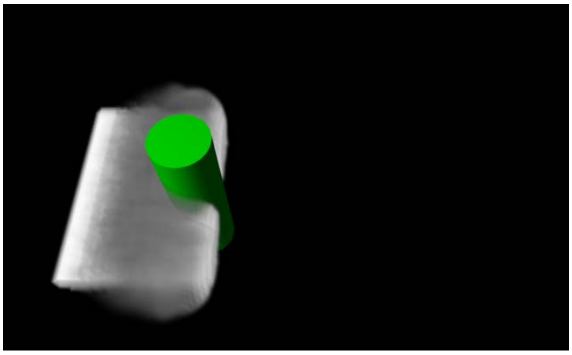


Frame =170

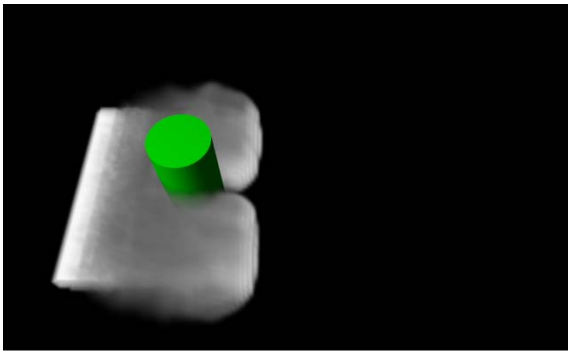
***A turbulence sheet of cloud intercepted by a pillar.***



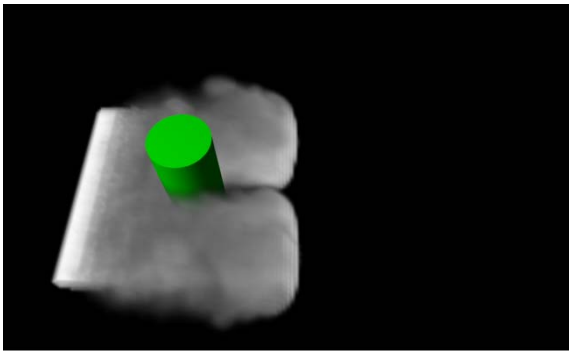
Frame =20



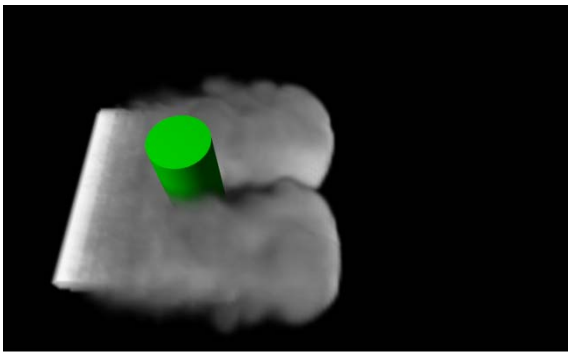
Frame =30



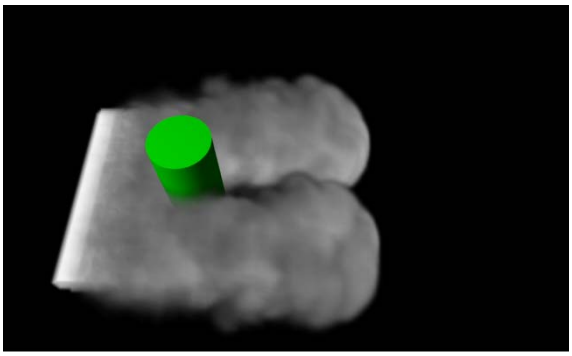
Frame =40



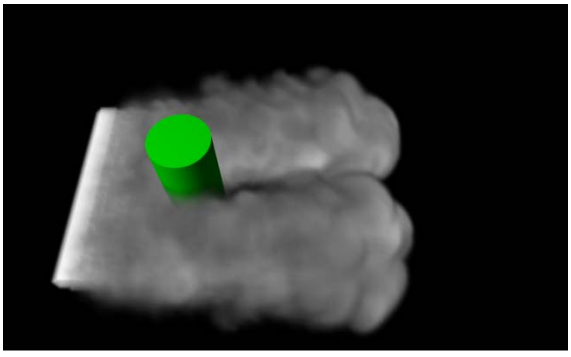
Frame =50



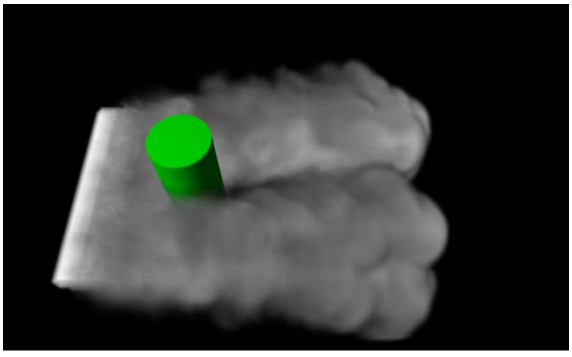
Frame =60



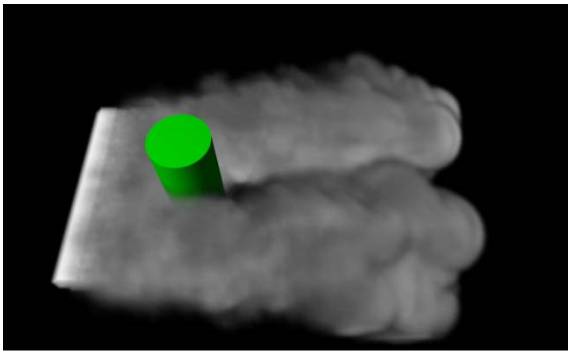
Frame =70



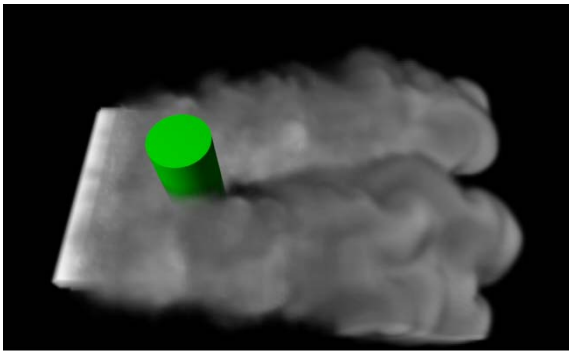
Frame =80



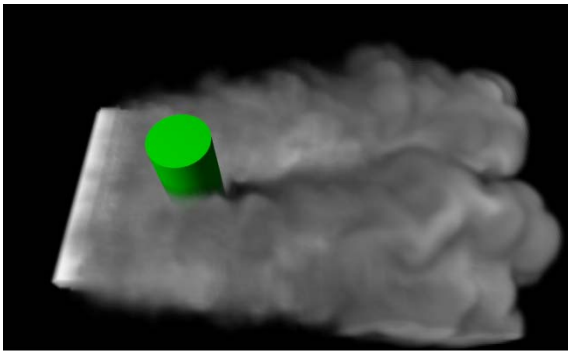
Frame =90



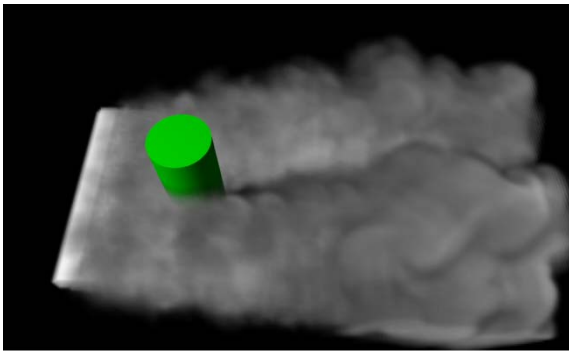
Frame =100



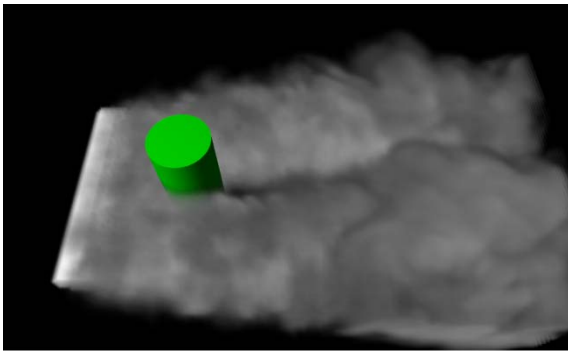
Frame =110



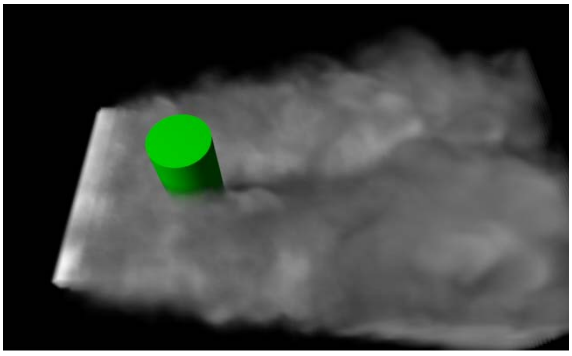
Frame =120



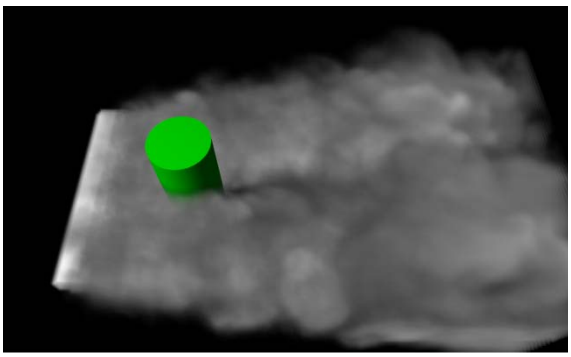
Frame =130



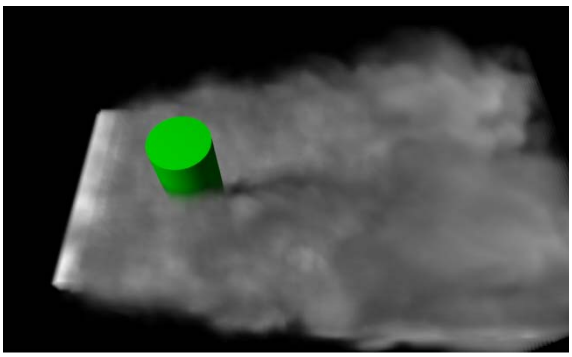
Frame =140



Frame =150



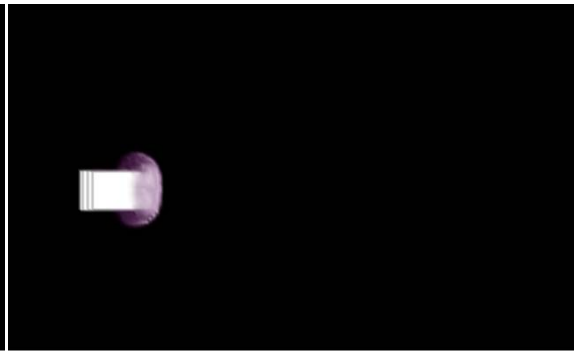
Frame =160



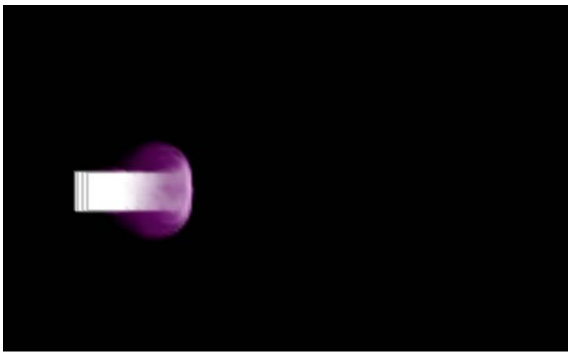
Frame =170

***Violet smoke flow rightward direction.***

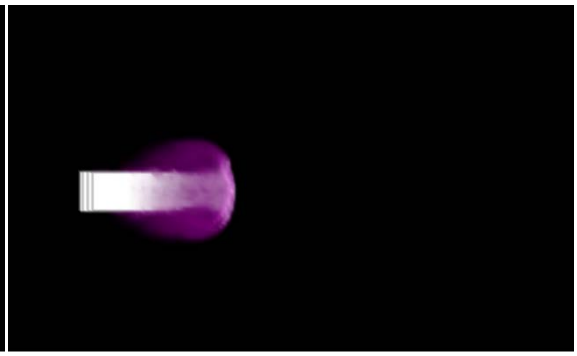
Frame =0



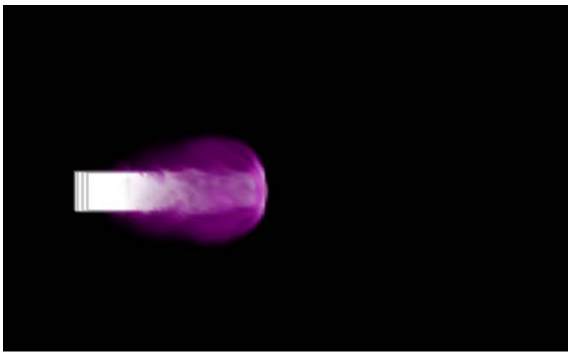
Frame =10



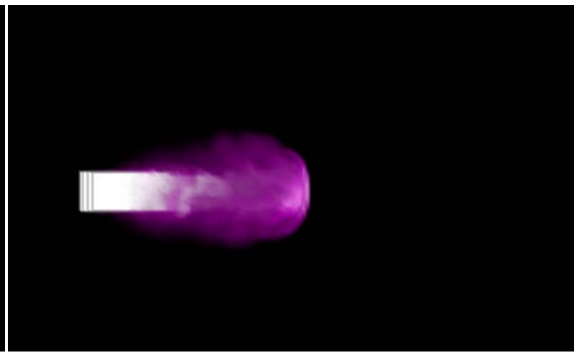
Frame =20



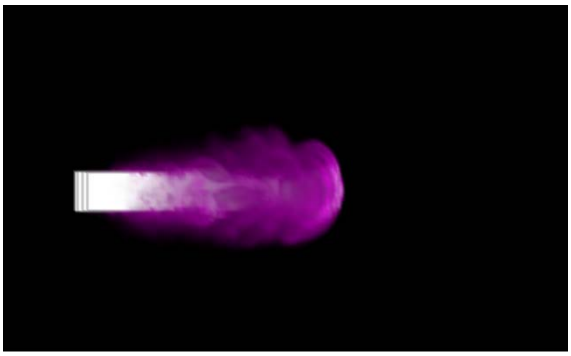
Frame =30



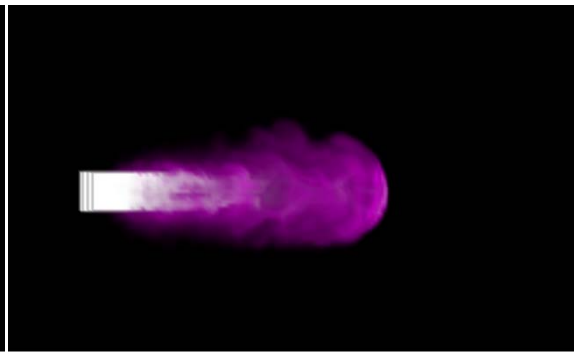
Frame =40



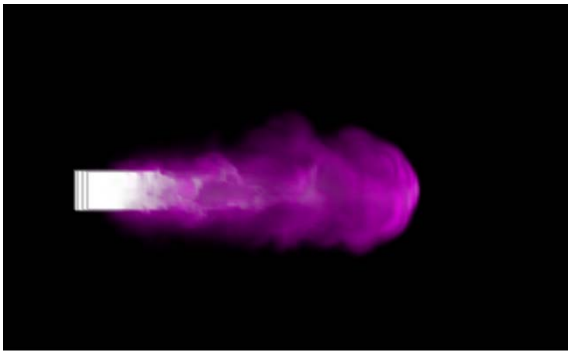
Frame =50



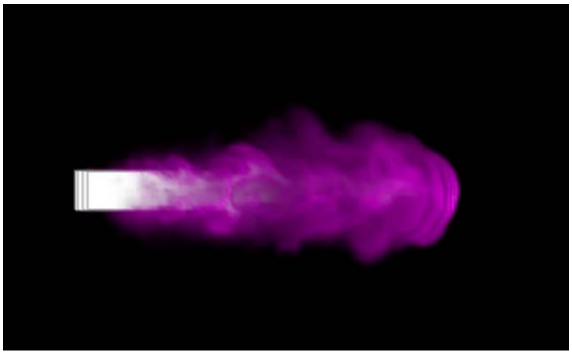
Frame =60



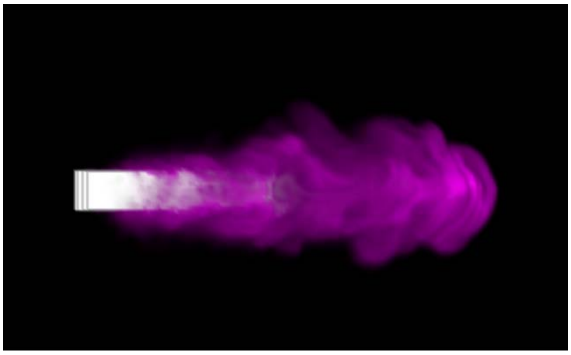
Frame =70



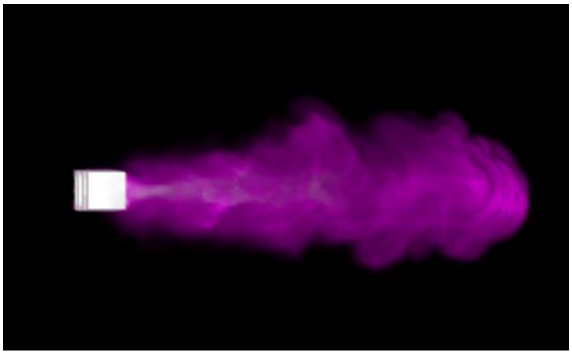
Frame =80



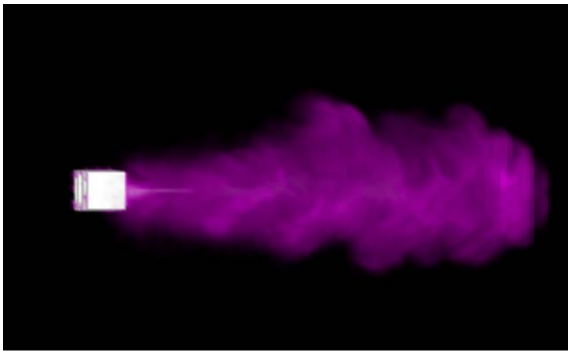
Frame =90



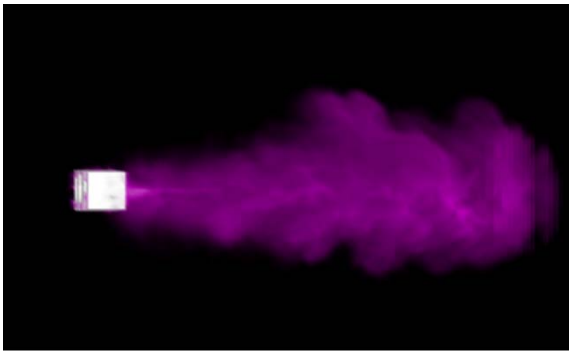
Frame =100



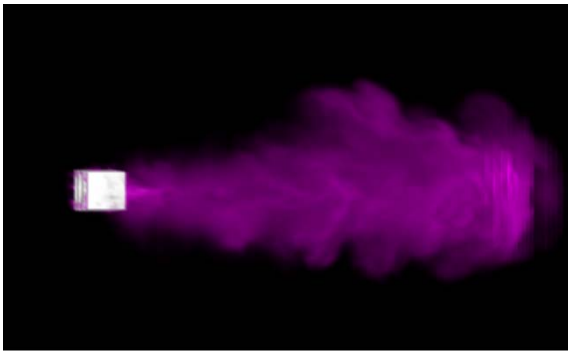
Frame =110



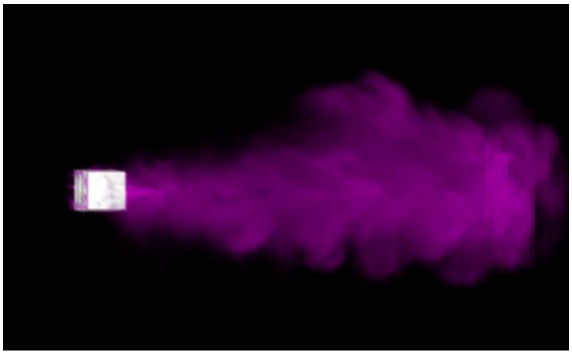
Frame =120



Frame =130

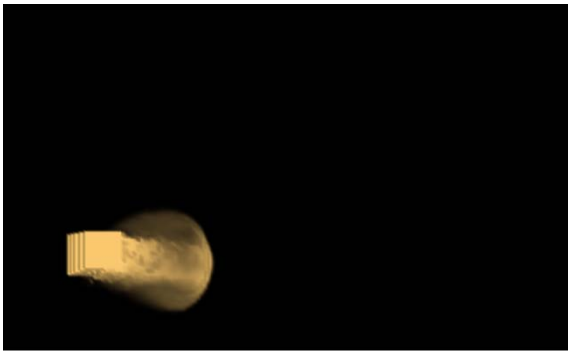


Frame =140

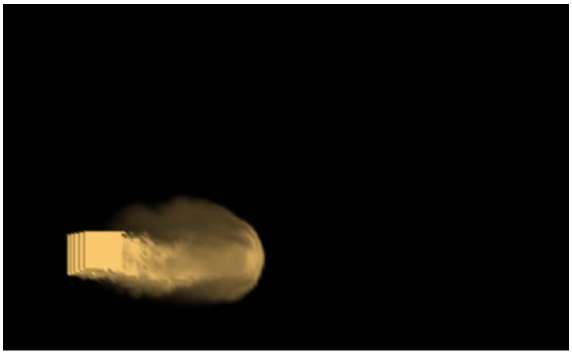


Frame =150

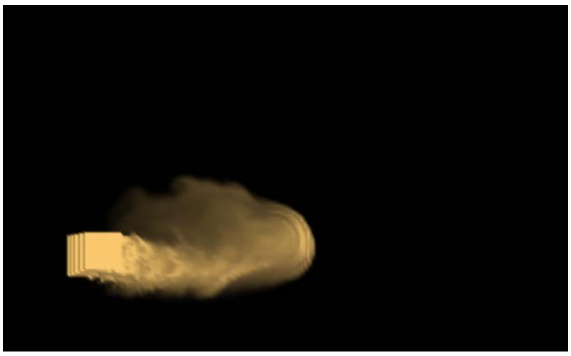
***Smoke flow rightward with slightly upward force.***



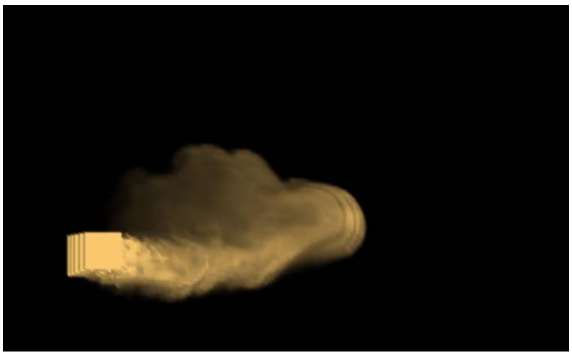
Frame =20



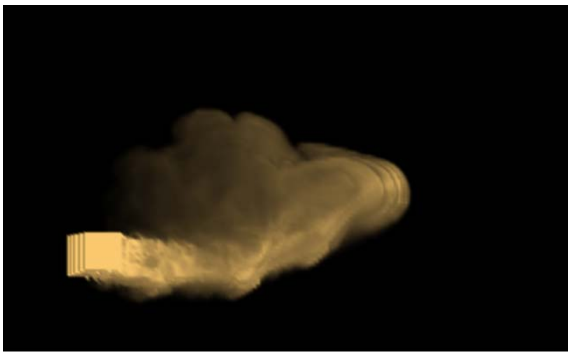
Frame =30



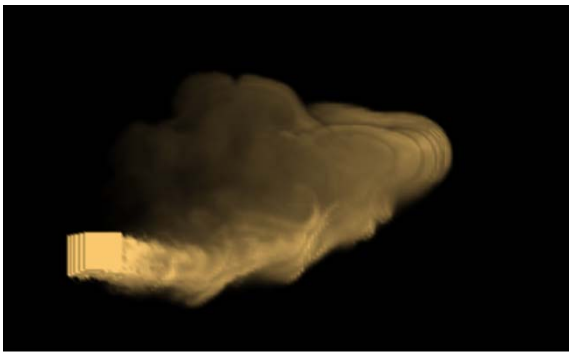
Frame =40



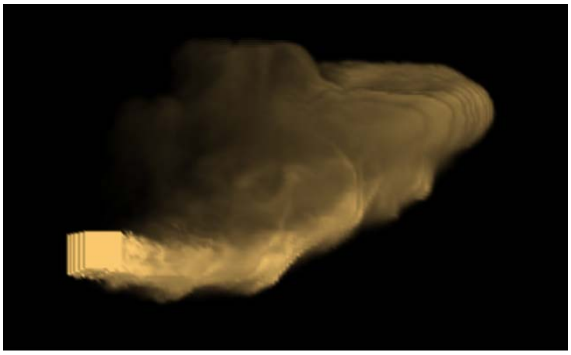
Frame =50



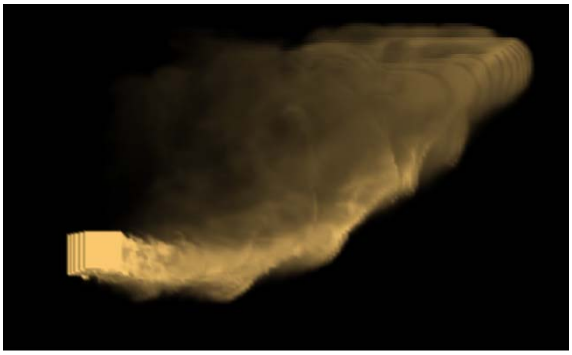
Frame =60



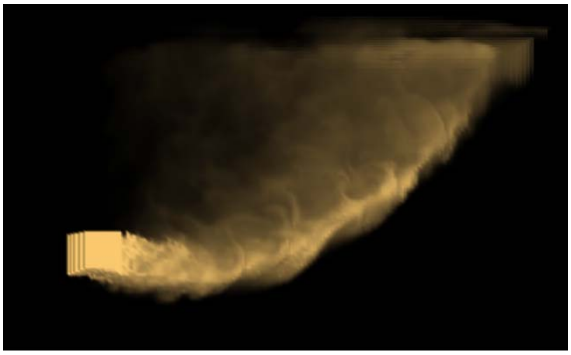
Frame =70



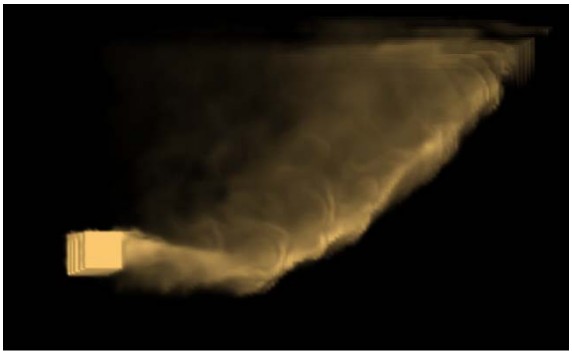
Frame =80



Frame =90



Frame =100



Frame =110

Frame =120

Frame =130

Frame =140

Frame =150

Frame =160

Frame =170

## Biography

Rinchai Bunlutangtum was born in Bangkok, Thailand, on August 1, 1988. He began his schooling at the Pasawee Elementrary School, and then continued his education at Bangkok Christian College, where his major's studies was physics and mathematics. In 2005, the year he gained his diploma, he entered Chulalongkorn University in Bangkok to be trained as an engineer. After four years of studied, he received his bachelor's degree and a scholarship for continuing master's degree in Computer Engineering, Chulalongkorn University.