ART-DIRECTABLE CLOUD ANIMATION

by

Yiyun Wang

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer Graphics Technology

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Tim McGraw, Chair

Department of Computer Graphics Technology

Dr. Bedrich Benes

Department of Computer Graphics Technology

Dr. Yingjie Chen

Department of Computer Graphics Technology

Approved by:

Dr. Nicoletta Adamo-Villani

ACKNOWLEDGMENTS

I wish to gratefully acknowledge my thesis committee for their insightful comments and guidance and my family and friends for their support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

fBm	fractional Brownian motion
CUDA	Compute Unified Device Architecture
OpenGL	Open Graphics Library
VAO	Vertex Array Object
VBO	Vertex Buffer Object
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPS	Frame Per Second

ABSTRACT

Volumetric cloud generation and rendering algorithms are well-developed to meet the need for a realistic sky performance in animation or games. However, it is challenging to create a stylized or designed animation for volumetric clouds using physics-based generation and simulation methods in real-time.

The problem raised by the research is the current volumetric cloud animation controlling methods are not art-directable. Making a piece of volumetric cloud move in a specific way can be difficult when using only a physics-based simulation method. The purpose of the study is to implement an animating method for volumetric clouds and with art-directable controllers. Using this method, a designer can easily control the cloud's motion in a reliable way. The program will achieve interactive performance using parallel processing with CUDA. Users will be able to animate the cloud by input a few vectors inside the cloud volume.

After reviewing the literature related to the real-time simulation method of clouds, texture advection algorithms, fluid simulation, and other processes to achieve the results, the thesis offers a feasible design of the algorithm and experiments to test the hypotheses. The study uses noise textures and fractional Brownian motion (fBm) to generate volumetric clouds and render the clouds by the ray marching technique. The program will render user input vectors and a three-dimension interpolation vector field with OpenGL. By adding or changing input vectors, the user will gain a divergence minimization interpolation field. The cloud volume could be animated by the texture advection technique based on the interpolation vector field in real-time. By inputting several vectors, the user could plausibly animate the volume cloud in an art-directable way.

1. INTRODUCTION

Many video games need a natural phenomena simulation to provide users a more realistic scene and a better immersive experience. Cloud simulation can be used in video games to improve sky scenes or weather systems. Though there are ways to simulate the cloud without considering it as volume, it is more difficult to offer a correct scattering effect inside the cloud. A common way to simulate the cloud is to treat it as volume and perform volume rendering. Most volumetric cloud generation processes will use noise textures to imitate the unpredictable shape of real clouds. Volumetric cloud rendering systems can be calculated fast enough on modern computers to provide real-time rendering. The cloud will look massive under volume rendering, and the viewport can move inside the cloud just like flying through real clouds on an airplane.

1.1 Problem

Video game companies have made some improvements in the cloud generation and rendering process in these years. For example, the dynamic atmosphere system in *Red Dead Redemption 2* Bauer [1] and the real-time cloudscapes in *Horizon: Zero Dawn* Schneider [2] are both chasing for realistic. The generation and rendering process from Bauer and Schneider are well developed, and the clouds rendering results are realistic enough. However, those methods can only work for natural weather systems or skyboxes, which means they can not generate clouds in a specific shape or animate the cloud by design. The animation industry or some video games production need more stylized and controllable clouds. Physics elements will have different shapes and move differently in those animation or games but still feel real. For example, in the short film Kevin and Peter [3], the clouds gather as the shape of giants and are acting like humans. Figure 1.1 shows the character in this film.

The clouds will partially blow when the giants move their bodies, making the cloud giants look reliable. Animations and rendering for films are generally calculated offline, but the previsualization of visual effects usually uses real-time rendering. If animators can test and generate models or motions for those elements in real-time, it will be more convenient. Moreover, the idea of a combination of realistic elements performance and controllable an-

Figure 1.1. Cloud giant in *Partly Cloudy*

imation is also a potential technique in real-time areas, such as video games. But current cloud generation and rendering methods used for games do not have enough art-directable controllers to simulate clouds' animation. To create real-time animations for clouds, people will use particle systems, fluid simulation, or flow maps. But those methods are either only applicable for single spots, not easy to control, or not flexible. Using particles can not display complex flowings of clouds, such as the convective phenomenon. Fluid simulation is the most precise method but mostly controlled by physics parameters, such as temperature and humidity. Those parameters are not directly related to motions. Flow maps are generated ahead by fluid simulation and stored in textures. They could provide complex and looping flow motions, but users can not modify the flow map in real-time. Users can not get real-time responses from a simulation base on flow maps. In a word, those current solutions are not both look real and art-directable.

In general, the problem addressed by this project is that current real-time volumetric cloud animation controlling methods are not art-directable.

1.2 Purpose

The purpose of the study is to implement an animating method for volumetric clouds and with art-directable controllers, meaning that a designer can easily control the cloud's motion in a reliable way.

1.3 Significance

Although physics-based simulation of clouds can provide realistic results, achieve a specific cloud shape or movement is not easy only based on simulation. "Animators need to adjust many non-intuitive parameters manually by a trial-and-error process [4]." Stam [5] implemented methods to solve Navier-Stokes equations using both Lagrangian and implicit methods, which propose a stable algorithm to solve the full Navier-Stokes equation for the first time. Stam used his method to render volumetric clouds as well, but his method can not shape the cloud in specific ways or animate them. Parameters of Stam's method and other physics-based methods are mostly about viscosity, diffusion rate, and dissipation rate. Changing those parameters makes it hard to generate a certain shape of clouds or control the movement. Figure 1.2 shows some physics-based volumetric clouds generated by Stam. Hong and Kim [6] generate a shape control method that can fill the volume clouds into a certain shape. The filling process is still physics-based, which means solving the Navier-Stokes equations. Figure 1.3 shows one of their results. It looks good, but their method took 21 minutes to compute and 1 minute per frame to render. Besides controlling the motion or creating art-directable controllers for animators, Hong *et al.*'s method aims to set the start and end shape of the cloud and let fluid simulation fill the blank. But, changing between specific shapes does not have universal usages in cloud animation. A more direct solution of controlled the cloud's animation without fluid simulation will save more time. So, developing an algorithm that helps animators easily edit the cloud's motion in real-time is significant.

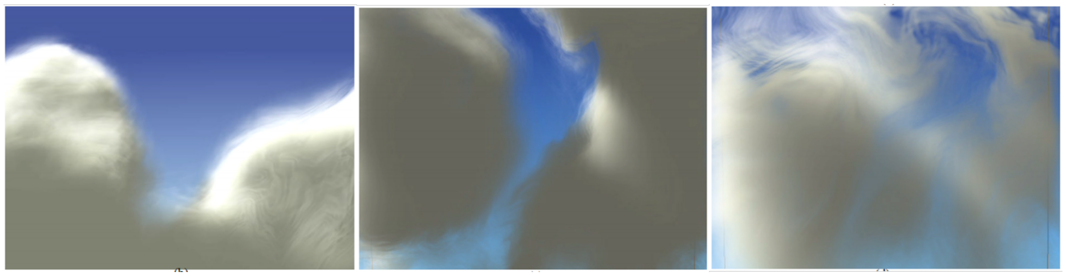


Figure 1.2. Physics-based simulation clouds from Stam [5]'s thesis

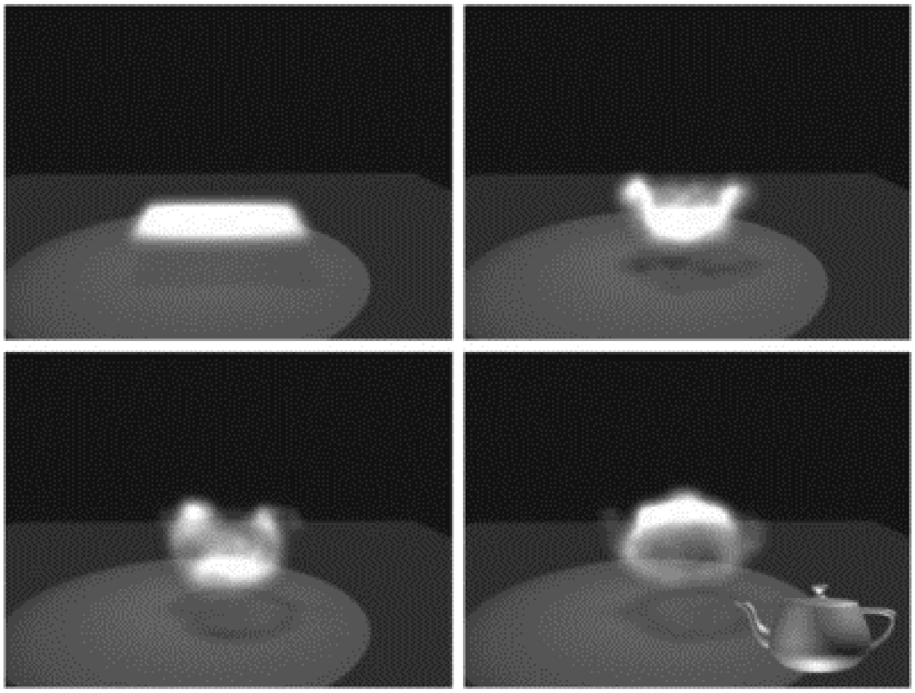


Figure 1.3. Physics-based simulation clouds from Hong and Kim [6]'s thesis

1.4 Research Question

Following questions need to be answered to develop the deliverable:

How to generate volumetric clouds?

How to make art-directable modifications to the cloud?

How to render the result to the screen?

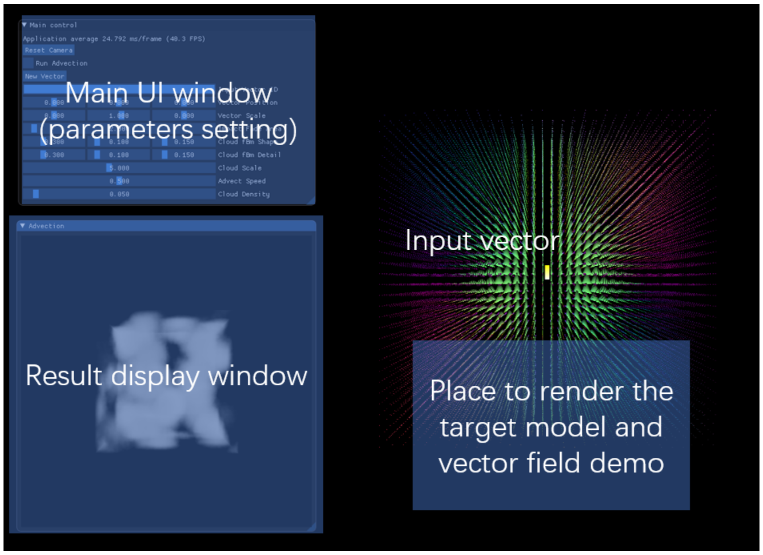
How to identify and analyze the result of the clouds?

1.5 Hypothesis

Vector field interpolation with divergence minimization combined with advection will be able to produce a plausible cloud animation while giving artist control over the final result.

1.6 Project Deliverable

The study's deliverable will be an executable CUDA program. A volumetric cloud and a vector field will render on the same screen. The cloud animation is based on the vector field, and the vector field is calculated from user inputs. Users can input multiple vectors from the main UI window and adjust the cloud's parameters. The program will calculate a divergence-free interpolation vector field from these vectors to simulate the cloud's movement without running a fluid simulation. Figure 1.4 shows the user interface design of this study.



Application average 24.792 ms/frame (40.3 FPS)

0.000 0.000 0.000 Vector Position

0.000 1.000 0.000 Vector Scale

0.300 0.100 0.150 Cloud fBm Shape

0.300 0.100 0.150 Cloud fBm Detail

5.000 Cloud Scale

0.500 Advect Speed

0.050 Cloud Density

Figure 1.4. An design of the user interface of this study

1.7 Assumptions

The assumptions for this study include:

The researcher will assume that each test's personal computer can support the real-time rendering requirement and render the game in equal visual qualities.

The researcher will assume the personal computer used in tests has Nvidia GPU and the correct CUDA version to run the executable files.

1.8 Limitations

The limitations for this study include:

For the algorithm developed in this study, the clouds' detail level will be limited by the textures' scale and fBm parameters.

The calculating speed will be limited by the CPU and GPU of the personal computer.

The quality of the rendering result will be limited by the GPU and monitor of the personal computer.

The value of the interpolation result is not guaranteed to be totally divergence-free on some specific inputs.

Some inputs will cause aliasing animation due to the interpolation results.

The density of the volumetric cloud is not conservative during the animation.

1.9 Delimitations

The delimitations for this study include:

No rendering or lighting model improvement of the cloud will be achieved in this research. The scope of the research is the modeling and animating of volumetric clouds. So, the researcher will use the current lighting method implemented by Lague [7], with different parameters.

Only volumetric clouds will be implemented in this research, which means the cloud is 3D and rendered by the ray marching process.

1.10 Definitions

In the broader context of thesis writing, we define the following terms:

Ray marching A technique used for visualizing three-dimensional data by casting rays from the camera and accumulating color for each ray [8]. In this research, ray marching is used for volumetric cloud rendering.

Texture advection Advection is the process by which a fluid's velocity transports itself and other quantities in the fluid [9]. " Texture advection is a technique to solve advection by trace each point of the texture backward in time when sampling [5]. In this research, texture advection is used for volumetric clouds animating.

Vector interpolation a process that does scatter data interpolation [10] in a vector field.

Perlin noise a type of gradient noise developed by Ken Perlin in 1999.

Worely noise a procedural noise developed by Steven Worley in 1996.

fBm a method that adds up the value from different channels multiple with different weights.

CUDA a development environment provided by NVIDIA for parallel computing, which can solve many complex computational problems more efficiently on NVIDIA GPUs than on a CPU. [11]

Host refers to CPU and its memory when mentioned in CUDA. [12]

Device refers to the GPU and its memory when mentioned in CUDA. [12]

ImGui an open-source graphical user interface for C++ used in this study.

1.11 Summary

The problem raised by this research is that a pure physics-based simulation of volumetric clouds can not provide a flexible controller to designers and animators in real-time. The purpose of this study is to implement an real-time art-directable method for users to interactively animate volumetric clouds.

2. REVIEW OF LITERATURE

The literature review will include methods of generating and rendering volumetric clouds and methods of achieving art-directable modeling and animation of clouds and other fluids.

2.1 Art-directable Methods For Fluids Animating and Modeling

2.1.1 Animating methods for water

From these two interview video [13], [14] between Brian Tong, senior editor, and Kyle Odermatt, visual effect supervisor from Disney, Kyle explain how they achieve the water effect in Moana. Moana is a movie "around water all the time" [13]. Besides making lots of beautiful ocean scenes, those animators even show the ocean as a character [14]. The ocean in Moana could grab stuff and people, nod, even give a high five (Figure 2.1).

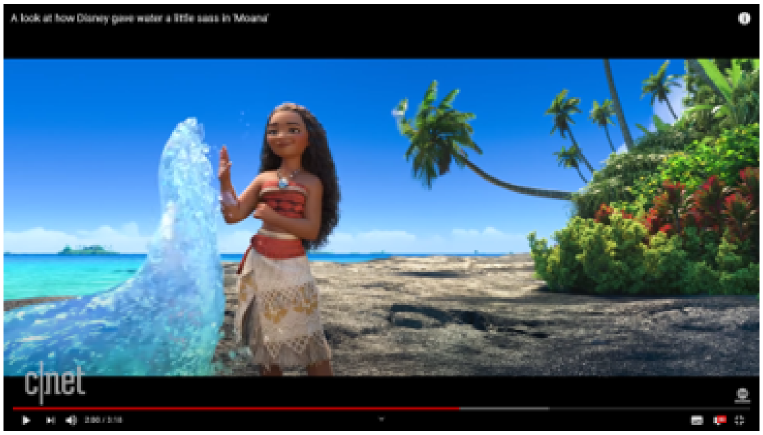


Figure 2.1. The ocean gives Moana a high five

It looks like a giant soft crystal creature and has a realism splash and millions of water drops bouncing around to show it is made of water. Figure 2.2 shows how animators combine the ocean's designed motion with those splash water drops and render the final scene with the help of simulation.

First, animators use rigged models, those two gray meshes in the top picture, to represent the ocean's movement as a character. After they decide the central movement, they run water simulation over those meshes and gain the position of each water drops around. Finally, they

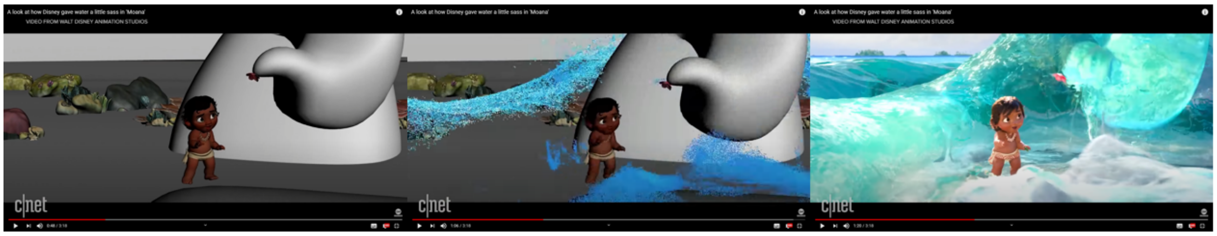

A look at how Disney gave water a little sass in 'Moana'
VIDEO FROM WALT DISNEY ANIMATION STUDIOS
A look at how Disney gave water a little sass in 'Moana'
A look at how Disney gave water a little sass in 'Moana'
VIDEO FROM WALT DISNEY ANIMATION STUDIOS
cnet
cnet
cnet

Figure 2.2. Processes of water rendering in Moana

render the final scene shown at the bottom with the help of multiple powerful computers. It shows both the crystal-like ocean water body, which holds the flower, and small water streams dripping down. Compare those pictures in Figure 2.2, the white foam at the bottom is generated from the blue particles.

2.1.2 Animating Methods For Cloud And Smoke

An interactive generation method had been published by Dobashi, Kusumoto, Nishita, *et al.* [15]. By setting new edges of the volume cloud, the cloud will be generated and grow to meet the edges. By setting a target contour line, they generate a 3D target shape. Their simulation is controlled to minimize the difference between the current cloud shape and the target shape. Dobashi, Iwasaki, Yue, *et al.* [4] improved their algorithm in 2008 and compared the rendering result with real photos. The method of Dobashi *et al.* could generate unnatural shapes of clouds. Figure 2.3 is an example of their result that they generated skull-shape clouds. The color and light parameters could be adjusted to create multiple rendering results realistically. Dobashi *et al.*'s method has a similar idea with Hong *et al.*, setting the cloud's target shape and letting the fluid simulation fill the process. Dobashi *et al.* use physical parameters to offer a basic force of the cloud. For example, they set a higher temperature at the bottom of the scene to let the cloud rise.

Treuille, McNamara, Popović, *et al.* [16] implemented a key frame control for smoke simulation. They let the user set keyframes for the smoke and formulate an objective function to descript the difference between the simulation and the keyframe. Then solve the force parameters which minimize that function. Their solution takes two to five hours to calculate each simulation in 50 × 50 pixels. Figure 2.5 shows one of their results. The keyframe in

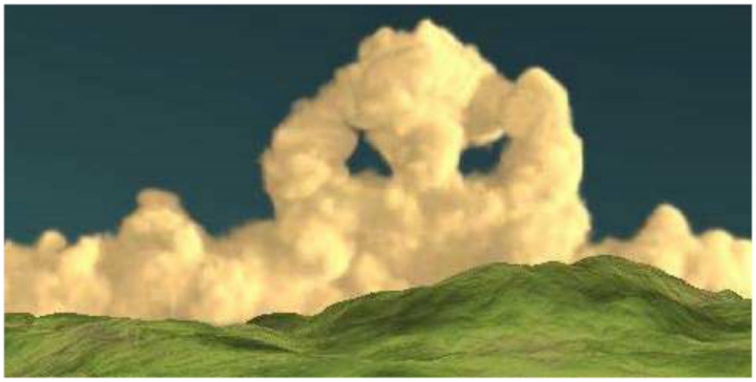


Figure 2.3. Dobashi *et al.* generated a skull-shape cloud

this simulation is the pattern of the text “smoke”. Treuille *et al.* uses velocity vector field as control parameters. McNamara, Treuille, Popović, *et al.* [17] use the adjoint method to control large 3D physics-based simulation of fluid. They use the Gaussian wind forces presented by Treuille *et al.* and add sources to their control parameters. McNamara *et al.*’s method produces very close matches to the keyframes and works both for water and smoke. However, their method takes hours, even days, to calculate. Figure 2.4 shows their result of a man running animation with water and smoke simulation. Fattal and Lischinski [18] implemented a target-driven method for smoke simulation. Their method concludes two terms: a driving force term to carry the smoke towards the target state and a gathering term to prevent over diffusion. Their method is much faster but can not provide a real-time performance in 3D. Fattal *et al.*’s result is shown in Figure 2.5.

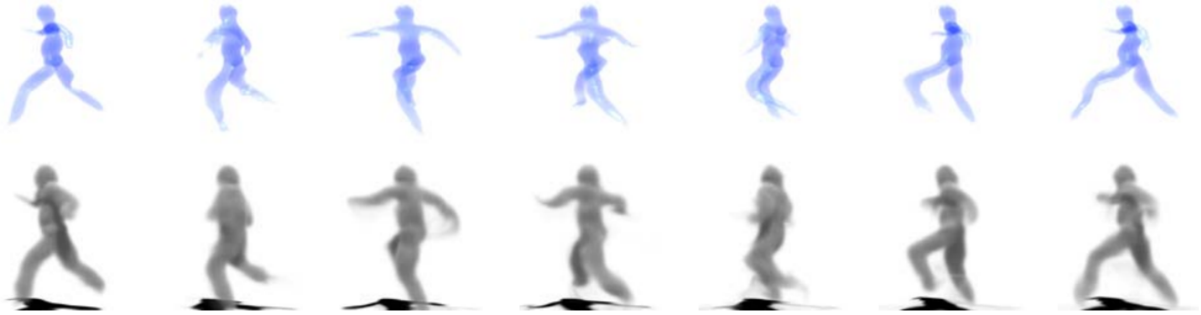


Figure 2.4. McNamara *et al.*’s fluid control simulation with adjoint method

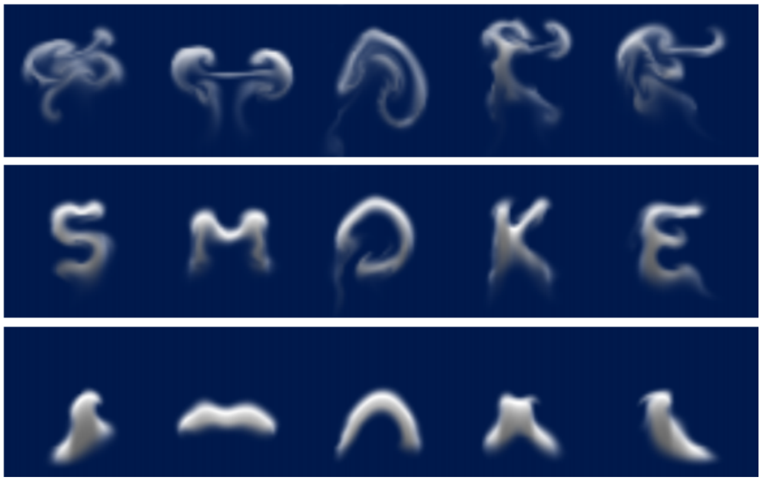


Figure 2.5. Treuille *et al.*'s key frame control simulation

2.1.3 Modeling Methods For Clouds

Bouthors and Neyret [19] proposed a method to build the shape of cumulus clouds by surface mesh and shaders. They filled the blobs with Perlin noise texture and used a view-dependent shader to create a fuzzy edge.

2.2 Cloud Rendering Methods

2.2.1 Volume Rendering

Ikits, Kniss, Lefohn, *et al.* [8] offers an overview of volume rendering techniques. The difference between volume models and regular mesh models is that volume models can show the inside details rather than the surface. Volume models assume that "light is emitted, absorbed, and scattered by a large number of particles in the volume [8]". Cloud is a volume because it is made of plenty of tiny water drops, and those water drops will absorb and scatter light passing the cloud. Instead of the regular shading method of mesh objects, a technique called ray marching will be used to see the inside of a volume. Ray marching provides an accumulated result for each ray passing the volume, but the calculation process will vary on different scattering and lighting rules. Harris and Lastra [20] provided a physics-based rendering method for static clouds, including light scattering illumination, multiple forward scattering, and eye scattering. The cloud rendering result of their algorithm includes shadows inside the cloud to make it realistic. However, the clouds in Harris *et al.* are a group of 2D quads rather than a 3D volume. But the idea of Harris *et al.* is helpful for their future works [21], which based on 3D clouds.

2.2.2 Cloud Generation Methods

For the generation process, Schpok, Simons, Ebert, *et al.* [22] used only Perlin noise to model the clouds. Changing the parameters when adding the noise, Schpok, Simons, Ebert, *et al.* [22] generated multiple types of clouds, such as cumulus and cirrus. Schneider [2] used 3D noise textures to generated volumetric clouds and rendered them by volume rendering. Those textures fill with Worley noise in four channels, from low frequency to high frequency. Moreover, a 2D texture to control the distribution of clouds in the sky. Hillaire [23] shared the implementation in their game Engine, which also refers to the process of Schneider [2] on the 2D cloud layer control and rendering process. Schneider's method is working as a skybox, so as Hillaire, the ray marching process is calculated between two hemispheres. By accumulating the density and calculating the light scattering at each point with Beer-Powder laws, each ray's result will be rendered to the screen. The implementation of Lague [7] in the Unity game engine renders the volumetric clouds inside a box by casting rays from the camera and intersecting with a box in the scene [8]. Then accumulate the cloud density and color from the front faces to the back faces of the box.

2.3 Physics-based Methods For Fluid Animating

2.3.1 Fluid Simulation

Stam [5] implemented methods to solve the Navier-Stokes equations using both Lagrangian and implicit methods, which propose a stable algorithm to solve the full Navier-Stokes equation for the first time. Harris, Baxter, Scheuermann, *et al.* [21] also implement a physics-based animate process based on the 3D clouds. However, their algorithm did not support interaction between the users and the clouds. The simulation is based on the Navier-Stokes equation, but they only show 2D results of clouds affected by the fluid. They used a flat 3D texture technique that 2D texture contains all slices of the volume. Therefore, all slices could be updated in a single rendering pass. Their work shows a possible approach for texture-based fluid simulation in cloud simulation. The physical model from Harris *et al.* is based on what Stam did in 1999, the stable fluid simulation. Fedkiw, Stam, and Jensen [24] use a fluid simulation equation for smoke simulation. Netzel and Weiskopf [25] implement a texture-based visualization method of advection by blending noise texture and advected texture.

2.3.2 Flowmap

Alex [26] introduced the usage of flow maps in their games. They generated a 2D flow map based on their level geometry in Houdini by fluid simulation in advance. The flow map provides unique 2D vectors for each point on the water surface. They use the flow map to distort the water surface's normal map, which provides real-time water flowing animation. Simon [27] implements software to generate flow maps. However, the flow map generator from Simon or Houdini can not finish generation in real-time, which means users can not interact with the flow map by real-time inputs.

2.4 Divergence-free Vector Field Interpolation

A method for generating user-controllable vector fields that represent plausible wind velocity fields is to impose physically constraints on the vector field. Arbitrary vector fields are not plausible if they do not represent incompressible flow. One way of imposing this constraint is to generate vector fields which are divergence-free. Yassine and McGraw [28] and Yassine and McGraw [29] developed methods for interpolating tensor fields of arbitrary order while imposing constraints on the divergence and curl of the resulting field. However, these methods require that the interpolated data lie on a regular grid. To provide more general artistic freedom to specify vectors at any location, a scattered-data interpolation technique will be more useful. McNally [30] and Mitrano and Platte [31] develop matrix-valued basis functions for generating scattered-data interpolation methods for divergence-free vector fields in 2D and 3D.

2.5 Texture Advection

Nelson, Roger, and Dean [32] discovered that advecting the cloud texture coordinates helps visualize the impact of wind flow. Their study used a 3D texture to render the cloud over a sphere and advected the texture by wind flow to simulate climate on earth. Yu, Bruneton, Holzschuch, *et al.* [33] presents a Lagrangian method for advecting textures. Their method takes a 2D velocity field and a Perlin noise texture as inputs and produces the noise texture following the velocity field. Their method aims to avoid over distorts of the input textures during advection. Figure 2.6 shows their method works for a climate simulation on earth, which likes Nelson *et al.*'s result but in higher resolution. Neyret [34] implemented an animated fluid method by combing layers of advected parameterizations to increase the details of the animation.

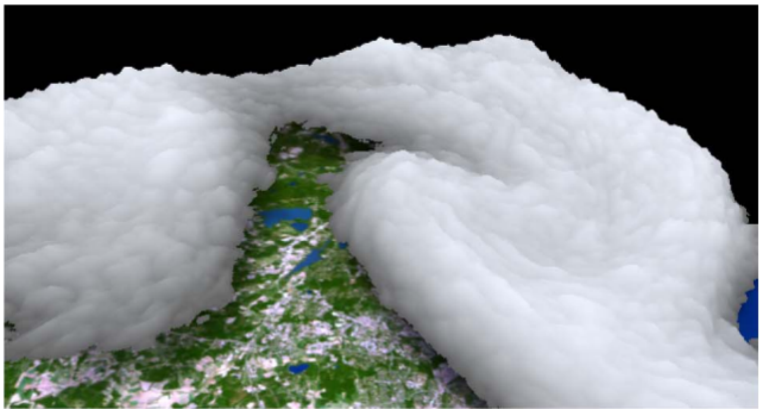


Figure 2.6. Yu *et al.*'s method on 3D clouds

3. METHODOLOGY

The methodology chapter presents the framework of the deliverable projects. It includes the main workflow, code organization, and implementation details of each step.

3.1 Development Tools

The project is using OpenGL and CUDA. The version of OpenGL is 4.6, and the CUDA version is 11.1. A computer needs to install a corresponding version of OpenGL and CUDA and a Navida GPU card to run this project.

3.2 Research Approach

This research is developmental. An art-directable animating program of volumetric clouds will be created based on the findings relevant to the proposed methodologies in the literature review chapter. The algorithmic implementation will use parallel computing, which means doing lots of independent computing at the same time with the help of a GPU card.

The program expects the user to modify the volumetric cloud and set an appropriate vector field before calculating the advection. The result of the cloud and vector field will update and display per frame. Users can add input vectors into the vector field or set fBm parameters. The program will calculate a divergence-free interpolation vector field and adjust the clouds' appearance based on the inputs. The result of the vector field and clouds will be stored in a 3D buffer. After the user chose to start the advection calculation, the program will stop updating the cloud's appearance and calculate advection based on current clouds. While calculating advection, the user can still modify the input vectors but can not adjust the fBm parameters.

3.3 Project Framework

3.3.1 Working Flow

Figure 3.1 gives a flowchart of the primary process in the program. The program starts from the preparation and initiation of multiple buffers. The preparation process includes malloc buffers on the host and device, including buffer storing the vector field and cloud density, 3D noise textures, input vectors, interpolation matrix, and render texture. After buffer malloc, the program will calculate the initial interpolation matrix based on the initial input vector and copy the host's data to the device. While running the main loop, the user can use two switches to control the calculation and rendering processes, which lead to three different CUDA kernels. Those kernels match three different program states: Shaping the volumetric clouds, running the advection, pausing the advection.

In shaping the volumetric clouds state, the program will run the cloud generation (fBm) kernel. The user can modify the vector field by adding or adjusting input vectors. The program will solve the divergence-free interpolation matrix and update the buffer on the device. The user can shape the cloud in this kernel by modifying the fBm parameters too. The cloud's shape will be stored as a 3D density buffer on the device. The kernel's inputs are vector field and cloud density data from the previous frame, user input vectors, interpolation matrix, and fBm parameters. When updating the vector field, the kernel will call a device function to calculate the interpolation value at the current position based on the coordinates and inputs. The kernel will update the vector field and cloud density data.

In running the advection state, the program will run the advect kernel. The user can still modify the vector field as the cloud shaping state and see real-time feedback. The user can not shape the cloud anymore. The cloud density will advect based on the velocity values from the current vector field. New density will be stored on the device. The kernel's inputs are vector field and cloud density data from the previous frame, user input vectors, and interpolation matrix. The kernel will update the vector field and cloud density data.

In pausing the advection state, the program will run the pause kernel. The user can only modify the vector field like previous states and see real-time feedback. The user can not shape the cloud, and the cloud will not move. This state is designed for observing the volume. The user can pause the advect calculation and rotate or scale the volume to observe it under different views. The cloud density value will remain unchanged in the buffer. The kernel's inputs are as same as the advect kernel. The kernel will only update the vector field data.

Implementation details for these functions will be present in the following sections.

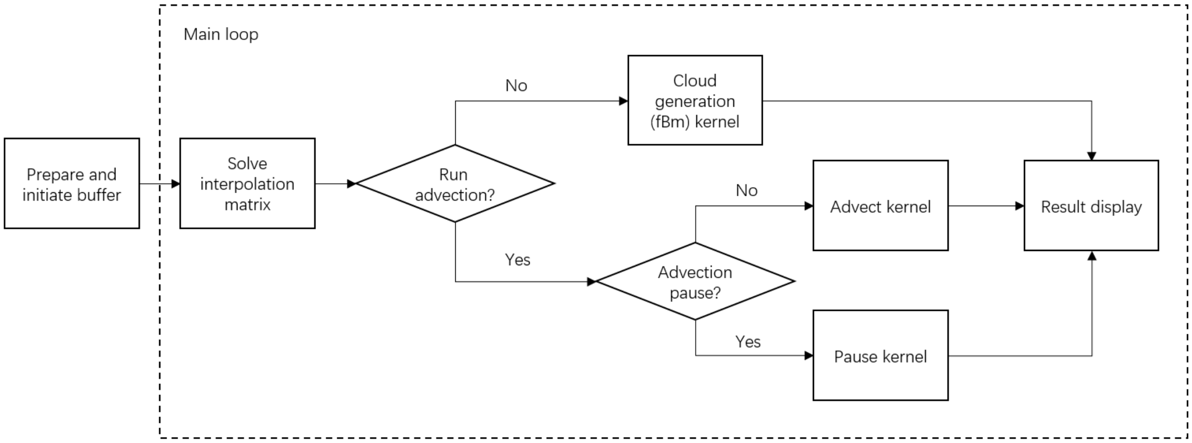


Figure 3.1. The main flowchart of the deliverable project in this research

3.3.2 Data Management

Table 3.1 lists and describe major data buffers used in this program, including the data types, location (host or device), default scale and buffers' function. The "Number" column is for easier description. Each buffer only implements once in the project. The data type includes vec3 and vec4 from the math lab, GLM.

Buffer name starts with "h" means the buffer is on the host, start with "d" means the buffer is on the device. A device buffer can only read and write by device or global functions. The program is passing only pointers of these buffers between functions.

Buffer 1 and 2 store the user inputs from the interface, including position, vector value, and weight. The weight is a float value that and that controls the influence area of the vector during interpolation. The program packs the vector value and weight as a vec4. Buffer 3 stores the interpolation matrix calculated by the host. The device functions need data in buffer 1, 2, and 3 to calculate the vector field interpolation. So, buffer 4, 5, and 6 copy the data from them and move to the device. Default scales of buffer 1 to 6 are all equal to 5. The program will malloc the maximum scale for those buffers even the initiate input number is one. The maximum scale number could be bigger, but this research aims to use fewer controllers to achieve the art-directable animating process.

Buffer 7 stores the data for user inputs display and bind to VBO, including vector value and color. The buffer scale is four times the input vector because each vector needs four vec3 to display: start position, start color, end position, and end color. This data is updated and used only on the host.

Buffer 7 stores the data for user inputs display and bind to VBO, including vector value and color. The buffer scale is four times the input vector because each vector needs four vec3 to display: start position, start color, end position, and end color. This data is updated and used only on the host. Buffer 8 stores the data for the vector field interpolation display. It has the same structure as buffer 7, including two positions and two colors. However, the buffer is linked to its VBO through CUDA. The program uses cudaGraphicsResource to bind this data with VBO. The data will be updated only on the device.

Buffer 11 and 12 are noise textures generated on the host during buffer preparation. These data will be set as CUDA texture for cloud generation on the device. They are using float3 and float4 rather than vec3 or vec4 to match the data type of CUDA texture.

Buffer 13 is a 2D texture buffer store the volume rendering result of the program. The program uses cudaGraphicsResource to bind this data with a pixel buffer object and use ImGui image to display.

3.4 Vector Field Interpolation

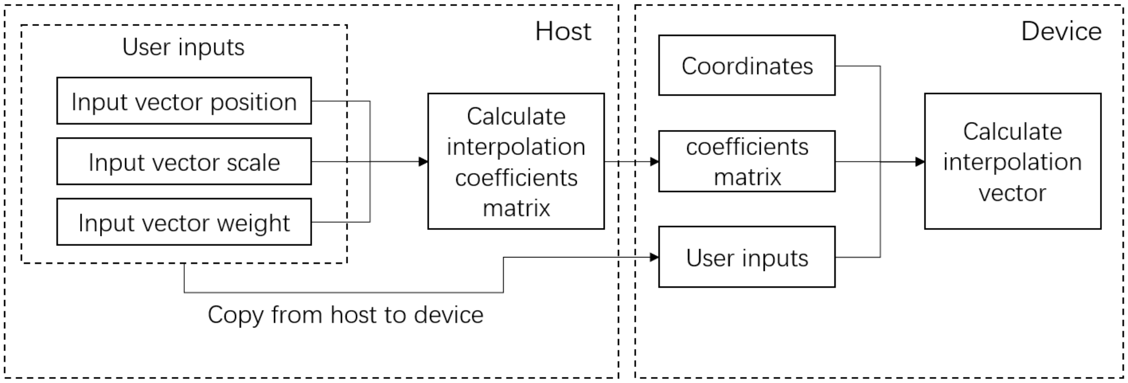


Figure 3.2. Interpolation process on the host and on the device

The program uses matrix-valued basis functions to generate divergence-free vector fields in 3D. The functions include calculating the interpolation coefficient matrix on the host and calculating the interpolation vector values for each point on the device. Figure 3.2 shows the process of interpolation on the host and on the device. When calculating the interpolation coefficient matrix, the program first fills all user input vector scale components into the column vector d. Suppose the number of inputs is N, the size of the column vector is $3 \times N$. Then the program fills a matrix Phi with the data of input vector position and weights. The algorithm will calculate a 3×3 matrix from the vector position difference and weights in the filling process and fills the nine components of the matrix into the Phi matrix. The Phi matrix's size is $3N \times 3N$. The pseudocode of the 3×3 matrix calculation algorithm is shown in Algorithm 1 and the filling algorithm is shown in Algorithm 2. The program uses a c++ linear algebra library, armadillo, to solve the inverse matrix.

Algorithm 1 Calculate 3×3 matrix

```
Require: vector p, weight $\epsilon$
    $r2 \leftarrow$ p dot p
    $d \leftarrow \mathrm{e}^{-\epsilon \times r2}$
    New $3 \times 3$ matrix phi
    $phi \leftarrow ((4 \times \epsilon - 4 \times \epsilon \times \epsilon \times r2) \times glm :: mat3(1) + 4 \times \epsilon \times \epsilon \times glm :: outerProduct(p, p)) \times d$
    return matrix phi
```

After copy the user inputs data and the interpolation coefficient matrix to the device, the program can use these data and the volume's current coordinates to calculate the exact interpolation vector at this point. Algorithm 3 shows the process on the device. During the process of this device algorithm, Algorithm 1 is still needed. So, the program implements it again on the device. The final output of this algorithm is the interpolated vector at a given position. Figure 3.3 shows an interpolation result with these algorithms in MATLAB. These four red vectors are the input vector, and these algorithms generate other blue vectors.

3.5 Cloud Generation and Rendering

The Cloud generation process includes two parts. One is generating 3D noise textures before the loop. The other is sampling from the noise to make an fBm cloud result.

Algorithm 2 Fill the interpolation matrix

```
Require: position x[N], scale v[N], weight $\epsilon$[N]
  for i = 0 to N do
    $d[\mathrm{i}] \leftarrow v[\mathrm{i}].x$
    $d[\mathrm{i}+N] \leftarrow v[\mathrm{i}].y$
    $d[\mathrm{i}+2\times N] \leftarrow v[\mathrm{i}].z$
  end for
  for i = 0 to N do
    for j = 0 to N do
      Call Algorithm1 with x[i] - x[j] and $\epsilon$[i] on the host
      Get 3 ×3 matrix p
      $Phi[\mathrm{i},\mathrm{j}] \leftarrow p[0][0]$
      $Phi[\mathrm{i}+N,\mathrm{j}] \leftarrow p[0][1]$
      $Phi[\mathrm{i}+2\times N,\mathrm{j}] \leftarrow p[0][2]$
      $Phi[\mathrm{i},\mathrm{j}+N] \leftarrow p[1][0]$
      $Phi[\mathrm{i}+N,\mathrm{j}+N] \leftarrow p[1][1]$
      $Phi[\mathrm{i}+2\times N,\mathrm{j}+N] \leftarrow p[1][2]$
      $Phi[\mathrm{i},\mathrm{j}+2\times N] \leftarrow p[2][0]$
      $Phi[\mathrm{i}+N,\mathrm{j}+2\times N] \leftarrow p[2][1]$
      $Phi[\mathrm{i}+2\times N,\mathrm{j}+2\times N] \leftarrow p[2][2]$
    end for
  end for
  New vector cvec
  New vector c
  $cvec \leftarrow Phi/d$
  for i = 0 to N do
    $c[\mathrm{i}].x \leftarrow cvec[\mathrm{i}]$
    $c[\mathrm{i}].y \leftarrow cvec[\mathrm{i}+N]$
    $c[\mathrm{i}].z \leftarrow cvec[\mathrm{i}+2\times N]$
  end for
  return vector c
```

Algorithm 3 Interpolation on the device

```
Require: pos, position x[N], scale v[N], weight $\epsilon$[N], coefficient matrix interp[N]
  for i = 0 to N do
    Call Algorithm1 with pos - x[N] and $\epsilon$[i] on the device
    Get 3 ×3 matrix p
    $v \leftarrow v + phi \times interp[\mathrm{i}]$
  end for
  clamp v between 0 to 1
  return vec3 v
```

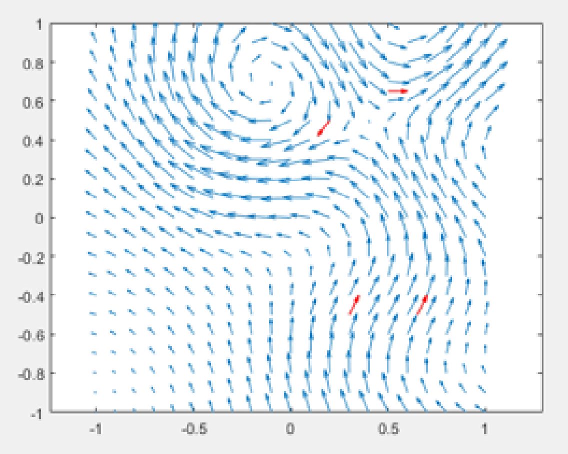
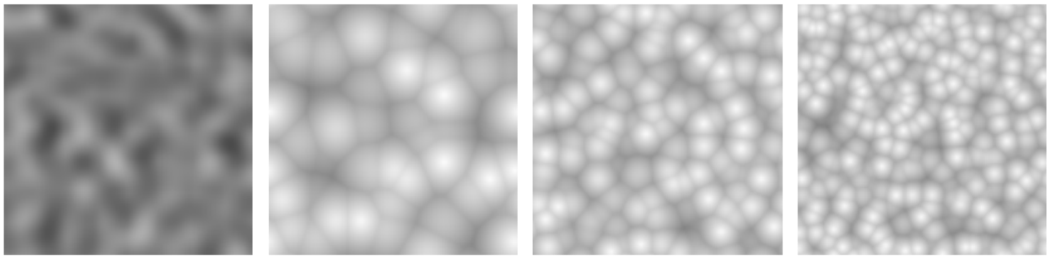


Figure 3.3. An interpolation result in MATLAB

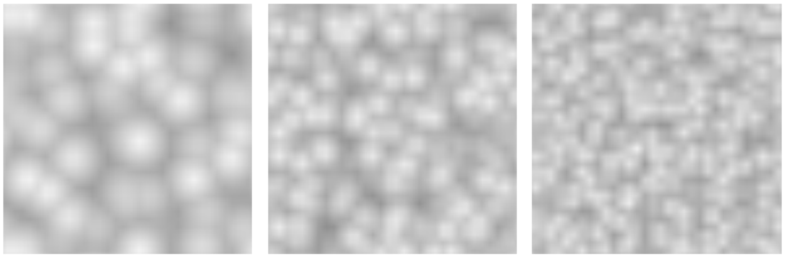
3.5.1 3D Noise Generation

The program generates two types of 3D textures: a shape texture and a detail texture. The shape texture uses RGBA color channels, and its resolution is 128 × 128 × 128. The basic shape of clouds is sampled from the shape texture by fBm. Figure 3.4 and 3.5 show the cutting surfaces of noise textures filled in each channel. The shape texture includes one Perlin noise texture and three Worley noise textures. The detail texture is using RGB color channels, and its scale is 32 × 32 × 32. However, Both textures need to be seamlessly repeating, so that the user can scale it without limit, and the texture will self-repeating with higher frequency. Both noise textures are copied to the device as CUDA textures. They are read-only from the device faster than stored in global memory. And they can easily linearly sampled by tex3D functions.



R: Perlin noise G: Worley noise (low) B: Worley noise (medium) G: Worley noise (high)

Figure 3.4. The cutting surface of each shape noise channel



R: Worley noise (low) G: Worley noise (medium) B: Worley noise (high)

Figure 3.5. The cutting surface of each detail noise channel

3.5.2 Cloud rendering with fBm

Figure 3.6 shows each shape noise channel's ray marching result and the result combined by fBm. If the ray intersects with the box containing the volumetric clouds, the marching will begin between the entry and exit points. For each step of the marching, the program will unify the current position to texture coordinates and sample the shape and detail textures' value. Those values will combine as a density of the current point. The program will then cast a ray from the current point against the light direction and do another marching to calculate the lighting. Figure 3.7 shows that those yellow points represent the marching for lighting calculation. The result of this marching process represents the cloud density between the light source and the starting point. Large density leads to smaller transmission, and the result of the current position (the green point) will be darker.

After lighting calculation, the program will check if the position reaches the exit point or the density is already opaque. If not, it will match a small step forward and accumulating the density and lighting results. If so, the marching will end, and the program will write the result color to the pixel buffer.

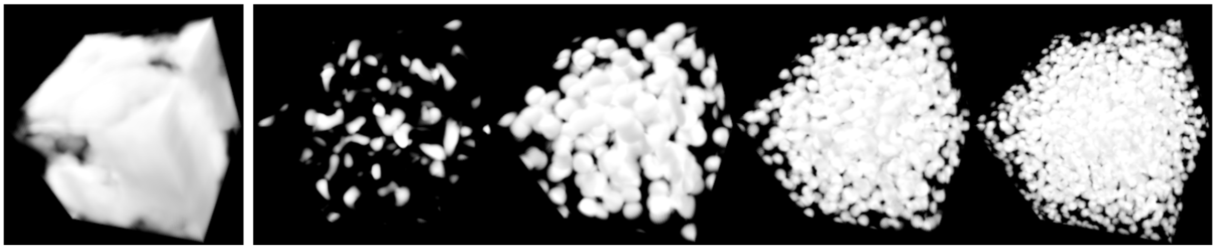


Figure 3.6. The fBm result and the ray marching result of each shape noise channel

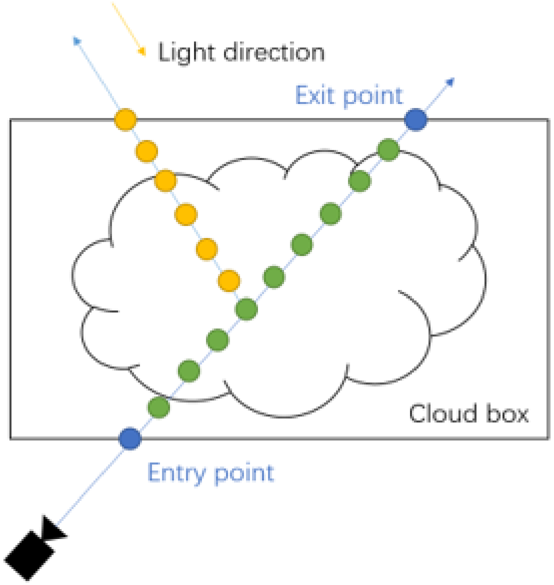


Figure 3.7. A schematic diagram of ray marching process

3.6 Volume Texture Advection

The program uses Ping-Pong buffers to save the cloud from the previous frame and the advection result. When calculating the advection, each point inside the cloud box will sample from the 3D vector field for velocity and cloud density. The program will trace against the velocity and reach the new position. A new density value will be sampled from this new position [34] and blended with the old one.

Figure 3.8 shows a 2D example of advection. When tracing back a vector, the example calculates a linear interpolation result from neighboring grids. However, in this research, the program uses the exact value from the current grid because the resolution of the 3D texture is large enough. The possibility of a 3D linear interpolation sampling method for the advection will be discussed later in future works section.

3.7 Vector Field Display

The program will use two groups of vectors to display the user input vectors and the interpolation vector field. Each group of vectors is bound to separate VAO and VBO. The

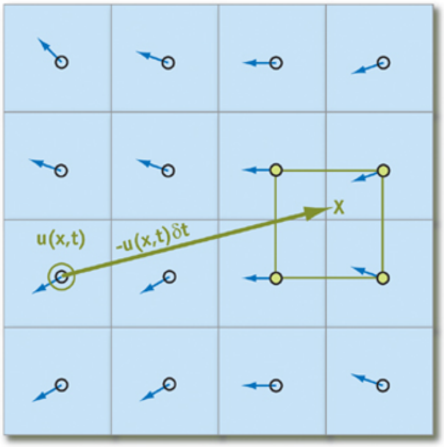


Figure 3.8. A 2D example of advection algorithm [9]

input vectors are stored and update on the host, but the interpolation vector field data are updated by the device. The program uses cudaGraphicsResource to bind the display data array with VBO. The vector field display function is called after calculation processes per frame and sample from the vector field data buffer. The sample number is related to a constant parameter: vector field view scale. The view scale should be obviously smaller than the data scale (256 × 256 × 256 by default) but large enough to show the trend of the field. Figure 3.9 shows the vector field display in different scales.

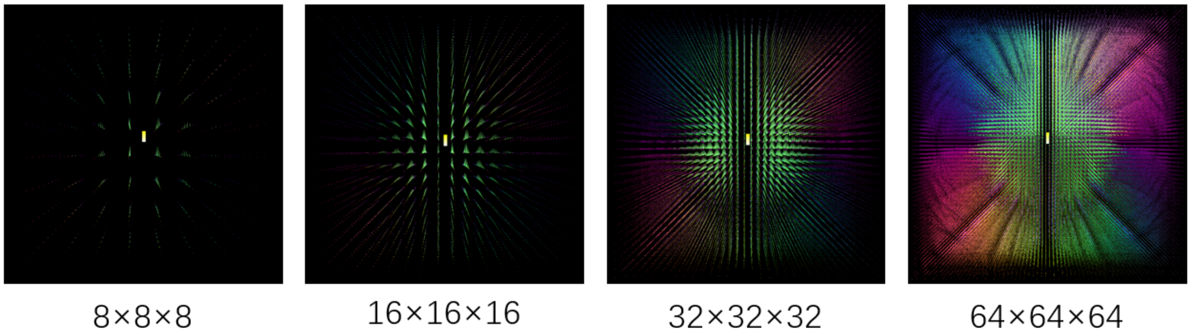


Figure 3.9. Four display results under different view scales

Figure 3.9 shows the rendering details for each vector and a $64 \times 64 \times 64$ view scale result. The color of each interpolation vector is related to its direction, and the length represents the vector length. For interpolation vectors, the start of the vector will be brighter, and the end of the vector will be darker. The input vector has a larger line width, and the color is bright yellow and white. However, if the view scale is too big, the input vectors will be blocked. So, the program uses 32 × 32 × 32 as a default view scale.

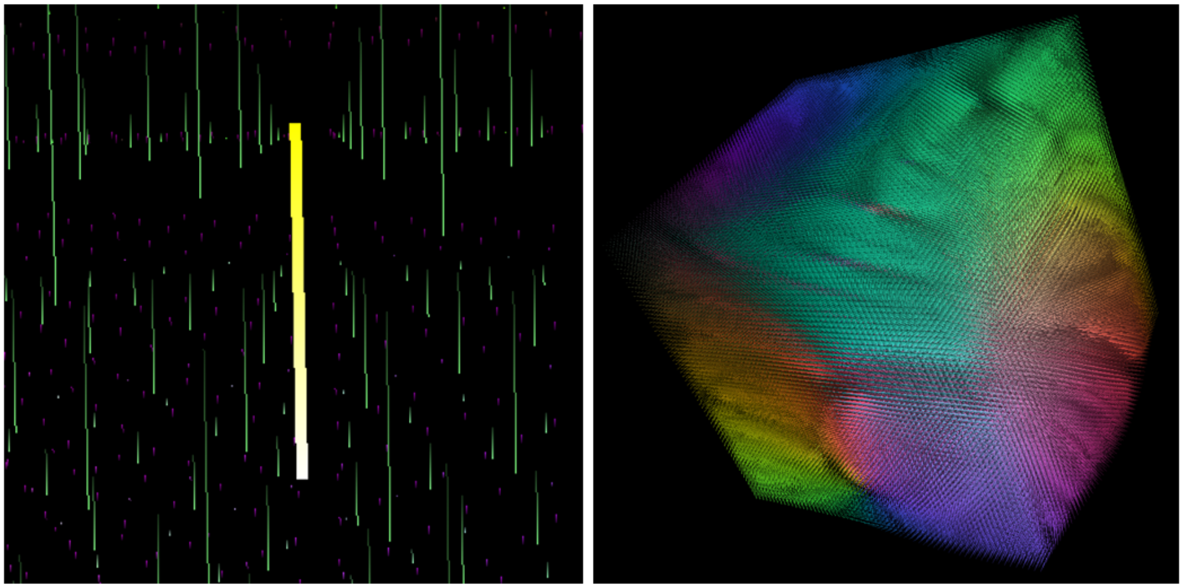


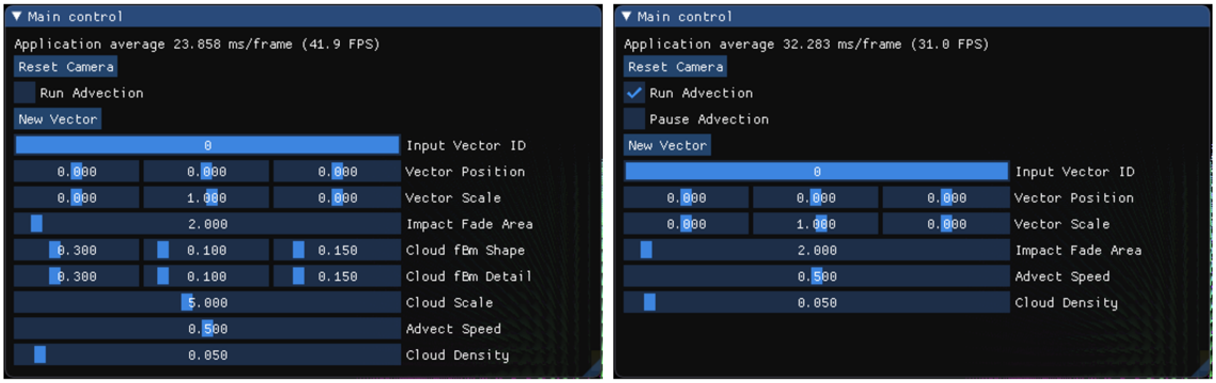
Figure 3.10. Rendering detail of vectors (Left) and a 64 × 64 × 64 view scale result (Right)

3.8 User Interaction and Interface

The user can make multiple interactions during the animating process, including moving the camera by mouse drag and setting parameters through the user interface. The program will show different user interfaces in different states. If the value is unable to modify in the current state, the input slider will be hidden. The camera implementation refers to the volume rendering CUDA example project. Mouse action will update the view matrix. The view matrix affects the rendering of VBO vectors and the camera during volume rendering. So, the vector field and cloud volume will rotate at the same angle.

The user can use checkboxes to switch between three program states: cloud shaping, advection, and pause. The default state is cloud shaping. After check the “Run Advection” checkbox, the program will switch to the advection state and hide cloud shaping parameters.

In this state, the “Pause Advection” checkbox will appear. The user can check it to pause or continue the advection. If the user unchecks the “Run Advection” checkbox, the program will switch back to cloud shaping and automatically hide and uncheck the “Pause Advection” checkbox. Figure 3.11 shows the main parameter window in cloud shaping and advection states. Only during the cloud shaping state, the user can shape the cloud by modifying the fBm parameters. There are two groups of fBm parameters. One is for the cloud shape, and the other is for the cloud details. The user can set the scale of the detail by a slider as well.



Application average 23.858 ms/frame (41.9 FPS)
☐ Run Advection
0 Input Vector ID
0.000 0.000 0.000 Vector Position
0.000 1.000 0.000 Vector Scale
2.000 Impact Fade Area
0.300 0.100 0.150 Cloud fBm Shape
0.300 0.100 0.150 Cloud fBm Detail
5.000 Cloud Scale
0.500 Advect Speed
0.050 Cloud Density

Application average 32.283 ms/frame (31.0 FPS)
☑ Run Advection
☐ Pause Advection
0 Input Vector ID
0.000 0.000 0.000 Vector Position
0.000 1.000 0.000 Vector Scale
2.000 Impact Fade Area
0.500 Advect Speed
0.050 Cloud Density

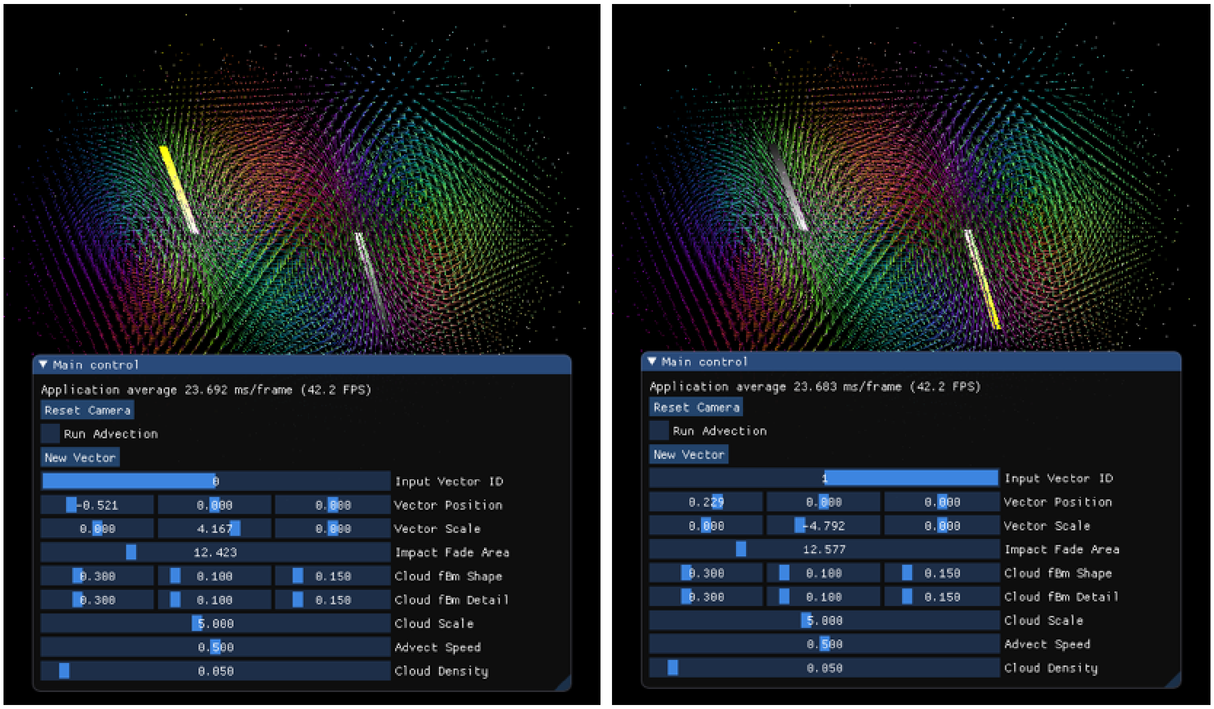
Figure 3.11. Main parameter window in cloud shaping state (Left) and advection state (Right)

The user can add a new vector by clicking the “New Vector” button until the input vectors’ number reach the maximum number. The user can use the “Input Vector ID” slider to pick a vector to modify its position, scale, and weights during interpolation. Figure 3.12 shows the user use the “Input Vector ID” slider to switch between two different input vectors. The picked vector will be highlighted in yellow, and the others will be gray.

3.9 Research Questions

Research question 1: How to generate volumetric clouds?

Based on the literature, the process of cloud generation and rendering will be based on the approach of Lague. 3D noise textures will be used, and the cloud will be rendered in a box. The noise textures will be generated by CUDA. The generation kernel will be called once before the main loop. Since the noise texture will remain the same during the interaction



```
▼ Main control
Application average 23.692 ms/frame (42.2 FPS)
Reset Camera
☐ Run Advection
New Vector
0                                 Input Vector ID
-0.521     0.000     0.000        Vector Position
0.000      4.167     0.000        Vector Scale
12.423                            Impact Fade Area
0.300      0.100     0.150        Cloud fBm Shape
0.300      0.100     0.150        Cloud fBm Detail
5.000                             Cloud Scale
0.500                             Advect Speed
0.050                             Cloud Density
```

```
▼ Main control
Application average 23.683 ms/frame (42.2 FPS)
Reset Camera
☐ Run Advection
New Vector
1                                 Input Vector ID
0.229      0.000     0.000        Vector Position
0.000     -4.792     0.000        Vector Scale
12.577                            Impact Fade Area
0.300      0.100     0.150        Cloud fBm Shape
0.300      0.100     0.150        Cloud fBm Detail
5.000                             Cloud Scale
0.500                             Advect Speed
0.050                             Cloud Density
```

Figure 3.12. Using the "Input Vector ID" slider to switch between two input vectors

and rendering processes of the algorithm, there is no good to generate the noise per frame on the GPU since it is busy enough to deal with the calculation of animating and rendering processes. Furthermore, the texture can even be generated once and loaded into the program to make the algorithm more efficient. The algorithm will not need to generate new textures every time after it started since the noise will look pretty similar generated by the same function. If needed, The noise texture could be made in other software and load to the program.

The program will use Worley noise as Schneider suggested in his study to achieve the cloud's blobby appearance. Worley noise is a noise function introduced by Steven Worley in 1996, widely used in natural phenomena simulation in the computer graphics area. While in Schpok *et al.* choose Perlin noise, a type of gradient noise developed by Ken Perlin in 1983, for this step. Consider their sampling method and research preference is to simulate multiple types of clouds, it is also a possible choice. In general, both Perlin noise and Worley noise will be used for cloud generation. The sampling method is to use fractional Brownian motion to combine textures in different frequencies.

Research question 2: How to make art-directable modifications to the cloud?
The user will input vectors to the program, and the inputs will be interpolated to a vector field. The volume cloud area will be divided into small blocks as Dobashi *et al.* did in their study. The program will use 3D texture advection based on the interpolation vector field and cloud density from the previous frame. When the user inputs are not empty, the clouds will advect based on the vector field per frame.

Research question 3: How to render the result to the screen?
For vector field display, the program will sample from the vector field data and set the color based on the direction. The sampling result will be bound to VBO and rendered by OpenGL. For volume cloud rendering, the method implemented by Lague will be used during ray marching. The output of this method is the accumulated color of the cloud for each ray. The result will be stored in a pixel object buffer and display by ImGui image.

Research question 4: How to identify and analyze the result of the clouds?
Referring to the literature, other researchers will take screenshots for the rendering result in each short period and compare them to see how the results change over time [4], [6], [15], which called image sequence. So, the research will use this method too. Figure 3.13 is an example of the result display by image sequence. The researcher will also analyze the framerate to assess whether interactive performance has be achieved.

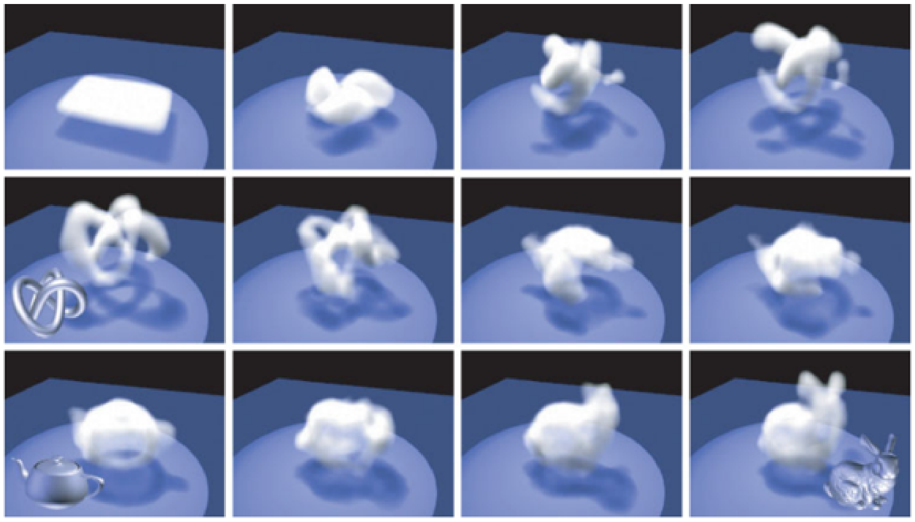


Figure 3.13. An example of image sequence from Hong and Kim

3.10 Data Collection and Analysis

The results of the deliverable shown in the thesis are image sequences generated by multiple tests. The researchers will input different vectors and output the volumetric cloud rendering result every multiple frame. Moreover, use these output images to make an image sequence. Suppose the input vectors can be interpolated into the divergence-free vector field, and the vector field can modify the volumetric cloud in a fluid-like way. In that case, this study's hypotheses will be considered correct.

Table 3.1. Major data buffers in this project

Number	**Name**	**Data Type**	**Location**	**Default Scale**	**Function**
1	h_input_pos	vec3	host	5	User inputs
2	h_input_vec	vec4	host	5	User inputs
3	h_interp_c	vec3	host	5	Interpolation matrix
4	d_input_pos	vec3	device	5	User inputs
5	d_input_vec	vec4	device	5	User inputs
6	d_interp_c	vec3	device	5	Interpolation matrix
7	h_input_vbo	vec3	host	5 × 4	Display user inputs
8	d_VF_vbo	vec3	device	32 × 32 × 32 × 4	Display interpolation VF
9	d_VF_0	vec4	device	256 × 256 × 256	Store VF and cloud density
10	d_VF_1	vec4	device	256 × 256 × 256	Store VF and cloud density
11	h_CloudShapeData	float4	host	128 × 128 × 128	Store cloud shape noise
12	h_CloudDetailData	float3	host	32 × 32 × 32	Store cloud detail noise
13	d_render_result	vec3	device	512 × 512	Display volume rendering

4. ANALYSIS OF RESULTS

This section presents and analyzes the data of the deliverable project in this research. The data includes vector field interpolation results, cloud shaping results, and the advection and animation image sequence. All results are gathered in real-time around 30 fps.

4.1 Vector Interpolation

4.1.1 Single Input Vector

Figure 4.1 shows an interpolation result in a lower view scale. There is one input vector at the center of the vector field, pointing up. The interpolation vectors' directions perform as vortexes around the input vector. These vortexes will help to imitate fluid phenomenons of cloud animation.

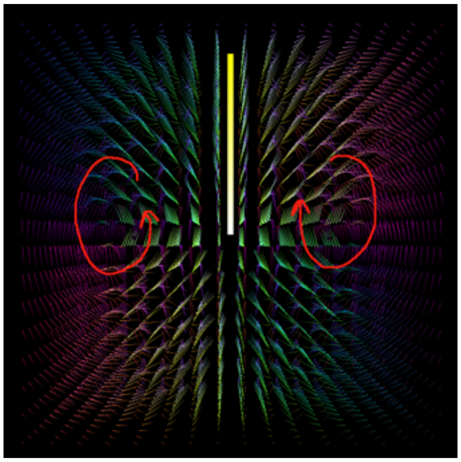


Figure 4.1. Red arrows show the tendency of curves at each side of the input

Figure 4.2 shows the interpolation vector field with one input at the center and the bottom with the same vector scale and weight. When the vector field's current position is outside the influence area of the input vector, the interpolation result will be zero.

Figure 4.3 shows the interpolation vector field with one input in different weights. The position and scale of the inputs are the same. The input vector is pointing up, which maps to green when shading. So, the interpolation vectors are mostly green when the weight is large.

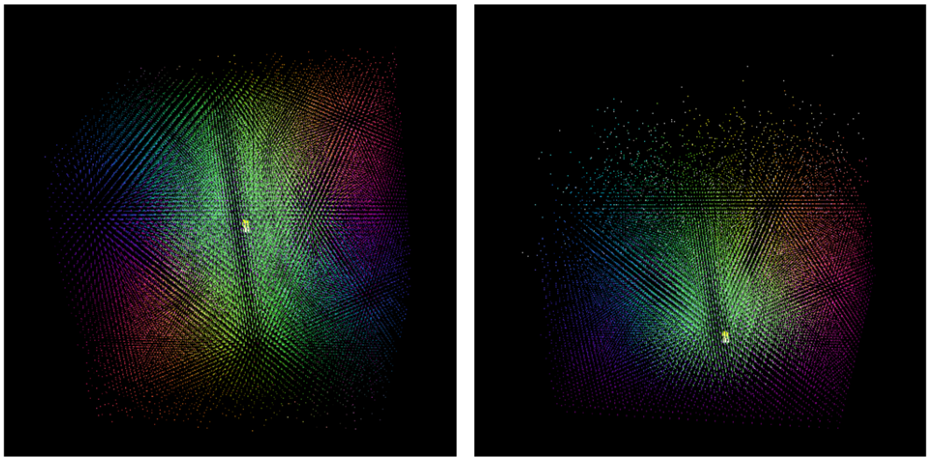


Figure 4.2. Interpolation vector field with one input at the center (Left) and the bottom (Right)

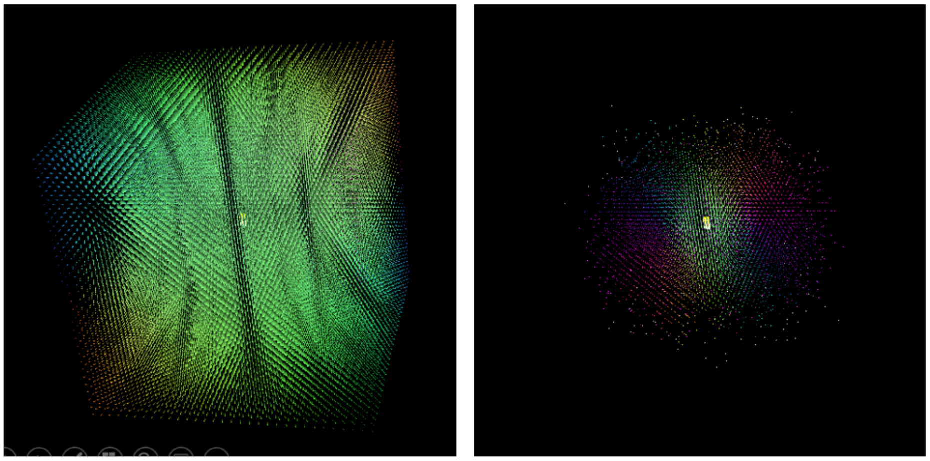


Figure 4.3. Interpolation vector field with large weight (Left) and small weight (Right)

Figure 4.4 shows the interpolation vector field with one input at the center but pointing at different directions. Because the color of the vectors is mapped from the direction, the vector field has different shading color.

4.1.2 Multiple Input Vectors

Figure 4.5 shows the interpolation vector field with multiple inputs. The vector field is transiting smoothly between different inputs and generates complex vortex patterns. The

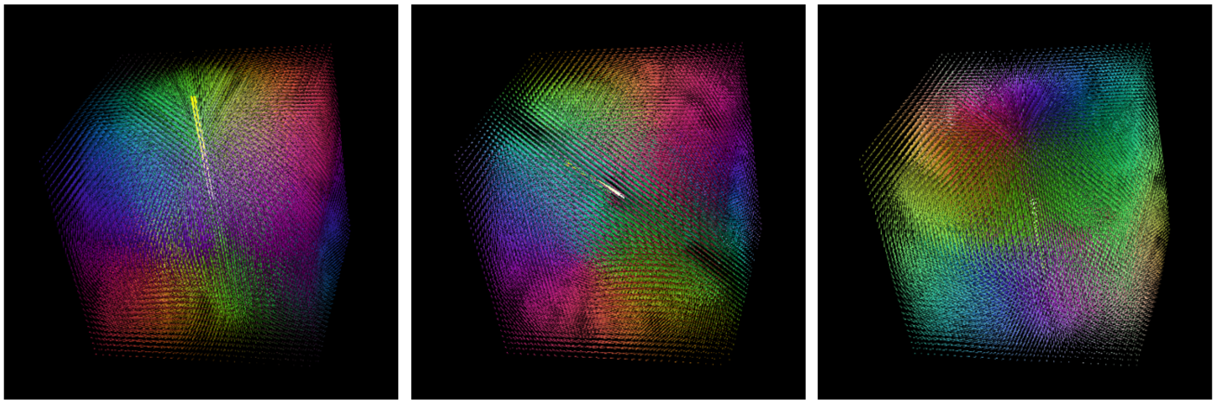


Figure 4.4. Interpolation vector field with one input point at up (Left) left up (Medium) and down (Right)

red arrows highlight some vortexes from the current view direction. The functions of these vortexes will be displayed by cloud advection in the following chapter.

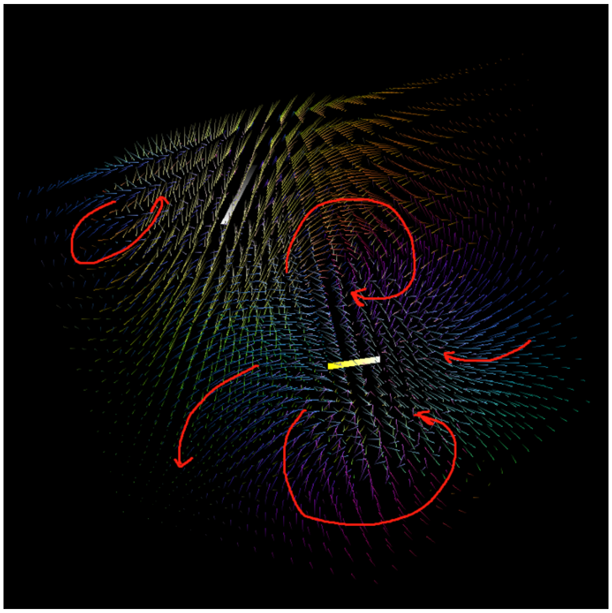


Figure 4.5. Interpolation vector field with two inputs

However, an interpolation problem will occur with multiple inputs. When two inputs are near and their directions are different, the interpolation result will blow up. The program clamps the result of interpolation to avoid this effect. Figure 4.6 shows the vector field before and after clamp when the result blows up. The program limits the input number because there is not enough place for each input when there are many of them. With the blow-up interpolation result, advection will trace for unreasonable large steps, which makes the animation distorted.

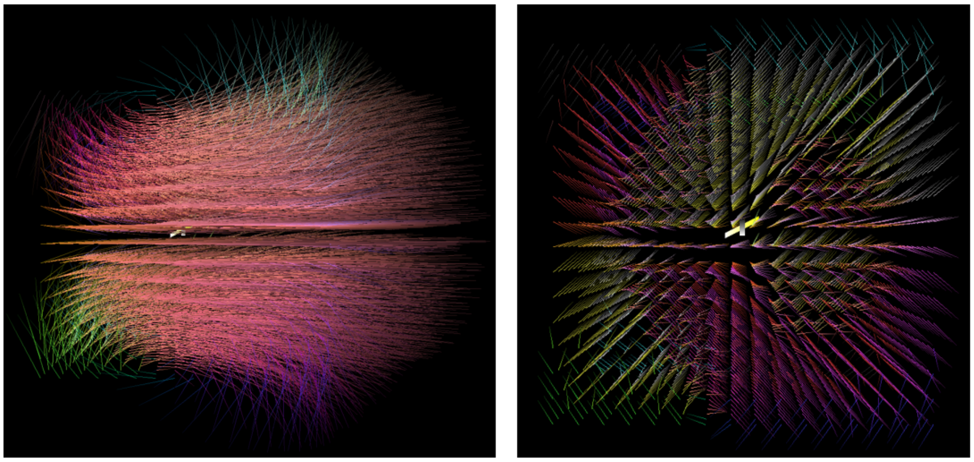


Figure 4.6. Interpolation blow up result before (Left) and after (Right) clamp

4.2 Cloud Generation

Figure 4.7 shows the cloud's generation results influenced by different texture channels. The left one shows a cloud result combined by the all shape noise channel. The right figures show the impact of each shape channel.

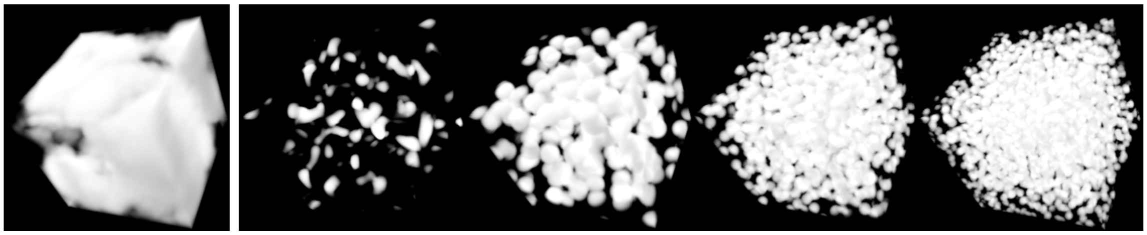


Figure 4.7. Rendering result of the cloud with different shape fBm values

Figure 4.8 shows the cloud's generation results influenced by different texture channels. The left one shows a cloud result without any details. The right figures show the impact of each detail channel. More details will appear when using high resolution Worely noise textures.

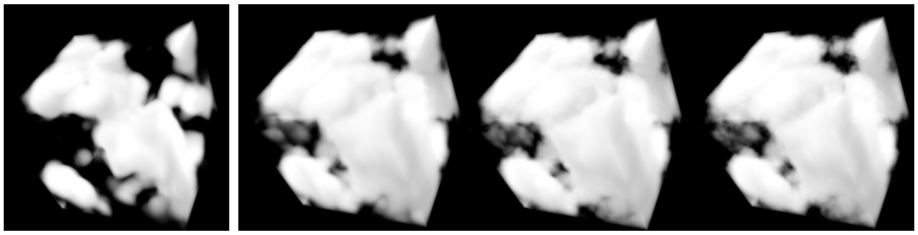


Figure 4.8. Rendering result of the cloud with different detail fBm values

4.3 Cloud Animation

Figure 4.9 shows the cloud's animating result from different angles under the same input.

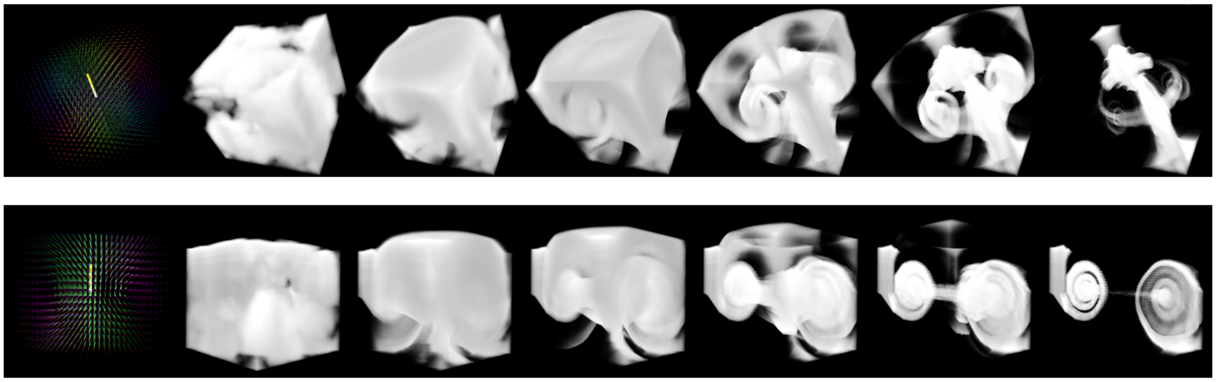


Figure 4.9. Rendering result of the cloud with single input

Figure 4.10 shows the cloud's rendering result with different input vectors in image sequences.

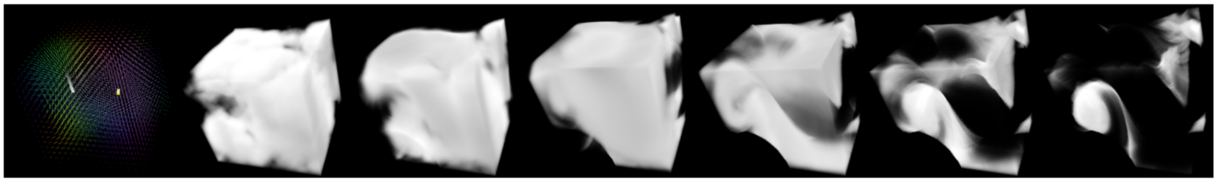


Figure 4.10. Rendering result of the cloud with two input vectors

4.4 Performance

Table 4.1 shows the program performance under different buffer settings in release mode. The FPS is for the advection state, which is the lowest FPS among all states. The increase of vector field buffer highly increases GPU memory usage and decreases the FPS. Due to the memory malloc process in this program, GPU memory usage will not change while running. The increase of inputs will slightly decrease the FPS because the more inputs the program have, the more calculation will be down during the advection. The blow-up result will not influence the FPS under the same input number.

Table 4.1. Program performance in this study

Inputs Num	VF Buffer Scale	FPS	GPU Memory Usage
1	$512 \times 512 \times 512$	11	4.2 GB
1	$256 \times 256 \times 256$	33	0.8 GB
1	$128 \times 128 \times 128$	>120	0.3 GB
2	$256 \times 256 \times 256$	31	0.8 GB
3	$256 \times 256 \times 256$	30	0.8 GB
4	$256 \times 256 \times 256$	30	0.8 GB
5	$256 \times 256 \times 256$	29	0.8 GB

5. SUMMARY

5.1 Conclusion

The problem addressed by this research is that a pure physics-based simulation of volumetric clouds can not provide a flexible controller to designers and animators in real-time. This study aims to implement a real-time art-directable method for users to interactively animate volumetric clouds. The study's hypothesis is vector field interpolation with divergence minimization combined with advection will be able to produce a plausible cloud animation while giving artists control over the final result. In this research, the user can animate shape volumetric clouds by a few inputs. The inputs are used to generate a divergence-free interpolation vector field. The vector field is used in the advection process for animation creation. By adding and modifying input vectors, the user can interactively animate the clouds as their wish. So, the hypothesis is considered true.

However, the interpolation result's value is not guaranteed to be divergence-free on some specific inputs. When two input vectors are near to each other, the interpolation result will blow up. The vector field's divergence is still minimized to zero, but all vectors' total length is too large and not making sense. It is like getting infinity power from nowhere. The program in this study chooses to clamp the maximum length of each vector to a reasonable value. The vector field's divergence may not equal zero, but the animation in such situations is still plausible.

The research process that creates the interpolation vector field has the same meaning as calculating a flow map in 3D per frame. The flow map uses physics-based algorithms and takes a much longer time to calculate, while the divergence-free interpolation is much faster. The output of these two processes are both textures storing vectors at each point and used for advection or distorting. The research is trying to use the interpolation result to replace the physics-based simulation result. Though the animation made by this research looks plausible, the volumetric clouds are not conservative under the interpolation vector field. It means, for most interpolation vector fields, the total density of clouds inside the vector field area will get down little by little when there is no source in the volume. However, animations using a flow map are loopable no matter the distorting happens on albedo or normal textures.

Compared with other cloud animating methods in the literature related to this study, the delivery program focuses on simulating the wind flow that animates the clouds. This program's result is similar to what Yu, Bruneton, Holzschuch, *et al.*'s result in Figure 2.6, but can not achieve a specific shape result like Dobashi *et al.*, Treuille *et al.* or McNamara *et al.* Their results showed in Figure 2.3, Figure 2.5 and Figure 2.4, all need to generate a target or keyframe for the animation and offline computing. The method in this study offers real-time user interaction will the volumetric clouds. The cloud can not turn into specific shapes by the user inputs but can plausibly move as blew by wind flow. If the user can use multiple input vectors to represent the target shape, the cloud in this study may achieve a stable shape. Currently, it is easier to dig a hole inside the cloud by input vectors with small weights. The cloud inside the affected area will be blown out of the vector field or compress, and the rest remain unchanged. However, it is still difficult for this method to hold a certain shape. The researcher considers the study's method as a real-time approach to generate a 3D interactable flow map. Users will have a chance to animate the cloud under specific and art-directable winds.

5.2 Future works

There are a few suggestions for future works:

First, volumetric clouds in this study are placed inside a box, which has limited video game usage. It will be helpful for skybox rendering if the algorithm could be extended to hemisphere coordinate systems. Alternatively, implement a 2D solution for 2D games or background.

Second, the user interface could be improved. Currently, the user can only add input vectors and modify them by button and sliders, which is not very convenient. The user can not remove the input either. If the program could be extended to a VR system and let the user input by a VR controller and surrounded by the volumetric clouds, the interaction experience will be highly improved. The input vectors may be automatically generated by the controller's motion trails and fade after a few seconds.

Last, the data management in this project still needs optimization. The vector field data does not have a read-write conflict and calculated on the device. It could be unpacked from the data density and use CUDA surface for a linear interpolation sampling during advect.

REFERENCES

[1] F. Bauer, "Creating the atmospheric world of red dead redemption 2: A complete and integrated solution," SIGGRAPH2019, PowerPoint slides, Jun. 2019, [Online]. Available: http://advances.realtimerendering.com/s2019/index.htm.

[2] A. Schneider, "The real-time volumetric cloudscapes of horizon: Zero dawn," SIGGRAPH2015, PowerPoint slides, Aug. 2015, [Online]. Available: http://advances.realtimerendering.com/s2015.

[3] R. Kevin and S. Peter, *Partly cloudy*, Short Film, 2009. [Online]. Available: https://www.imdb.com/title/tt1425244/?ref_=ttmi_tt.

[4] Y. Dobashi, K. Iwasaki, Y. Yue, and T. Nishita, "Visual simulation of clouds," *Visual Informatics*, vol. 1, no. 1, pp. 1–8, 2017, ISSN: 2468-502X. DOI: 10.1016/j.visinf.2017.01.001.

[5] J. Stam, "Stable fluids," *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pp. 121–128, 1999. DOI: 10.1145/311535.311548.

[6] J.-M. Hong and C.-H. Kim, "Controlling fluid animation with geometric potential," *Computer Animation and Virtual Worlds*, vol. 15, no. 34, pp. 147–157, 2004. DOI: 10.1002/cav.17.

[7] S. Lague. (2019). Coding adventure: Clouds, [Online]. Available: https://youtu.be/4QOcCGI6xOU.

[8] M. Ikits, J. Kniss, A. Lefohn, and C. Hansen, "Gpu gems," in. Boston: Addison-Wesley Professional, 2004, ch. 39. Volume Rendering Techniques. [Online]. Available: https://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch39.html.

[9] M. Harris, "Gpu gems," in. Boston: Addison-Wesley Professional, 2004, ch. 38. Fast Fluid Dynamics Simulation on the GPU. [Online]. Available: https://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch38.html.

[10] I. Amidror, "Scattered data interpolation methods for electronic imaging systems: A survey," *Journal of Electronic Imaging*, vol. 11, pp. 157–176, Apr. 2002. DOI: 10.1117/1.1455013.

[11] NVIDIA. (2021). Cuda c++ programming guide, [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[12] H. Mark. (2012). An easy introduction to cuda c and c++, [Online]. Available: https://developer.nvidia.com/blog/easy-introduction-cuda-c-and-c.

[13] CNET. (2016). How disney's 'moana' created its amazing water effects, [Online]. Available: https://youtu.be/S-HG8IA-2TI.

[14] CNET. (2016). A look at how disney gave water a little sass in 'moana', [Online]. Available: https://youtu.be/NldkLGYYJKg.

[15] Y. Dobashi, K. Kusumoto, T. Nishita, and T. Yamamoto, "Feedback control of cumuliform cloud formation based on computational fluid dynamics," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–8, Aug. 2008, ISSN: 0730-0301. DOI: 10.1145/1360612.1360693.

[16] A. Treuille, A. McNamara, Z. Popović, and J. Stam, "Keyframe control of smoke simulations," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 716–723, Jul. 2003, ISSN: 0730-0301. DOI: 10.1145/882262.882337.

[17] A. McNamara, A. Treuille, Z. Popović, and J. Stam, "Fluid control using the adjoint method," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 449–456, Aug. 2004, ISSN: 0730-0301. DOI: 10.1145/1015706.1015744.

[18] R. Fattal and D. Lischinski, "Target-driven smoke animation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 441–448, Aug. 2004, ISSN: 0730-0301. DOI: 10.1145/1015706.1015743.

[19] A. Bouthors and F. Neyret, "Modeling clouds shape," in *Eurographics 2004 - Short Presentations*, M. Alexa and E. Galin, Eds., Eurographics Association, 2004. DOI: 10.2312/egs.20041020.

[20] M. J. Harris and A. Lastra, "Real-time cloud rendering," *Computer Graphics Forum*, vol. 20, no. 3, pp. 76–85, 2001. DOI: 10.1111/1467-8659.00500.

[21] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra, "Simulation of cloud dynamics on graphics hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, ser. HWWS '03, San Diego, California: Eurographics Association, 2003, pp. 92–101, ISBN: 1581137397. [Online]. Available: https://dl.acm.org/doi/10.5555/844174.844189.

[22] J. Schpok, J. Simons, D. S. Ebert, and C. Hansen, "A real-time cloud modeling, rendering, and animation system," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '03, San Diego, California: Eurographics Association, 2003, pp. 160–166, ISBN: 1581136595. [Online]. Available: https://dl.acm.org/doi/10.5555/846276.846299.

[23] S. Hillaire, "Physically based sky, atmosphere and cloud rendering in frostbite," SIGGRAPH2016, PowerPoint slides, Jun. 2016, [Online]. Available: https://www.ea.com/frostbite/news/physically-based-sky-atmosphere-and-cloud-rendering.

[24] R. Fedkiw, J. Stam, and H. W. Jensen, "Visual simulation of smoke," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01, New York, NY, USA: Association for Computing Machinery, 2001, pp. 15–22, ISBN: 158113374X. DOI: 10.1145/383259.383260.

[25] R. Netzel and D. Weiskopf, "Texture-based flow visualization," *Computing in Science and Engineering*, vol. 15, no. 6, pp. 96–102, 2013. DOI: 10.1109/mcse.2013.131.

[26] V. Alex, "Water flow in portal 2," SIGGRAPH2019, PowerPoint slides, Jul. 2010, [Online]. Available: http://advances.realtimerendering.com/s2010.

[27] B. Simon. (2021). Flowmap generator, [Online]. Available: http://www.superpositiongames.com/products/flowmap-generator.

[28] I. Yassine and T. McGraw, "A subdivision approach to tensor field interpolation," in *Workshop On Computational Diffusion MRI*, Citeseer, 2008, pp. 117–124. [Online]. Available: https://web.ics.purdue.edu/~tmcgraw/papers/mcgraw-cdmri-2008.pdf.

[29] I. Yassine and T. McGraw, "4th order diffusion tensor interpolationwith divergence and curl constrained bézier patches," in *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, IEEE, 2009, pp. 634–637. DOI: 10.1109/ISBI.2009.5193127.

[30] C. P. McNally, "Divergence-free interpolation of vector fields from point values—exact b= 0 in numerical simulations," *Monthly Notices of the Royal Astronomical Society: Letters*, vol. 413, no. 1, pp. L76–L80, 2011. DOI: 10.1111/j.1745-3933.2011.01037.x.

[31] A. A. Mitrano and R. B. Platte, "A numerical study of divergence-free kernel approximations," *Applied Numerical Mathematics*, vol. 96, pp. 94–107, 2015. DOI: https://doi.org/10.1016/j.apnum.2015.05.001.

[32] M. Nelson, C. Roger, and W. Dean, "Visualizing wind velocities by advecting cloud textures," *Proceedings Visualization '92*, pp. 179–184, 1992. DOI: 10.1109/VISUAL.1992.235210.

[33] Q. Yu, E. Bruneton, N. Holzschuch, and F. Neyret, "Lagrangian texture advection: Preserving both spectrum and velocity field," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 11, pp. 1612–1623, Nov. 2011, ISSN: 1941-0506. DOI: 10.1109/TVCG.2010.263.

[34] F. Neyret, “Advected textures,” in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’03, San Diego, California: Eurographics Association, 2003, pp. 147–153, ISBN: 1581136595. [Online]. Available: https://hal.inria.fr/inria-00537472/document.