Procedural 2D Cumulus Clouds Using Snaxels

by

Ramin Modarresiyazdi, **B.Sc.**

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
Ottawa, Ontario
April, 2017

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Procedural 2D Cumulus Clouds Using Snaxels

submitted by **Ramin Modarresiyazdi**, **B.Sc.**

in partial fulfillment of the requirements for the degree of

Master of Computer Science

Professor David Mould, Thesis Supervisor

Professor Oliver van Kaick,
School of Computer Science

Professor Jochen Lang,
School of Electrical Engineering and Computer Science

Professor Michiel Smid, Chair,
School of Computer Science

Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
April, 2017

Abstract

This thesis develops a procedural modeling algorithm to model cumulus clouds or objects with similar shapes and surfaces in 2 dimensions. Procedural modeling of clouds has been extensively studied and researched in computer graphics. Most of the literature follows physical characteristics of clouds and tries to model them using physically-inspired simulations. Cumulus clouds, volcanic ash clouds and similar naturally shaped bodies come in many different shapes and sizes, yet the surfaces share similarities and possess high irregularities in details which makes them difficult to model using conventional modeling techniques. We propose a framework for modeling such surfaces and shapes with minimal user intervention. Our approach uses an active contour model which propagates through a tessellation. In this thesis, we will describe our technique for modeling cloud looking structures and we will present our results of our algorithm and show case a simple user interactive framework as well.

Acknowledgments

It cannot be argued with that the most influential person in my graduate studies at Carleton University has been my supervisor, Dr. David Mould. I would like to show my greatest gratitude to him. I am especially grateful to his guidance, advice and patience. His kindness and knowledge has left me word-less. I cannot hope to have anyone better to be my supervisor.

I would like to thank the thesis committee for reviewing my work and giving me valuable suggestions. I must also express my appreciation to my colleagues and friends in GIGL lab and the School of Computer Science. Also I would like to thank Carleton University and GIGL lab for their persistent financial support. And finally, I would like to thank those who shaped me into who I am. Without them and their support the writing of this thesis would not have been possible.

We used many images from Flickr under a Creative Commons license. Thanks to the numerous photographers who provided the material.

Table of Contents

Abstract **iii**

Acknowledgments **iv**

Table of Contents **v**

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

1.1 Problem Statement 1
1.2 Motivation 2
1.3 Snaxels 3
1.4 Statement of Contribution 4
1.5 Overview 4

2 Background **6**

2.1 Procedural Modeling of Natural Irregular Objects 6
2.1.1 Clouds 6
2.1.2 Rocks 7
2.1.3 Terrain 8
2.1.4 Cities 10
2.2 Sketch-Based Modeling 10
2.3 Noise 11
2.3.1 Perlin Noise 11
2.3.2 Gabor Noise 11
2.3.3 Worley Noise 11

2.3.4 Stochastic Modeling . . . 12
2.4 Curve Synthesis . . . 12

3 Algorithm **14**
3.1 Introduction . . . 14
3.2 Snaxels . . . 15
3.2.1 Birth . . . 16
3.2.2 Merge . . . 18
3.2.3 Growth . . . 20
3.2.4 Death . . . 21
3.2.5 Stop . . . 21
3.3 Curves . . . 21
3.3.1 Updating Curves . . . 22
3.4 Speed Scheme . . . 26
3.4.1 Base Speed Scheme . . . 28
3.4.2 Circular Speed-Assigning Scheme . . . 32
3.5 Growth Strategies . . . 36
3.5.1 Length-Limits Table . . . 36
3.5.2 Clusters . . . 40
3.5.3 Sketch-based . . . 42

4 Results & Discussion **44**

5 Future Work **61**
5.1 Snaxels in 3D: Mesh Growth Using Snaxels . . . 62

6 Conclusion **64**

List of References **65**

Appendix A Snaxels in 3D: Mesh Growth Using Snaxels **70**
A.1 Transition . . . 70
A.2 Case Based Approach . . . 73
A.3 Hole Patching Approach . . . 76

List of Tables

3.1 Parameters used in the simulations 32
3.2 Parameters used in the simulations 35
3.3 Example of a Weight-Limits Table 37
3.4 Modified version of Table 3.3 39
3.5 Parameters used in the simulations 39
3.6 Parameters used in the simulations. N_s stands for number of snaxels on the snake. N_v stands for number of tessellation's vertices. 42
3.7 The Weight-Limits Table used in figure 3.23 42
3.8 Parameters used in the simulations. N_s stands for number of snaxels on the snake. N_v stands for number of tessellation's vertices. 43
4.1 Length-limits table used in our results 48
4.2 Parameters used in the simulations. n_c stands for number of clusters. 57
4.3 One of the length-limits table used in our results 58
4.4 One of the length-limits table used in our results 58
4.5 Length-limits table used in our results 58
4.6 Length-limits table used in our results 59
4.7 Length-limits table used in our results 59
4.8 Length-limits table used in our results 59
4.9 Length-limits table used in our results 60

List of Figures

1.1 Example of a natural phenomenon. Eyjafjallajökull volcanic eruption in Iceland. 2

1.2 Snaxels moving on a tessellation;. Red lines indicate segments of the snake. Yellow vertices are snaxels and the blue ones are tessellation's vertices. 4

3.1 The snake propagating through the tessellation. Dashed lines indicate the snake. Black vertices are snaxels and blue ones are tessellation's vertices. 15

3.2 Before and after a birth event. (a) Snaxel P is about to arrive at its destination. (b) After snaxel P arrives, it is deleted and the birth event takes place which results in new snaxels being born and added to the snake. 17

3.3 Before and after a merge event. (a) Snaxels m_1 and m_2 are about to merge with one another. (b) The snaxels are deleted and their neighbors connected to one another. 19

3.4 A snaxel with both of its connections on a single triangle. Note that this situation is impossible. 20

3.5 Types of connections between a snaxel and its neighbors. (a) Snaxels S has each of its connections on a separate triangle. (b) Snaxels S has one of its connections on the edge it is growing on. 20

3.6 Example of a curve. Snaxels S_0 to S_n form a curve. 22

3.7 Example of breaking of a curve. The curve designated by snaxels S_0 to S_n, is divided into to new curves; one from S_0 to S_i and another from S_j to S_n. 23

3.8 Before and after a merge event of two non-connected snaxels. (a) Snaxels m_1 and m_2 are about to merge with one another. (b) Their merge results in part of each of the curves they belong to being removed. 24

3.9 Example of merging of a Curve with another. (a) A Small curve starting from snaxel C to D has reached the minimum allowed length. (b) The curve has merged with its neighboring curve starting from A to B, thus expanding the neighboring curve to expand its range to A to D. 25
3.10 A Volcanic ash-cloud. 27
3.11 A result of linear interpolation over each curve. The linear interpolation is computed over the curve by setting the middle of the curve as the maximum input velocity and the corners at zero. Note that the inner contours are for illustrating and visualizing the growth process and that different curves are visualized with different colors for simplification. 28
3.12 Curve 'B' is joint to curves 'A' and 'C' at snaxels 'j_{BA}' and 'j_{BC}' respectively. 29
3.13 Result of computing each snaxels' speeds by adding a basic speed to the linear interpolation of the snaxels' locations in the curve. 31
3.14 Result of controlling the base speed through an effect coefficient. . . . 32
3.15 Comparison of Circular scheme method and linear interpolation over the curves. Both results were generated with the same values for each parameter shown in Table 3.2. 33
3.16 Circular Speed-Assigning Scheme Example. The velocity of snaxel 's' on the curve is set to the length of the vector '$\vec{sx}$'. 34
3.17 Example of an arbitrary shape with occlusions. The occlusions prohibits the method to be able to assign a speed to the snaxel 's'. . . . 35
3.18 Example of a cumulus cloud's surface 37
3.19 A result obtained by controlling the curves' length-limits through Table 3.3. The values for each parameter are shown in Table 3.5. 38
3.20 A result obtained by controlling the curves' length-limits through Table 3.4. The values for each parameter is shown in Table 3.5. 39
3.21 Examples of Clouds' Structures 40
3.22 A result obtained growing 2 clusters following Table 3.3. The horizontal line at the bottom of the image indicates that the growth has reached the boundaries of the screen. In our implementation, the snaxels are stopped when they reach the boundaries of the screen. The values for each parameter is shown in Table 3.6. 41

3.23 User attempt to model a mushroom cloud. Left shows the result and the initial shape drawn by the user. Right shows the final result. The process was stopped by the user. The values for each parameter is shown in Table 3.8. 43
4.1 (a) Eyjafjallajökull volcanic eruption in Iceland. (b) One of our results. (c) A 2 dimensional silhouette extracted from the image. 45
4.2 Comparison between our results and real world cumulus clouds and volcanic ash-clouds. 47
4.4 A 'neck' feature in one of our results 49
4.3 Results obtained through same settings with exception in number of clusters From left to right column, 2,3 and 4 clusters were used. Table 4.1 was used for length limit manipulations. 50
4.5 Result obtained by increasing the number of clusters 51
4.6 A result with high number of clusters 51
4.7 Results obtained by reducing the final row of the length-limit used shown in Table 4.1 to 25. Figure 4.7(b) is the same result as the one shown in Figure 4.2(e). 52
4.8 Magnifying details on the result in Figure 4.2(e) 53
4.9 Comparison of clusters' locations influence 54
4.10 Comparison between our results and previous works. (b) A result from Bouthors and Neyret [6]. (d) A result from Neyret [43]. (f) A result from Trembilski and Broßler [63]. 56
5.1 Example of a coral reef; Elkhorn coral near Vega Baja, Puerto Rico. . 62
A.1 Different types of connections between snaxels inside tetrahedras. The dashed edges indicate that they are occluded. 72
A.2 Before and after a birth event on a mesh. The mesh before the birth event (a), the parent is removed and the children have formed a cap (b). Note that only a part of the mesh is visualized. 73
A.3 Before a merge event. The color green indicates the outside surface of the mesh and the color red indicates the inside surface of the mesh. . 74
A.4 Before and after a merge event inside a tetra 75
A.5 Example of a re-stitching problem 75

A.6 Two different scenarios for a merge event. The color green indicates the outside surface of the mesh and the color red indicates the inside surface of the mesh. Merging snaxels are illustrated by yellow squares. 76
A.7 Two different types of holes. The color green indicates the outside surface of the mesh and the color red indicates the inside surface of the mesh. 77
A.8 Stitched holes with a patch. 78

Chapter 1

Introduction

While manual modeling has the power of flexibility, it can also be tedious when it comes to highly irregular, asymmetrical and natural looking objects and phenomena. Automatic processes can make it much easier to generate irregular objects. However, automatic processes may not capture the details and specific features that a user wishes the ending results to have, Interactive modeling can allow the user to manipulate the modeling process as it is happening. This way the result will be more satisfactory and tailored to the user's needs. Interactive modeling with automatic processes can help procedural modeling of irregular objects by providing control over an automated process.

1.1 Problem Statement

Most work on procedural modeling of clouds renders them as density fields. Although density fields can obtain realistic results, a simplified representation would suffice to produce the more opaque models such as cumulus clouds or volcanic ash-clouds. Although clouds are essentially volume densities, some of them appear to be opaque. A simplified approach to modeling such types of clouds may be more suitable due to the fact that only the outside facade of opaque clouds is visible, meaning that light does not penetrate them deeply and therefore, they appear opaque at a distance. Opaque clouds are particularly hard to model because of their intricate details. Such details are caused by physical processes. Computer simulations are used by researchers to try to mimic these processes. However, simulations based on physical processes can be costly and complicated.

Modeling clouds as opaque objects is difficult due to the complexity of the cloud's surface. Since the dynamic processes inside clouds cause certain regularities and repeated patterns on the surface in the macro scale, more customized algorithms are needed in order to model these surfaces. The problem we face is procedurally modeling cumulus clouds or volcanic ash-clouds as opaque objects in 2 dimensions. This can be broken down into modeling their arbitrary complex shapes and capturing their boundary details.

1.2 Motivation

Manually modeling complex irregular natural surfaces is tedious. Figure 1.1 shows an example of such surfaces. These surfaces possess a lot of details which are hard to capture by hand. The solid appearance of objects such as cumulus clouds motivated us to treat them as just a surface or boundary rather than a volume density. The growth of these irregular boundaries is a process, and therefore it can be captured by using a simulation. To represent surfaces, meshes are widely used. They require less memory compared to volumetric approaches. But growing meshes has its own problems such as surface point density and self-collision.

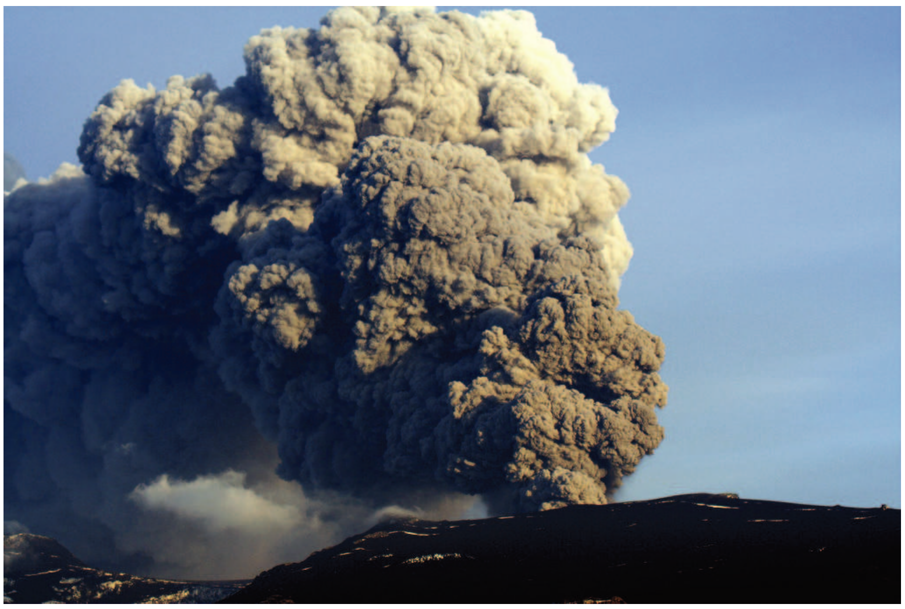


Figure 1.1 Example of a natural phenomenon. Eyjafjallajökull volcanic eruption in Iceland.

We introduce an algorithm which models natural complex and irregular boundaries such as cumulus clouds in 2 dimension. Since the exterior shape of a cumulus cloud is opaque and it appears as a solid object, we decided to model it as a growing frontier. Our system allows the user to create complicated boundaries procedurally in 2 dimensions. Our algorithm is an example for modeling irregular natural objects. In the future, we hope to extend the technique to capture other irregular natural objects as well.

There are various applications for which our modeling results can be used. We envision a scenario in which a user may wish to add irregular complexities to an existing model or combining our results with objects in a photo. Multiple irregular shapes generated by our method can also be laid on top of one another and rendered manually or automatically to obtain realistic or artistic looking shapes resembling 3 dimensional clouds or similar bodies. We envision our results to be used in vector images too. A user may wish to use our algorithm to create realistic looking clouds or similar bodies in a vector image.

1.3 Snaxels

Active contours, also called snakes, are interconnected particles. The particles of a snake are linked together, forming a moving chain. The snake does not have any self-intersection. Snakes are a framework used often in computer-vision applications to move inside images to find object boundaries. Snakes have been expanded to computer graphics [4] and are configured to propagate through a tessellation. This allows the snake to move in the space of the tessellation with having a means to detect self-collision and maintaining the density of the particles on the snake. These two issues are handled by the framework inherently and therefore, allow us to only focus on the modeling aspect.

We will now explain snakes in more detail. Kass et al. [30] introduced an energy minimizing, deformable spline which can be influenced by constraints to pull it towards image contours. Bischoff et al. [4] introduced a new method for representing and evolving snakes that are constrained to move on a triangular tessellation. The new representation adapts the snake resolution to the surface tessellation automatically with efficient collision detection and complete control over the topological behavior of the snakes.

The term *snaxel* was first used by Sobottka and Pitas [54], as a combination of “snake” and “element”. The term is used for the particles (or vertices) of an active contour model. In our work we assign the snaxels a speed. We run a simulation to allow them to grow on the tessellation. In chapter 3 we will discuss how the speed is assigned to the snaxels.

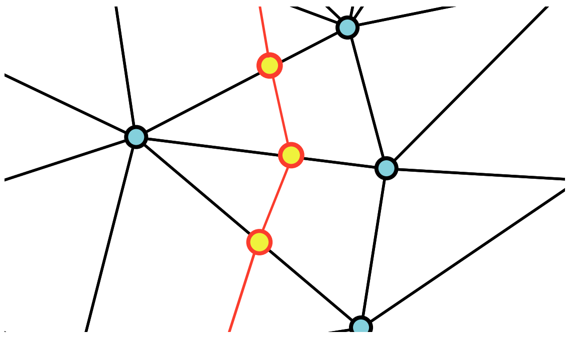


Figure 1.2 Snaxels moving on a tessellation;. Red lines indicate segments of the snake. Yellow vertices are snaxels and the blue ones are tessellation’s vertices.

1.4 Statement of Contribution

This thesis contributes to the area of procedural modeling in computer graphics. The primary objective of this thesis is to use snaxels for procedural modeling in 2 dimensions and it targets cumulus clouds in 2 dimensions as an example. Our contribution is a modeling technique to procedurally synthesize complex natural boundaries mimicking cumulus clouds’ shape and surface.

1.5 Overview

The thesis is organized as follows:

In chapter 2, we review previous work and the work that has been done in the field of automatic modeling. Our work is a procedural/sketch-based modeling system for creating cumulus looking bodies through an active contour model which propagates through a tessellation. Therefore, we first survey procedural modeling of natural phenomena and sketch-based modeling systems where we will discuss the different approaches to model such phenomena. Also simulation based systems and noise generation will be discussed and we will explain why such approaches were not considered for this system. Then we proceed to discuss previous work on curve synthesis.

In subsequent chapters, we describe in detail the modeling system that we have developed. In chapter 3 we will present our algorithm. First, we begin by discussing snaxels' behavior more in depth, then we present a plan for dividing the snaxel contour into sub-contours dubbed Curves. Later in chapter 3, we discuss our methodology of assigning speed to each snaxel using our plan to subdivide the snaxel contour as we discussed in the previous chapter. Finally, we will discuss our plans to grow the snaxels and produce cumulus looking structures.

In chapter 4, we will present our results and discuss their variations and comparisons. Next, in chapter 5, we will discuss future works and extending snaxels from 2D to 3D, and we will discuss what we have done so far in regards to that. Chapter 6 concludes the thesis. Finally, appendix A will discuss the efforts we made to extend snaxels into 3D space with discussion on issues we ran into. We will also describe the problems we are currently facing for future research into this area.

Chapter 2

Background

2.1 Procedural Modeling of Natural Irregular Objects

Modeling irregular objects and surfaces via user input alone is cumbersome and would require long hours if not days to complete. Procedural Modeling eases the process by generating the results through rules and user input values. Procedural Modeling has been used to model different phenomena such as clouds, rocks, and terrain. In this section we will discuss some of these phenomena which are related to our work.

2.1.1 Clouds

A lot has been done on procedural modeling of clouds, Neyret [43] combines some of the effects of fluid mechanics and thermodynamics as characterized by atmospheric physicists, to model cloud creation and dynamics. In this model, a cloud is represented by bubbles. The bubbles' behavior is guided by physical characterization of the evolution of the shape of convective clouds. The shape of the cloud is represented by structures smaller than the bubbles themselves. The bubbles have a recursive substructure. As we will discuss in the upcoming chapters, this representation motivated our work. Similar methods have also been investigated which involve complete or partial usage of fluid and cloud dynamics [25, 41, 66]. Trembilski and Broßler [63] present surface-based methods for visualization of mainly cumulus clouds. In their approach, in order to model the clouds' shape, they use a modified version of Neyret's method [43]. They produce a set of hemispheres to describe the surface of the cloud.

Their methodology is limited in the kinds of shapes that it can represent. Bouthors and Neyret [6] proposed a model for representing the shape of cumulus clouds. Their method stores a hierarchy of spherical particles placed on top of each other. An implicit field defines the shape of these particles by the influence of other particles. They use a set of shaders to create a volumetric appearance based on Gardner's textured ellipsoids [20]. Similarly, this representation of the clouds' surface does not create an irregular surface as all particles are spherical. Volumetric representations are widely used for modeling clouds, since these representations are essentially density fields. Miyazaki et al. [41] proposed a method for modeling clouds based on an extended method of cellular automaton. They developed an interactive system for modeling various types of clouds. Their modeling process is a simulation of cloud formation and the computational cost of the simulation is inexpensive relative to more physically accurate simulations due to using a simplified numerical model. The rendering of their results is realistic as they are obtaining a three-dimensional density distribution. Others have investigated modeling clouds as three dimensional densities and volumes [15, 51]. This type of representation is computationally expensive. For modeling opaque clouds, more simplified representations would suffice.

There is also a great body of work on rendering of clouds [5, 7, 8, 12, 13, 16, 20, 22, 26, 33, 38, 44–46, 61]. However, we are not focused on the rendering since it is mostly related to shading and propagation of light in the body of a cloud. These issues, although fascinating, are not the problems we address here. Our focus is on the shape and modeling of the surface of a cumulus cloud.

2.1.2 Rocks

Modeling of rocks has not been researched as much as clouds or even terrain. Research in this area is relatively new. Even on a large scale, few works have been trying to capture rock formations such as arches or tors and any other column shaped structure which is almost vertical or possesses cavities. These shapes are particularly hard to model with standard practices such as height map editing. They are relevant to our work since their overall shapes are related to the ones we want to model.

Dorsey et al. [14] model weathered stone by using a slab data structure which is a surface-aligned volume placed around a narrow region of the boundary of the stone.

They simulate the flow of moisture and recrystallization of minerals near the surface. Their model governs the surface erosion of the stone. This representation models the underlying structure of real stone. Söderlund [55] investigated two different approaches to procedurally model rocks' shapes and generate rocks and boulders. The first method is sphere inflation which is tessellating a sphere through recursive subdivision on a base shape. This approach is successful in creating results with similar characteristics to eroded rocks with a fair variety. The other method is recursive subdivision of edge segments but ultimately was unsuccessful.

Some work has been done on rock textures. Souli et al. [59] propose a method based on mimicking natural phenomena by representing the microscopic granular structure of a target model. Their study focuses on granite as a heterogeneous granular material. They first start by choosing the grain positions in space. Then by using a simple rendering process taking into account each material component and subsurface scattering, they generate a granite texture. Their model can generate natural looking granite with a complex surface. Unfortunately, the system is computationally costly.

There are also a few works on complex rock formations such as arches and sandstone goblins. Beardall et al. [3] present an algorithm for generating sandstone goblins by simulating spheroidal weathering. Their weathering simulation uses bubbles centered on axis aligned voxels. This way they approximate geometry-dependent effects of spheroidal weathering. Their work is a step toward the automatic generation of concave landscape features. Peytavie et al. [49] present a framework for representation of complex terrains features such as overhangs, arches and caves, and with different materials such as sand and rocks. Their hybrid model combines a volumetric discrete data structure and an implicit representation for sculpting and reconstructing the surface of the terrain. Their system allows scenes to be edited and sculpted interactively. Their work allows for representation of arches and overhangs through simulating an erosion process and stabilization simulation.

2.1.3 Terrain

There is an extensive body of work on terrain modeling and synthesis. Smelik et al. [53] survey different procedural methods for terrain modeling. They present multiple different techniques used to generate height fields for the basis of a terrain model.

One early technique is midpoint displacement method [40]. It iteratively subdivides a course height field to obtain a finer one. A more common technique is fractal noise [19, 42]. Fournier et al. [19] applied Mandelbrot's fractional Brownian motion model for terrain and other natural phenomena. They showed several methods for creating stochastic modeling primitives of one and two dimensions and demonstrated the use of stochastic interpolation of real sampled data to create realistic representations of the sampled phenomena. Physics-based algorithms are also a good way for modifying the terrain so it would look more natural. Musgrave et al. [42] introduced hydraulic erosion. In their method, a terrain's surface is gradually modified by allowing water on the surface of the model to carry sediments to lower heights. Rusnell et al. [50] presented a terrain synthesis method to generate a variety of features with simple user control over the shape and location of features. Their synthesis method is based on distances in a weighted graph. Their method provides user control over both terrain feature placement and terrain style. It also employs path planning to produce a height field. Using the weighted graph, a final height field is produced. However, height fields are not suitable for structures such as overhangs and caves. There are other works that use height fields for modeling terrain as well [11, 27, 60].

Example-based techniques have also been investigated. Zhou et al. [67] presented an example-based terrain synthesis system. Their system would take patches from height field represented sample terrain and use it to generate new terrain model. The extracted patches are joined together using graph cuts and Poisson editing. Guérin et al. [24] propose a framework based on a description of the terrain generation process by sparsely combining a small set of terrain dictionary atoms. It blends the different land-form features stored in a dictionary. They also demonstrate their method in several applications such as inverse procedural modeling of terrains, terrain amplification and synthesis from a coarse sketch. Brosz et al. [9] presented an example based terrain synthesis technique. They use multi-resolution analysis to obtain and extract small-scale features from the example to the base terrain's large-scale features. Their interactive system copies detail from the target to the base in a predictable manner. However, similar to any example-based approach, the quality and variety of the final output depends on the input examples. Also their interactive method requires more input from the user than the automatic one.

2.1.4 Cities

There has also been an extensive body of work on procedural modeling of cities and buildings [17,23,31,47]. Since cities are among man-made phenomena, the techniques used mimic the grid-like patterns of urban areas and the real road networks. Similarly, the results will include these patterns and grids, for example Manhattan-style grids. As a result, these methods are not suitable for modeling irregular natural phenomena where we expect the result not to take such obvious grid-like structures or pattern which the eye can pick up quickly.

2.2 Sketch-Based Modeling

A lot of work has been done on sketch-based systems for modeling purposes. Cook and Agah [10] provide a discussion on sketch-based modeling as well as survey existing techniques. Inflation allows the user to draw an object and have it pop into 3 dimensions. For example, Stiver et al. [62] proposed a freehand sketching system to control the modeling of volumetric clouds. They use sketch analysis and the elevation at which the cloud is drawn to identify the cloud type and then generate a mesh. The mesh is created using a rotational blending surface. The user can edit the cloud volume by addition and removal of material through Boolean operations. The cloud is rendered through filling the cloud mesh with particles. Wither et al. [64] describe a sketch based interface for modeling cumulus clouds. Their system allows the user to rapidly construct a 3D cloud surface representation. The system automatically creates a skeleton from the user's strokes and places spheres along the skeleton. The final mesh is obtained from the union of the spheres. This representation of the clouds' surface does not create an irregular and detailed surface.

Paint-like methods have also been investigated. Lawrence et al. [36] investigated combining interactive manipulation and physical simulation to control surface deformations. They created a painting interface so a user can apply “paint” to the model which defines its instantaneous surface velocity. They discovered that their painting metaphor gives the user direct, local control over surface deformations for applications such as: creation of new models, noise removal, and adding geometric texture at multiple scales. There are two issues that is still an open problem, self-collision and surface point density.

2.3 Noise

For modeling natural phenomena, noise is one of the most used techniques in computer graphics. Noise synthesizing algorithms have several advantages. They are fast and most require minimal coding and are simple to implement. They provide the ability to add complex and intricate details at low memory cost. There are many types of noise functions. Some of the most used ones in the procedural modeling are listed below.

2.3.1 Perlin Noise

Perlin [48] introduced a noise function primitive which utilizes an integer lattice to model texture. The value at each pixel on the image surface is actually an evaluation of an interpolated linear equation on the lattice. Perlin showcases different textures which were modeled by evaluating the value returned by the noise function, at each pixel. Moreover, by layering Perlin noise, Perlin turbulence can be created where instances of Perlin noise are added together. Perlin noise is widely in use, both in research and industry.

2.3.2 Gabor Noise

Lagae et al. [34] introduce sparse convolution noise with a Gabor kernel. The Gabor kernel provides support over the frequency domain and the spatial domain thus, enabling accurate control over the power spectrum. Gabor noise is procedural and fast to evaluate. It offers accurate spectral control with parameters such as orientation, principal frequency and bandwidth. It is non-periodic and anisotropic.

2.3.3 Worley Noise

Worley [65] presented a noise function by partitioning of space into a random array of cells. The idea is to scatter random points in the space, then the value of each point in space is its distance to the nth-closest randomly scattered point. The choice of ‘n’ alters the resulting image. The result looks similar to Voronoi cells.

2.3.4 Stochastic Modeling

Fournier et al. [19] used fractional Brownian motion and multiple other fractional based functions and made algorithms to generate irregular surfaces and curves. They introduce a new algorithm that computes a realistic approximation to fractional Brownian motion which is faster than with exact calculations. The advantage of this technique is that it allows for computation of the surface at arbitrary levels of details without increasing the database. Therefore, objects with complex surfaces can be displayed using a very small database. Although it generates irregularities and details at any level, it lacks control over the result. Therefore, techniques which can have more control on the shape and features of the results are needed.

2.4 Curve Synthesis

There is a body of work on synthesizing and modeling curves. Curve synthesis has been a topic in the area of computer aided geometric design (CAGD) for a long time. The goal is to generate curves through a certain parameterization. We focus on the techniques which are relevant to our work. Lang and Alexa [35] propose a method for on-line synthesis of free-hand drawing styles along arbitrary base paths. They use an autoregressive Markov Model which learns the input's style and applies it to an input curve. They allow addition of complexity to a curve by their proposed method which uses a supplementary Markov chain to guide the synthesizing process. Their method is example-based and needs the input model's style to have some degree of repetitiveness, otherwise it will result in noisy results. This would mean that to obtain a variety of styles on an input curve, a tedious amount of work need to be put into the creation of them. Therefore, for modeling natural irregular surfaces, this approach will introduce regularities to the model. Merrell and Manocha [39] present a synthesis algorithm for procedurally generating complex curves. Their method requires an input example. Their method can preserve many of the example's features such as tangent directions, curvature, branch nodes, and closed loops. Their algorithm has application to generating complex, curved models of man-made objects. We are focused on natural objects and phenomena, and they possess highly irregular and complicated surfaces. As discussed, similar to any example-based approach, the quality and variety of the final output depends on the input examples.

There has been a lot of effort put into curve representation and parameterization [52]. However, we are concerned with the procedural modeling aspect. We chose snakes (active contours) due to two characterizing properties that they possess. Their handling of self-collision and their control over the resulting contour's resolution.

Chapter 3

Algorithm

3.1 Introduction

In this section, we are going to give a full description of our algorithm. First, we start by giving a full description of snaxels' behavior and organization. Later, we will describe the structures that we build using snaxels to allow modifications to their behavior as a group. Finally, we will present our algorithm by describing our plans for growing the snake, on the tessellation.

Our algorithm starts with Poisson-disk distribution in 2 dimensions and Delaunay Triangulation to obtain the tessellation. Once the tessellation is ready, the process starts with a chain of snaxels set around an arbitrary vertex of the tessellation with that vertex set as the snaxels' source vertex. A propagation method repeatedly updates each snaxel. The process finishes at either a user input criteria to stop all snaxels or the user manually stops the growth once satisfied with the results. Figure 3.1 shows the snake after a few steps have been executed. The *snake* is marked by dashed-lines.

Cumulus clouds' structure and surface appears hierarchical. We need to mimic the bulges and bumps that appear on top of one another. We segment the snake into smaller chains of snaxels. We call each smaller chain a *Curve*. We capture the bulges and bumps of a cloud by growing the curves during the simulation with different speeds. The curves pass through a life cycle which controls how they grow. The life cycle depends on how much distance curves have traveled and controls their growth using their traveled distance. However, curves alone are not enough to obtain a varied and complex shape. Curves are further organized into *Clusters*, so that groups of curves can be given similar parameters. The clusters give us another explicit level of hierarchy, helping us model loosely hierarchical structures such as clouds.

We came up with strategies for guiding the growth of the snake. All of them manipulate the snaxels' speeds. We will discuss these strategies in the coming sections. First, we start by describing the snaxels' behavior and constraints.

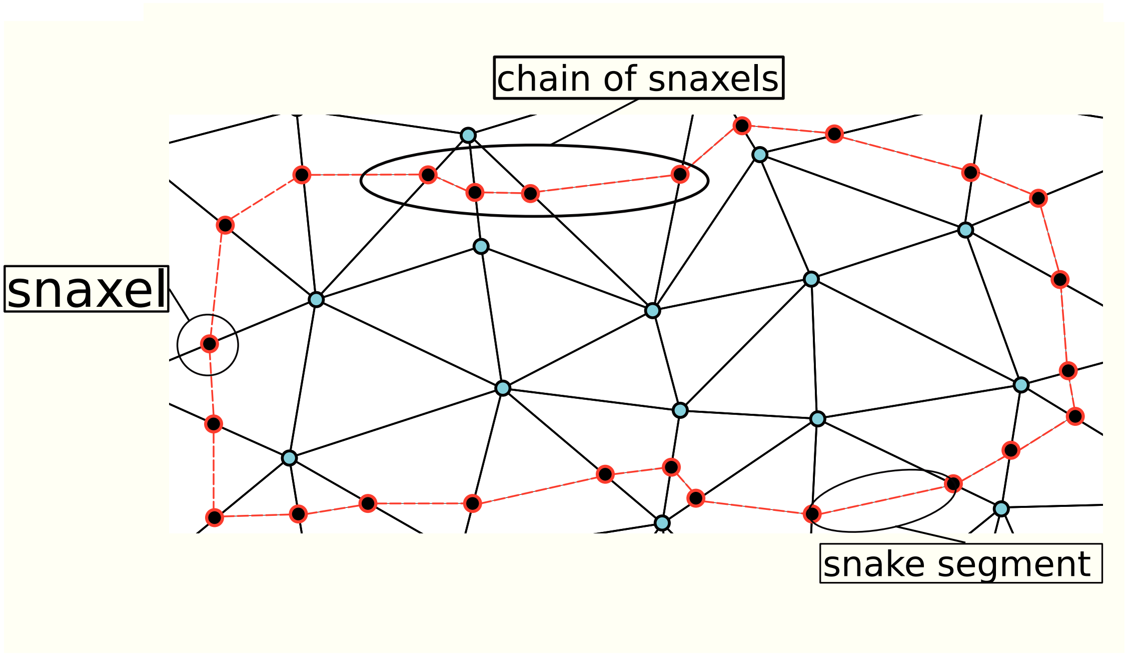


Figure 3.1 The snake propagating through the tessellation. Dashed lines indicate the snake. Black vertices are snaxels and blue ones are tessellation's vertices.

3.2 Snaxels

In order for the snake to propagate through the tessellation, certain constraints are needed. These constraints will define the basis to form the rules and behavior of the snaxels. Similar to Bischoff et al. [4]'s method, our snaxels have the following constraints:

- For our implementation's simplicity, only one snake can exist in the system.
- Snaxels move on tessellation edges.

- Each snake's segment can only connect snaxels which share a triangle.
- Snaxels are connected to one another forming a chain.
- For our implementation's simplicity, the snake is allowed to pass through each tessellation's vertex only once. Once a snaxel passes through a tessellation vertex, no other snaxel is allowed to pass through the same vertex.

As the snake propagates through the tessellation, the snaxels collide with each other and the tessellation vertices. These events need to be handled properly for the constraints to be always met. Therefore, Bischoff et al. [4] defined snaxels' behaviors as evolution and collision detection of the snaxels. Similar to them, we define the following behaviors:

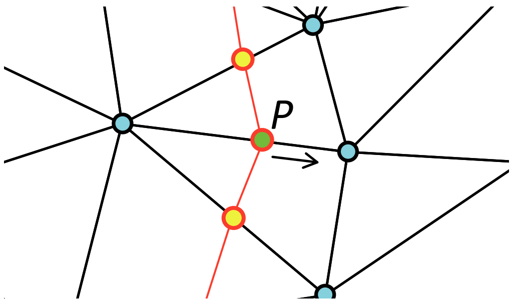
- Birth: A snaxel gives birth to new snaxels upon arrival at its destination tessellation vertex.
- Merge: Two snaxels merge with one another after colliding on a tessellation edge which results in both of their deaths and updating the snake.
- Growth: A snaxel grows along a tessellation edge from its birthing tessellation vertex headed towards its destination vertex.
- Death: A snaxel is completely removed from the system.
- Stop: A snaxel's growth rate is set to zero.

Using all these behaviors we can construct a state machine for each snaxel. Each state is one of these behaviors. In the following sub-sections, we will explain the events that take place at each described behavior.

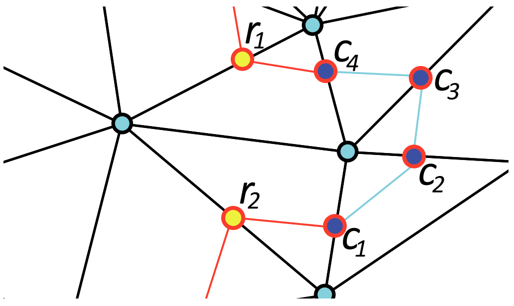
3.2.1 Birth

The snaxel arriving at its destination vertex is called *parent* and the edge by which it arrived at the destination vertex is called *in-going edge*. Once a snaxel reaches its destination vertex, we remove it from the snake. Removing the parent from the snake creates a gap in the snake's chain of snaxels. The snaxels connected to the parent prior to its removal are called *rim snaxels*. We put new snaxels on every edge of the destination vertex except for the in-going edge. The new snaxels are called *children*. The birth event is handled by replacing parent's position in the snake with the children. The children form a small chain of snaxels. We connect the ends of the children's chain to the rim snaxels such that the snake's chain does not self intersect. This maintains the constraint that no snake's segment can cross tessellation edges and allows the snake to continue growing.

As shown in Figure 3.2(a), the parent denoted 'P' is moving toward its destination vertex. Once it arrives at its destination vertex, it is discarded and its connections with rim snaxels are removed. After the removal of parent and its connections, each child is placed and set to grow on each edge of the destination vertex except the in-going edge, as shown in Figure 3.2(b). Once the children are created, they will be sorted by the angle of the edges they lie on with the angle of the edge which the parent came from. Then, each child in the sorted list is connected to the next one, starting with the first and ending with the last in the list. The children are sorted so that once they replace the parent's position in the snake, the snake's consistency is kept. Once the children are connected and formed a chain of snaxels, we check to see which one of the rim snaxels is on the same triangle as the first or the last child in the list. We associate each of the rim snaxels with the first and the last children based on their adjacency on the same triangle. Once they are associated, we connect them. For example, as shown in Figure 3.2(b), rim snaxel 'r_1' resides on the same triangle as 'c_4' and 'r_2' resides on the same triangle as 'c_1'. They are connected to one another thus, completing the snake's chain.



(a) Before the birth event



(b) After the birth event

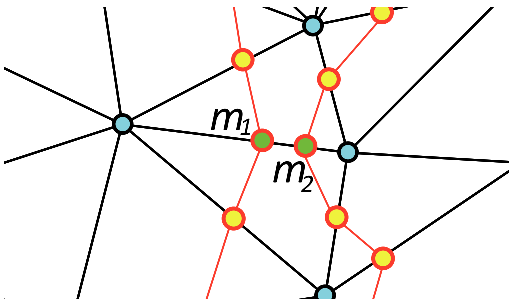
Figure 3.2 Before and after a birth event. (a) Snaxel P is about to arrive at its destination. (b) After snaxel P arrives, it is deleted and the birth event takes place which results in new snaxels being born and added to the snake.

3.2.2 Merge

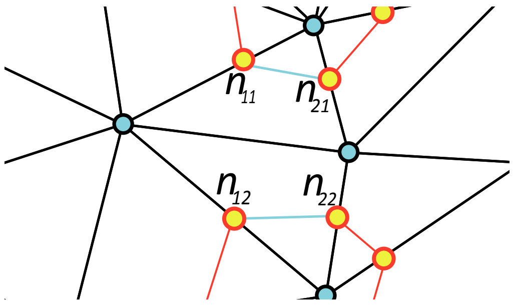
Snaxels may hit one another while growing through the tessellation's edges. In this case the snaxels involved in the collision are set to merge with one another. During this event, the state of both snaxels involved is set to the death state and their connections to their neighboring snaxels are removed. The snaxels involved are called merging snaxels and their neighboring snaxels are called neighbors. The neighbors of the merging snaxels are connected to one another to maintain the connectivity of the chain of snaxels on the snake. We will describe how the connectivity is maintained.

Similar to the birth event, we use the triangles shared between the neighbors of the merging snaxels to maintain the snake's chain's connectivity and consistency. However, if the merging snaxels are connected to one another, then this is an indication that the snake has passed through both vertices of the edge on which the merge is happening. Therefore, we can remove the merging snaxels and connect the neighbors. Then the snake's chain's connectivity is maintained. Also if the merging snaxels are only connected to each other, then they can be removed and the snake does not need any updating. If the merging snaxels are not connected, then we use the triangles shared between them to maintain the snake's chain's connectivity.

Figure 3.3 shows before and after a merge event. As shown in Figure 3.3(a), two snaxels called 'm_1' and 'm_2' are about to collide with one another. The neighbors of 'm_1' are called 'n_{11}' and 'n_{12}' and the neighbors of 'm_2' are called 'n_{21}' and 'n_{22}'. We find which neighbor of 'm_1' resides on the same triangle as which neighbor of 'm_2'. Then we connect those who share their triangle with one another. For example in Figure 3.3(a), 'n_{11}' shares the triangle it resides on with 'n_{21}' and 'n_{12}' shares its triangle with 'n_{22}'. Therefore, 'n_{11}' is connected to 'n_{21}' and 'n_{12}' is connected to 'n_{22}'.



(a) Before the merge event



(b) After the merge event

Figure 3.3 Before and after a merge event. (a) Snaxels m_1 and m_2 are about to merge with one another. (b) The snaxels are deleted and their neighbors connected to one another.

After a birth event or a merge event, we never connect the snaxels in such a way that would result in a snaxel with both of its connections on a single triangle. This kind of connection is illustrated in Figure 3.4. The Figure 3.4 shows snaxel 's' with both of its connections inside a single triangle. If we never connect snaxels in the way shown in Figure 3.4, then it is safe to assume we will never encounter such a connection before a merge or birth event. The only connections which are produced by our handling of birth and merge event are depicted in Figure 3.5. Figure 3.5 shows the ways snaxel 's' can be connected to its neighbors. A snaxel can have each of its connections in a separate triangle as depicted in Figure 3.5(a) or have one of them on the edge that it is growing on as shown in Figure 3.5(b).

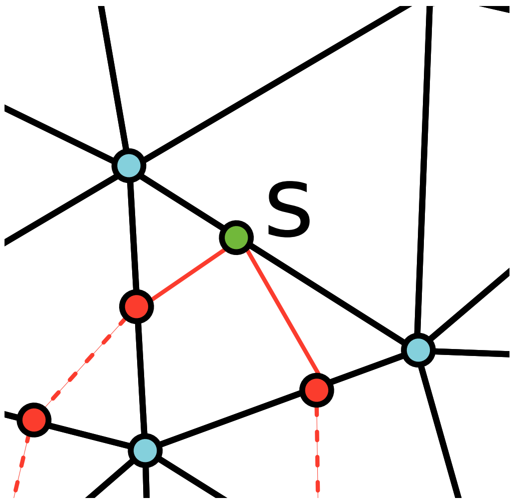


Figure 3.4 A snaxel with both of its connections on a single triangle. Note that this situation is impossible.

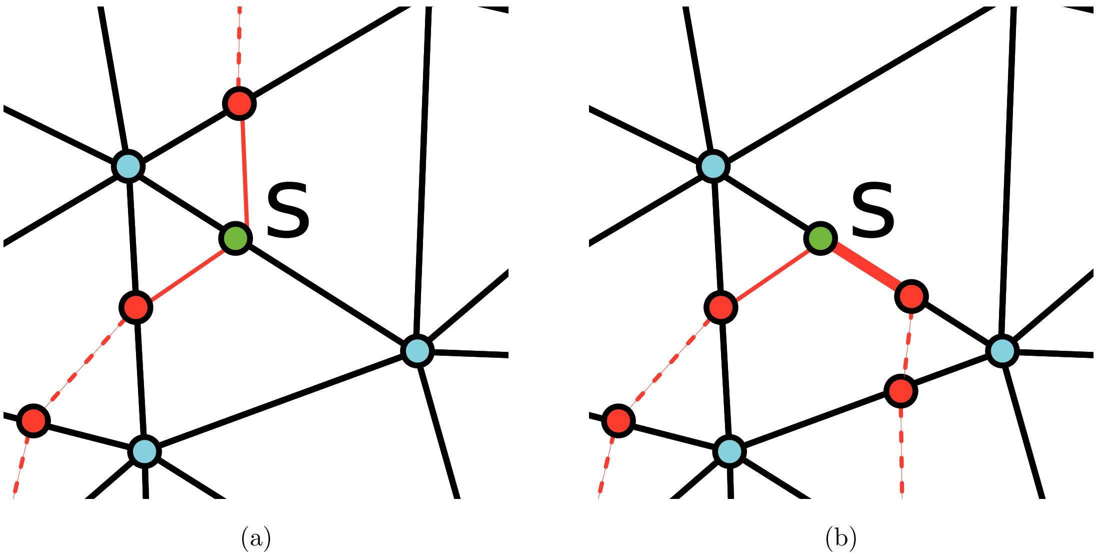


Figure 3.5 Types of connections between a snaxel and its neighbors. (a) Snaxels S has each of its connections on a separate triangle. (b) Snaxels S has one of its connections on the edge it is growing on.

3.2.3 Growth

Snaxels are set to grow right after their birth. It is within this state they are assigned a speed. The processes to calculate the speed are discussed in section

3.4. Once the speed is calculated, the location of the snaxel is updated by Euler integration:

$$I_x(t + \Delta t) = I_x(t) + v_s * u_e \Delta t$$

where $I_x(t)$ stands for the location of the snaxel at time step t, v_s stands for the speed of the snaxel, u_e stands for the normalized vector of the edge that the snaxels lies on in the direction of its growth and Δt stands for the elapsed time since the last update.

3.2.4 Death

Once a snaxel has merged with another one, it is set to die. Death of a snaxel results in the snaxel being removed from the system. In our algorithm death happens after the merge between two snaxels and once a snaxel reaches its destination.

3.2.5 Stop

Any snaxel can be stopped at any point during the simulation. This behavior can be triggered by imposing a criteria so that it will stop two different bodies from merging.

3.3 Curves

In the previous section, we discussed snaxels and their behaviors and constraints. In this section we are going to introduce a structure called a *Curve*. A curve is a sub-chain of the snake. As shown in Figure 3.6, a chain of snaxels forms a curve. The curves are not allowed to overlap. We subdivide the snake into multiple segments so that we can assign different speeds to each segment and thus model a complex shape by varying the speed of different curves. Figure 3.6 shows an example of a curve. A curve is illustrated which starts from and including snaxel 'S_0' to and ending with snaxel 'S_n'. All the snaxels in-between are included in the curve.

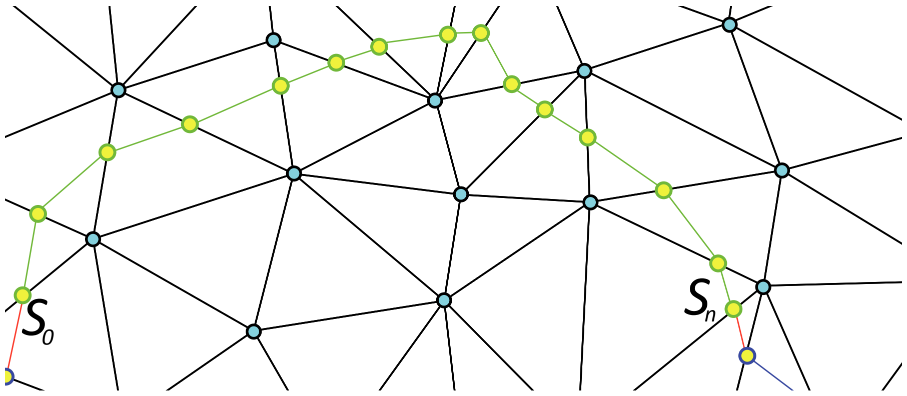


Figure 3.6 Example of a curve. Snaxels S_0 to S_n form a curve.

3.3.1 Updating Curves

As the snake propagates through the tessellation, the curves will need to be updated depending on the events that take place. Since each curve is made out of a chain of snaxels, it is important to keep the chain's connectivity when a snaxel dies or gives birth to new snaxels. In this section we are going to discuss how the curve is updated in the face of the events that occur.

Creating New Curves

As snaxels of a curve grow on the tessellation, the curve itself keeps on growing in size. Sometimes we may need to create new curves because they get too long. To control when to break the curves we assign a length limit to each curve, such that when a curve reaches a certain length, it will break into new curves. The break can happen at any location on the curve. When curves grow, their snaxels give birth to new snaxels and the curve gets longer as a result. Curves can have different number of snaxels but the same length. We also pick a random snaxel on the curve to break the curve from. When a curve reaches a predetermined maximum length, it will break into two new curves. Figure 3.7 shows the result of this process: a long curve is split into to new curves. After the break, each of the joints of the previous curve becomes one of the joints of the new curves. For example snaxel S_0 and S_n which where the joints of the previous curve, each have become a joint of the new curves. S_0 is a joint of curve S_0 to S_i and S_n is a joint of curve S_j to S_n.

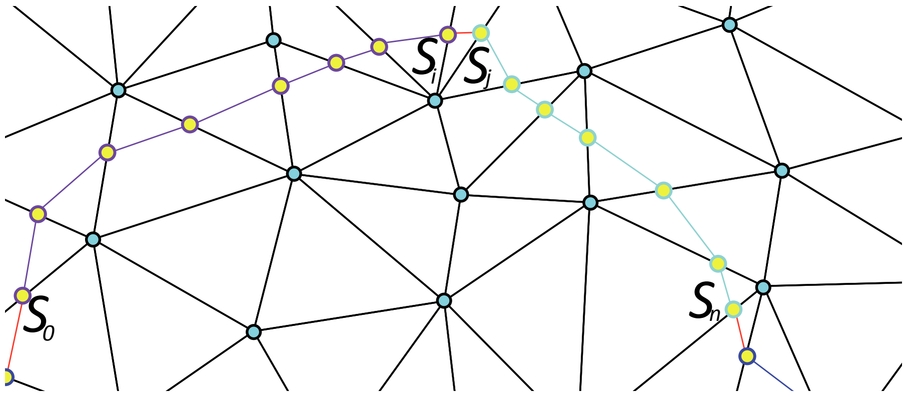


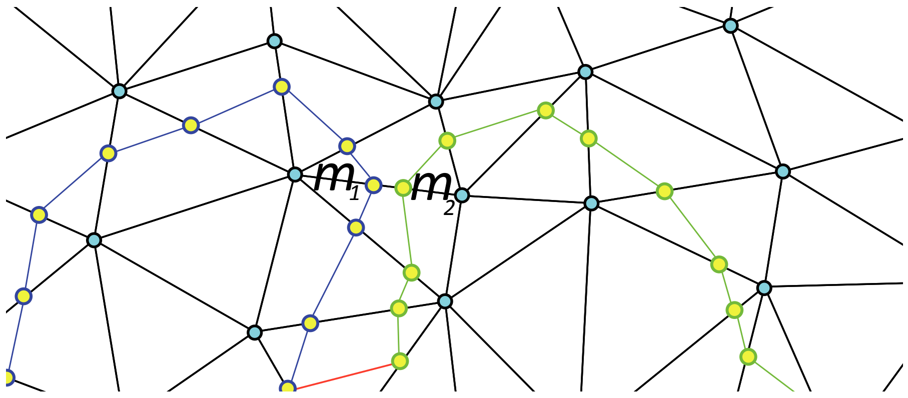
Figure 3.7 Example of breaking of a curve. The curve designated by snaxels S_0 to S_n, is divided into to new curves; one from S_0 to S_i and another from S_j to S_n.

Merging of Snaxels

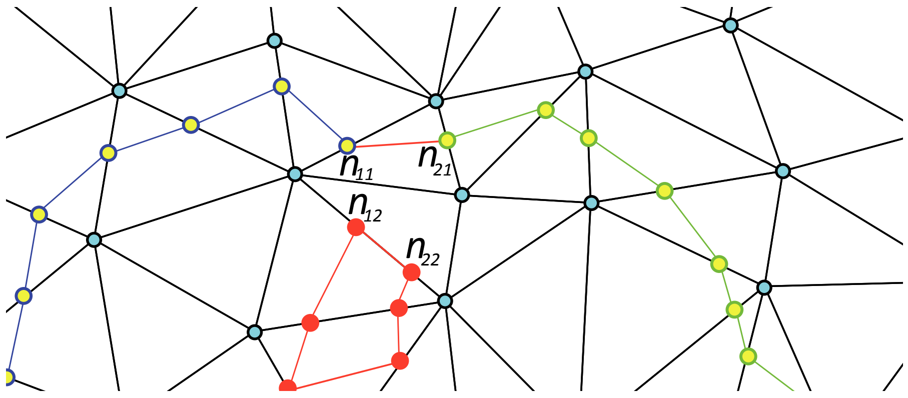
In our method, we use the right hand rule [18] to sort the snaxels inside each curve. This allows us to have a consistent chain and it makes the handling of events easier. As snaxels on the curve merge, the chain needs to be updated, especially at the joints where the head and tail of the curve gets updated.

When two snaxels which are connected to each other merge, they and their connections are deleted. To update the curve, we connect the neighbors as stated in the previous section to maintain the snake's chain's connectivity.

If the snaxels which are merging are not connected to one another, then two snakes are created. Since we are only growing one snake to obtain a model and snaxels only grow forward, then the snake never creates two separate snakes unless one is contained by the other. Therefore, out of the two snakes created from the merge, only one is the model which we are growing. Since one of them is contained by the other, we assume that the inner snake is shorter in length. However, this is a heuristic. In practice, we have not observed a case where the inner snake is longer than the outer one. The shorter one is the hole that was created by the merge event and is set to be discarded. Figure 3.8 shows a merge event happening between two snaxels 'm_1' and 'm_2' from two different curves. As shown, the snaxels involved in the merge event are removed from both curves and their neighbors are connected to one another as discussed in previous section.



(a) Before the merge event



(b) After the merge event

Figure 3.8 Before and after a merge event of two non-connected snaxels. (a) Snaxels m_1 and m_2 are about to merge with one another. (b) Their merge results in part of each of the curves they belong to being removed.

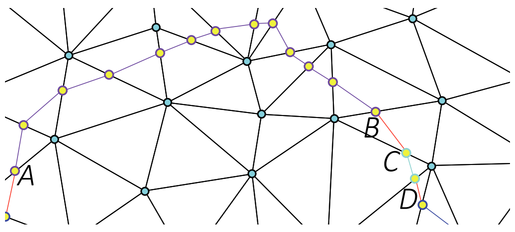
Detecting Holes

In Figure 3.8, the previous snake is split into two new ones: one containing the segment '$n_{11}n_{21}$' and the other containing the segment '$n_{12}n_{22}$'. This is an illustration of a self collision of the snake. Snaxels 'm_1' and 'm_2' on the snake are about to merge with one another,resulting in two new snakes. We need to be able to tell which resulting snake is the model which we are growing and which one is the hole. We need to be able to detect the holes which are generated as the snake propagates through the tessellation. After a merge event between two snaxels generates two snakes, we detect the outer snake which is our main snake by traversing both of them. We count the number of snaxels in each. After comparing the number of snaxels, we pick the one with more snaxels as the outer snake and the other is marked as a hole and discarded.

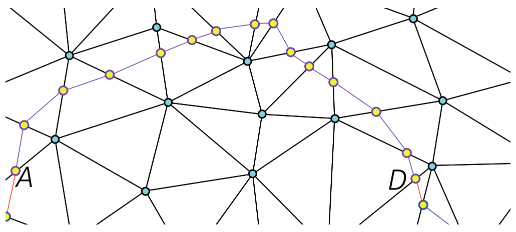
This mechanism for detecting holes inside the snake is fallible. The number of snaxels of the inner snake can be bigger than the outer one thus resulting the outer snake to be regarded as the hole. We assume that the inner snake is shorter in length. However, this is a heuristic and we are aware of mistaking the outer snake for a hole can occur. In practice, we have not observed a case where the inner snake is longer than the outer one.

Merging of Curves

Curves can become very small as their snaxels merge with snaxels of other curves. This reduction in number of snaxels will reduce the length of the curve. We decided to have a parameter called 'c_{min}' that allows a curve which its length is less than 'c_{min}', to merge with one of its neighbors. When a curve's length becomes smaller than the predetermined value, it will be merged with one of its two neighbors. Since the snaxels in each curve are kept sorted this process simply connects the remaining snaxels of the curve to either the end or the beginning of its neighbor's chain of snaxels. Figure 3.9 shows this process. A small curve starting from snaxel 'C' to 'D' has reached the minimum allowed length. Therefore, it is merged with its neighboring curve which starts from snaxel 'A' to 'B'.



(a) Before merging two curves



(b) After merging a smaller curve with one of its neighbor

Figure 3.9 Example of merging of a Curve with another. (a) A Small curve starting from snaxel C to D has reached the minimum allowed length. (b) The curve has merged with its neighboring curve starting from A to B, thus expanding the neighboring curve to expand its range to A to D.

3.4 Speed Scheme

In the previous sections, we discussed snaxels and how they propagate on the tessellation. We also discussed how snaxels are grouped together to form curves and discussed the events that result from snake's propagation on the tessellation. In this section, we will discuss how we manipulate the speed of each snaxel in a curve.

Cumulus clouds and volcanic ash-clouds and similar bodies, have details on their exterior visible at multiple different scales. As a result, the surface of these bodies looks segmented such that each segment has intricate details mimicking rounded segments on smaller scale. Each segment can be described as a bulge and the smaller details on them can be represented as smaller rounded smoothed bumps. At the growth state, we assign a speed to each snaxel. We are trying to mimic the overall shape of a cumulus cloud or a volcanic ash-cloud similar to Figure 3.10, such that each curve captures a bulge of the cloud.

One approach to assign speed to each snaxel is to interpolate the curve by setting the middle of the curve as the maximum input velocity and the corners at zero. This creates two issues which are illustrated in Figure 3.11. The first problem that can be observed is the immovable joints. The joints do not move and therefore throughout the process only get updated by the merging of their neighboring snaxels. This impacts the overall shape. The second problem is that the growth starts to become more round with little to no variations. We came up with plans to tackle these issues by changing the way a snaxel's speed gets assigned.

The speed is calculated in two steps. The sum of both steps is taken as the speed of the snaxels. The first step attempt to give the snake a consistent growth and the second step attempts to add details to the snake. In the first step, a *base* speed is calculated for each snaxel depending on the snaxel's position in the curve and the two neighboring curves. This step ensures that the corners move consistently with the rest of the object. In the second step, which is the details phase, we compute speeds based on a shape being imposed on the snaxel chain. By imposing a shape on the snaxels, we ensure that each curve can produce bulges and bumps resembling a real cumulus cloud or similar body. We will also describe the constraints on the type

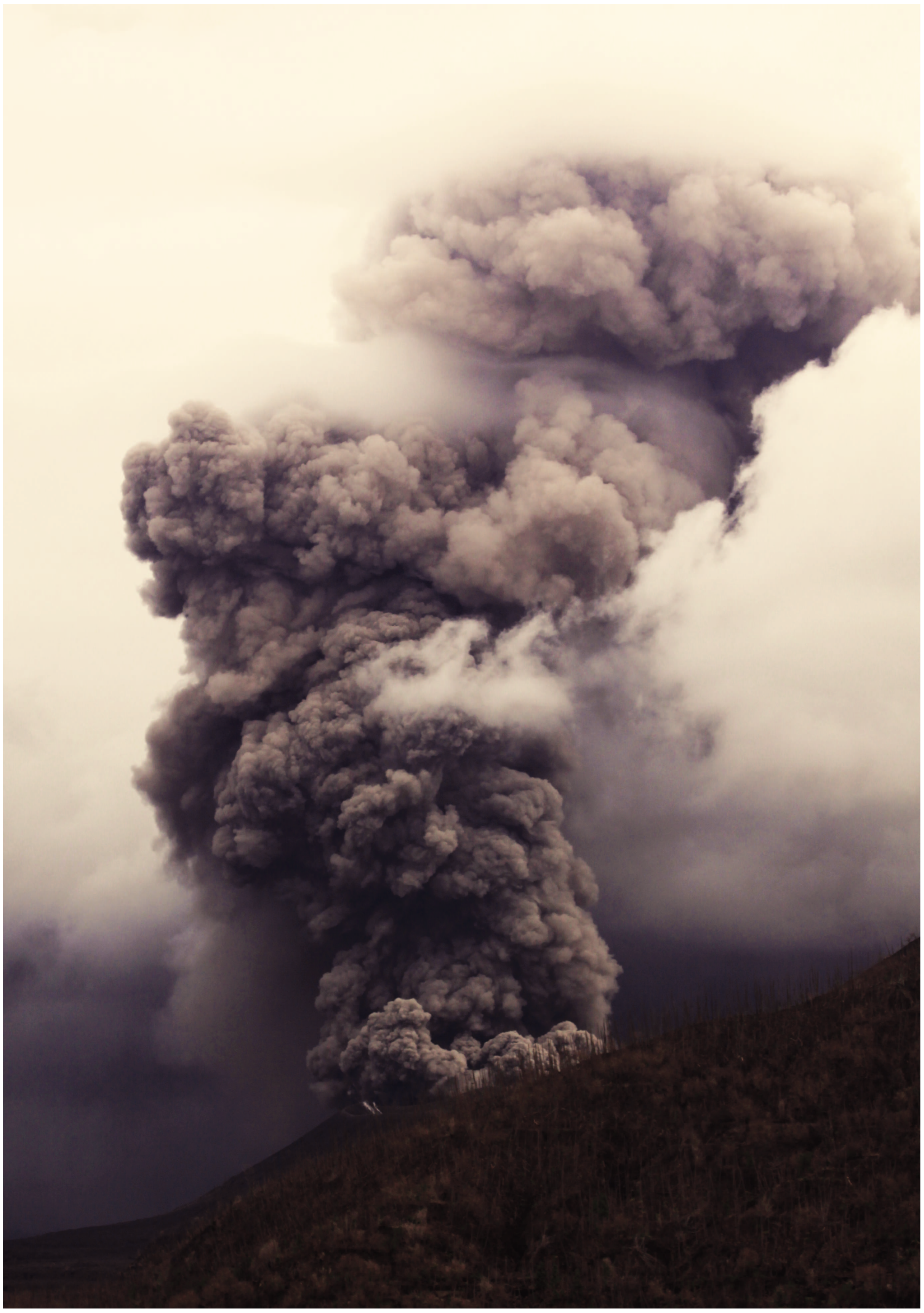


Figure 3.10 A Volcanic ash-cloud.

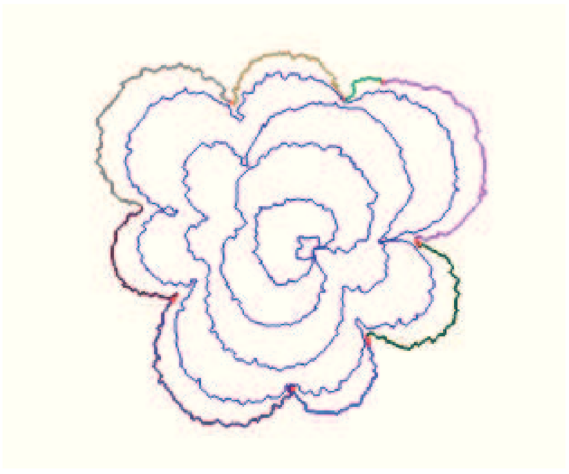


Figure 3.11 A result of linear interpolation over each curve. The linear interpolation is computed over the curve by setting the middle of the curve as the maximum input velocity and the corners at zero. Note that the inner contours are for illustrating and visualizing the growth process and that different curves are visualized with different colors for simplification.

of shape that is being used to be imposed on the snaxels.
In the next subsections, we are going to take a look at how each step is calculated to obtain a speed for a given snaxel.

3.4.1 Base Speed Scheme

We need to ensure the corners move consistently with the rest of the object. To achieve this we linearly interpolate the curve's snaxel chain such that all snaxels of the curve gain non-zero speed depending on their spatial distance on the curve from the mid-point of the curve and the distance from the mid-point of the neighboring curves. Each curve has its own joints, named 'j_{cx}', here 'c' stands for the the curve itself and 'x' stands for the neighboring curves. For instance Figure 3.12 shows a curve called 'B' with joints 'j_{BA}' and 'j_{BC}' connecting curve 'B' to its neighboring curves 'A' and 'C'.

Each curve is assigned a base speed value. The value is set by an input interval of $[v_{min}, v_{max}]$. The base speed values of the curves is needed to compute the base speed of each snaxel in that curve. For example, for each snaxel 's' on curve 'B' which has

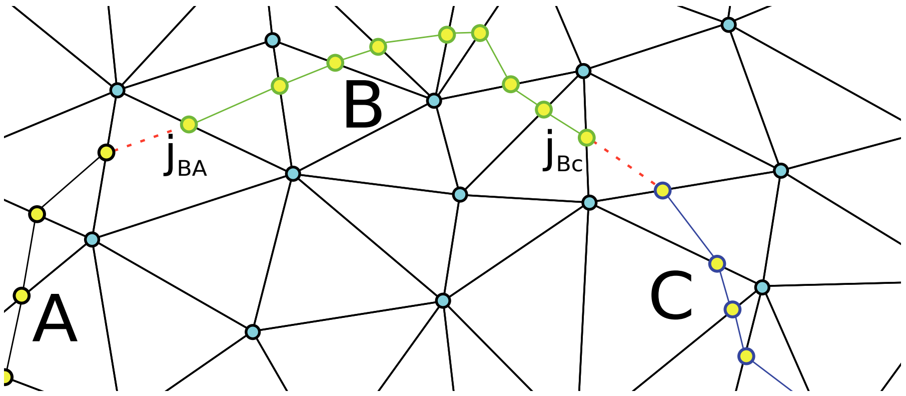


Figure 3.12 Curve ‘B’ is joint to curves ‘A’ and ‘C’ at snaxels ‘j_{BA}’ and ‘j_{BC}’ respectively.

2 neighboring curves called ‘A’ and ‘C’, we get the spatial distance from snaxel ‘s’ to the mid-points of the curves ‘B’, ‘A’ and ‘C’ called ‘p_b’, ‘p_a’ and ‘p_c’ respectively and use them to determine how much each curve contributes to the base speed of snaxel ‘s’. We compute these distances by taking the spatial distance from snaxel ‘s’ to the mid-point of curve ‘B’, ‘A’ and ‘C’ called ‘$d(s, p_b)$’,‘$d(s, p_a)$’ and ‘$d(s, p_c)$’ respectively. We also take the distances from snaxel ‘s’ to the joints of curve ‘B’ called ‘$d(s, j_{BA})$’ and ‘$d(s, j_{BC})$’ to determine which neighboring curve is closer to the snaxel ‘s’. The neighboring curve that is farther from snaxel ‘s’ does not contribute to its base speed. The distances are computed along the curve’s chain of snaxels. We take the distances from mid-point of curve ‘B’ to the mid-point of curve ‘A’ and ‘C’ called ‘$d(p_b, p_a)$’ and ‘$d(p_b, p_c)$’ respectively, and use them with the distances from ‘s’ to the mid-points of the curves and the joints of curve ‘B’ to compute the weights ‘w_b’, ‘w_a’ and ‘w_c’. The weights are computed depending on where snaxel ‘s’ lies on curve ‘B’:

$$w_b = \begin{cases} \frac{d(s,p_a)}{d(p_b,p_a)} & \text{if } d(s, j_{BC}) \geq d(s, j_{BA}) \\ \frac{d(s,p_c)}{d(p_b,p_c)} & \text{otherwise} \end{cases} \tag{3.1}$$

$$w_a = \begin{cases} \frac{d(s,p_b)}{d(p_b,p_a)} & \text{if } d(s, j_{BC}) \geq d(s, j_{BA}) \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$w_c = \begin{cases} 0 & \text{if } d(s, j_{BC}) \geq d(s, j_{BA}) \\ \frac{d(s,p_b)}{d(p_b,p_c)} & \text{otherwise} \end{cases} \tag{3.3}$$

Then we take the base speed value of 'B','A' and 'C' called 'v_B', 'v_A' and 'v_C' respectively. Using the weights for each of curve's base velocity, we compute a snaxel's base speed called 'b_v' by applying each of the weights computed above to each curve's base speed:

$$b_v = w_b \cdot v_B + w_a \cdot v_A + w_c \cdot v_C \tag{3.4}$$

where 'b_v' is the base speed of snaxel 's' on curve 'B'. The base speed is supposed to mimic a basic curved propagation through the tessellation at the beginning of the growth.

Figure 3.13 shows a result generated by computing the speed of the snaxels in two steps. First, the base speed described above is calculated and then the details phase is calculated as a linear interpolation of the snaxels' location over the curve's chain. Base speed causes the snake to grow in all directions. This is because the base speed guarantees none of the snaxels get zero speed value to grow the snake consistently. We need a method to gradually reduce the effect of the base speed so that we can grow the snake in any direction that we wish. To do this we use the weighted sum of both steps to compute the speed of each snaxel. Since the weight function needed to be designed in a way such that the influence of the first step slowly vanishes, we used a parameter to control the effect of base speed called the *effect coefficient* denoted 'k'. The effect coefficient is computed for each snaxel:

$$k = e^{-rt} \tag{3.5}$$

where 'r' is the influence rate set by the user and 't' is the traveled distance by the snaxel. The base speed is weighted by a predetermined user input parameter, 'w_s'. The effect coefficient is applied to 'w_s' to control it:

$$v_s = w_s \cdot k \cdot b_v + (1 - w_s \cdot k) \cdot l_v \tag{3.6}$$

where 'v_s' is the speed of the snaxel, 'b_v' is the basic speed and 'l_v' is the details phase. Figure 3.14 shows the result of this process. The inner contours illustrate the growth process with different curves visualized with different colors. Table 3.1 contains the parameter settings used to generate the figure.

So far, the details phase for a snaxel is the linear interpolation of the snaxel's location on the curve's chain and all the sample results presented have that in common. This way of setting the details phase's speed does not yield curves with varied shapes. We will replace the details phase with a new approach and discuss it more in depth in the next sub-section.

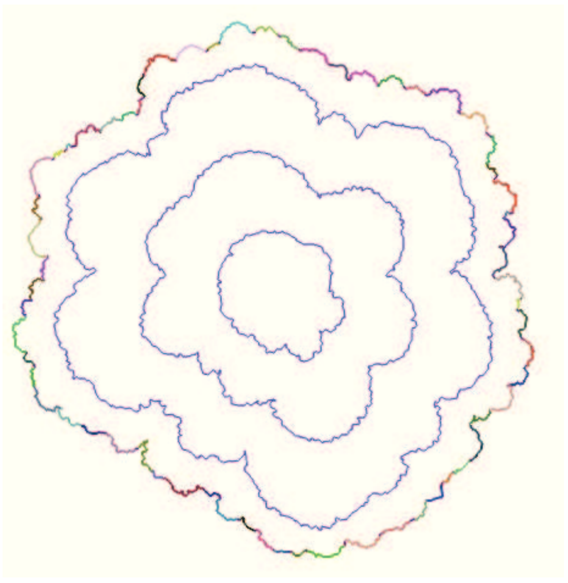


Figure 3.13 Result of computing each snaxels' speeds by adding a basic speed to the linear interpolation of the snaxels' locations in the curve.

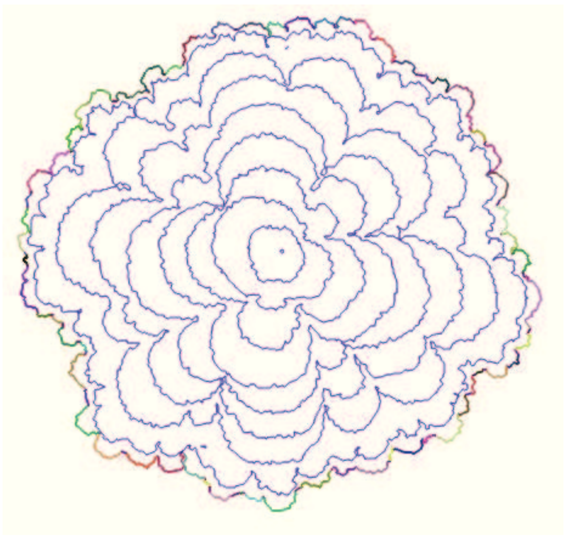


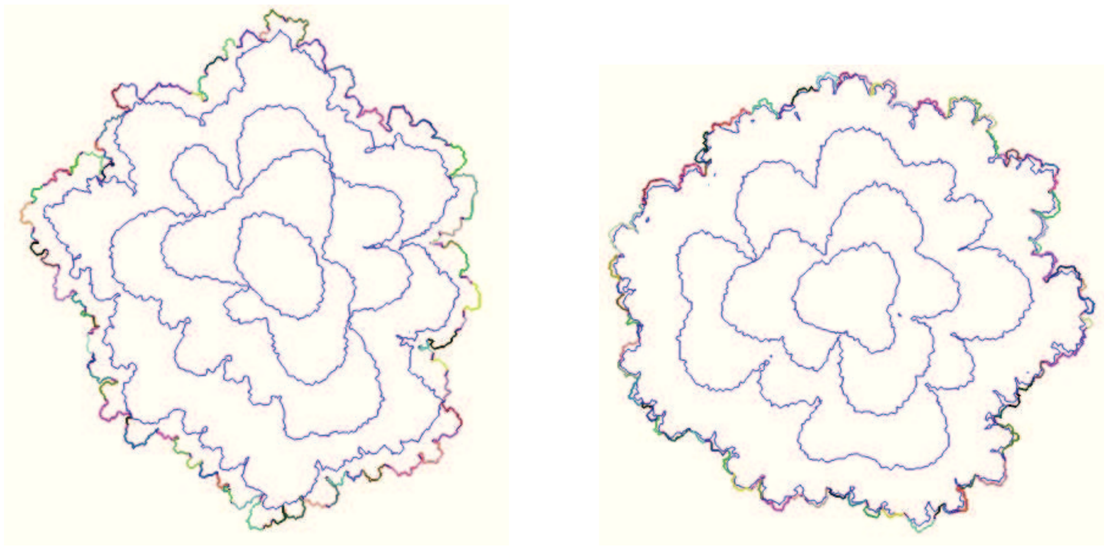
Figure 3.14 Result of controlling the base speed through an effect coefficient.

Figures	r	w_s	$[v_{min}, v_{max}]$	c_{min}
Figure 3.13	N/A	N/A	[5.0,10.0]	3.0
Figure 3.14	0.1	1.0	[5.0,10.0]	3.0

Table 3.1 Parameters used in the simulations

3.4.2 Circular Speed-Assigning Scheme

As mentioned previously, the details phase has been computed by the linear interpolation of the snaxels' locations on the chains. All the sample results presented above have that in common. The details phase needs to be computed in such a way that it can create a shape that is similar to a bulge. As mentioned above, we want to be able to produce bulges resembling cumulus clouds and similar bodies. We decided to impose a shape, for example a circle, on the growing curves. This will allow us to create bulges that better resemble a cloud shape. Figure 3.15 compares this method with a linear interpolation over the curve as discussed in previous section.



(a) Circular Scheme Result (b) Linear Interpolation Result

Figure 3.15 Comparison of Circular scheme method and linear interpolation over the curves. Both results were generated with the same values for each parameter shown in Table 3.2.

Similar to the calculation of base speed, we assign each curve a shape velocity. The shape velocity is the maximum speed a snaxel on that curve is allowed to have as its details phase velocity. We project the locations of the snaxels of a curve on a line by keeping their relative distances. See Figure 3.16 for an illustration of the scenario. We use the shape-velocity of curve 'c' as a vector dubbed '$\vec{v_c}$' to obtain a distribution of velocities across the curve's chain. We place the vector perpendicular to the line with the projected snaxels' locations. We use the shape-velocity vector to distribute velocities on the curve's chain such that the middle of the chain gets the shape velocity value and the joints of the curve's chain get zero.

We impose a circle onto the curve's chain by three points: shape-velocity vector's end and the two endpoints of the line holding the snaxels' relative projection. The circle serves as the distribution of velocities. Then, we take a fixed point at the bottom of the obtained circle. We create vectors which start at the fixed point and go through each projected snaxel's location on the line to intersect with the circle. Finally, we take a partial length of each vector starting at the projected snaxel's location and ending at the vector's intersection on the circle. This partial length of each of these vectors is the speed of the corresponding snaxel.

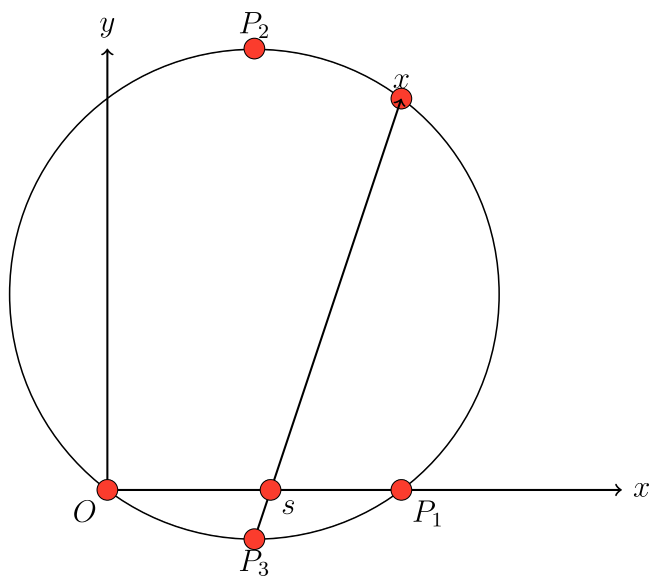


Figure 3.16 Circular Speed-Assigning Scheme Example. The velocity of snaxel 's' on the curve is set to the length of the vector '$\vec{sx}$'.

Figure 3.16 shows the process of acquiring snaxel speed using this scheme. The curve's snaxels' locations are projected on the line starting at the origin labeled 'O' and ending at point 'P_1'. The length of line 'OP_1' is a fixed user input called 'G_{max}'. The distances between the snaxels on the curve are kept. As mentioned above, the shape-velocity vector of each curve 'c' is dubbed '$\vec{v_c}$'. Vector '$\vec{v_c}$' which is perpendicular to the line 'OP_1', intersects with the circle at point 'P_2'. Point 'P_2' is located at '$(\frac{G_{max}}{2}, |\vec{v_c}|)$'. Therefore, the middle of the curve will get the shape velocity of the curve. The shape velocity is generated randomly in our method from an interval of $[v_{min}, v_{max}]$. The minimum and maximum are input parameters as discussed in the base speed section. Point 'P_3' is the intersection of the circle with vector '$-1 \cdot \vec{v_c}$', as shown in the figure. Point 'P_3' represents the fixed point which is used to shoot vectors through each projected snaxel's location. For example for a snaxel 's' on the curve, as shown in the figure, a vector is placed at the fixed point 'P_3' going through the projected snaxel's location to hit the surface of the circle at point 'x'. We name the vector '$\vec{v_s}$'. The length of this vector starting from the projected snaxel's location and ending at the point 'x' is the speed of the snaxel 's', which will always be positive or equal to zero. At the curve's joints (the curve's first and last snaxels) it will always be zero.

Figures	r	w_s	$[v_{min}, v_{max}]$	c_{min}	G_{max}
Figure 3.15(a)	0.1	1.0	[10.0,25.0]	3.0	10.0
Figure 3.15(b)	N/A	N/A	[10.0,25.0]	3.0	N/A

Table 3.2 Parameters used in the simulations

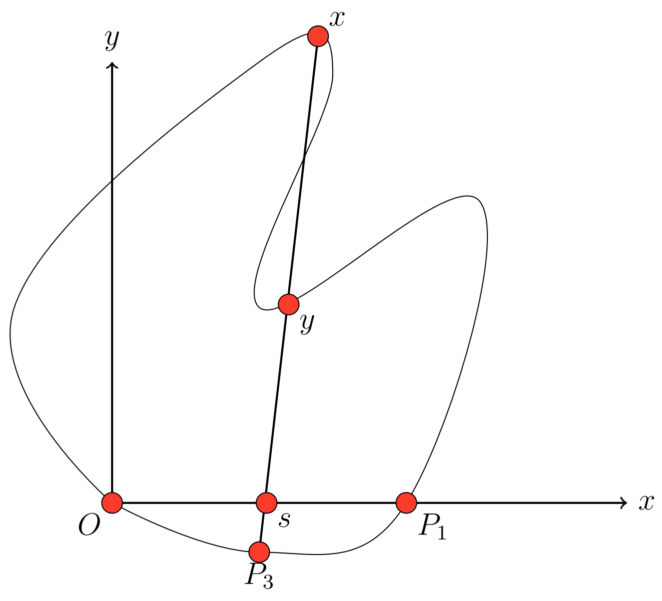


Figure 3.17 Example of an arbitrary shape with occlusions. The occlusions prohibits the method to be able to assign a speed to the snaxel 's'.

Every point on the shape should be visible from the point 'P_3' otherwise, the non-visible part of the shape will not be captured. As shown in Figure 3.17, if a part of the shape is occluded by itself, it will not affect the snaxels' speeds as it should. For example as shown in the figure, the velocity of snaxel 's' cannot be determined due to the fact that the vector '$\vec{P_3x}$' hits the shape in more than one point. This is one of the drawbacks of this speed-assigning scheme mechanism. Since the vector of each snaxel does not curve around occlusions and therefore only shapes without occlusions are suitable for this mechanism.

One can consider other shapes as long as it meets the criteria mentioned above. However, we are not focusing on these variations, our focus is on modeling cumulus clouds features which resemble different sized circles.
In the coming section, we'll describe our plans to grow the snake on the tessellation using the scheme discussed here to capture structures similar to cumulus clouds or volcanic ash clouds.

3.5 Growth Strategies

In the previous sections, we discussed how we represent the growing contour and how it moves on the tessellation. We also discussed how the speed of each snaxel is set with respect to which curve it belongs to and its location on that curve. However, the growth of the curves needs to be manipulated to get results which have structures similar to cumulus clouds or volcanic ash clouds. In this section we will discuss how we control the overall shape of the final result by introducing strategies for manipulating curves' growth. We also allow user interaction through a sketch-based system.

3.5.1 Length-Limits Table

Some control over the limit of each curve is needed. If the length limit stays the same for each curve, the curves will simply grow and keep on splitting. As shown in Figure 3.18, The small bumps of a cloud reside on big bulges: a recursive subdivision of volume and surface. We want to mimic this pattern using the allowed length limit of each curve.

As curves grow, their length increases. The control their length limit can control the exterior shape of the snake. Therefore, we decided to provide different length limits at different distances traveled by each curve. We constructed a table similar to the one shown in Table 3.3. The traveled distance by each curve is computed by picking the biggest distance traveled by any of the snaxels of that curve. If at any point, the traveled distance of the curve lies in any of the ranges shown in the table, the curve's length limit becomes the corresponding maximum length limit. Figure 3.19 shows a result obtained through this method. We use the tables to first allow for some simple structure to be formed. Then allow the structure to grow for some

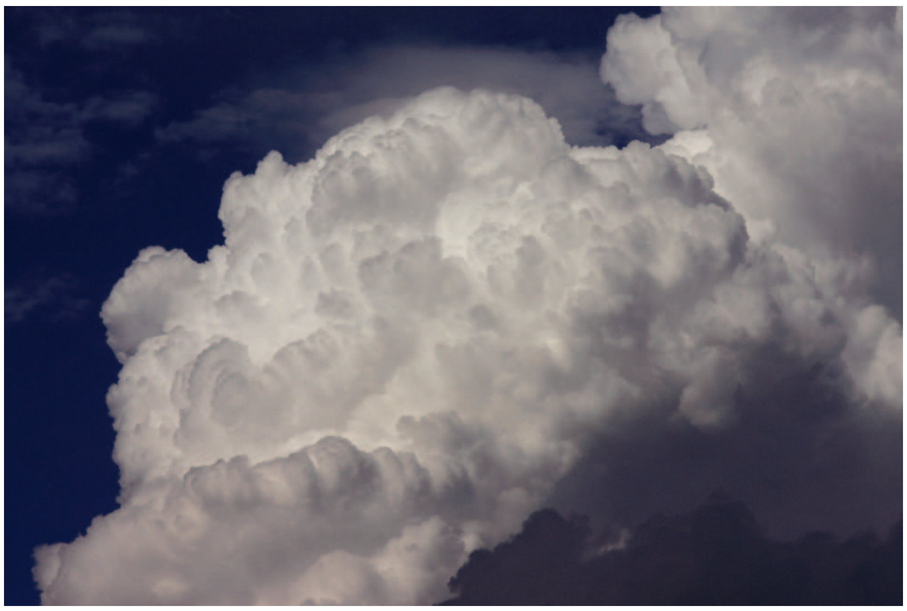


Figure 3.18 Example of a cumulus cloud's surface

distance and finally, apply texture by reducing the curves' lengths.

Traveled Distance Range	Maximum Weight-Limit
0-50	200
50-300	100000
> 300	25

Table 3.3 Example of a Weight-Limits Table

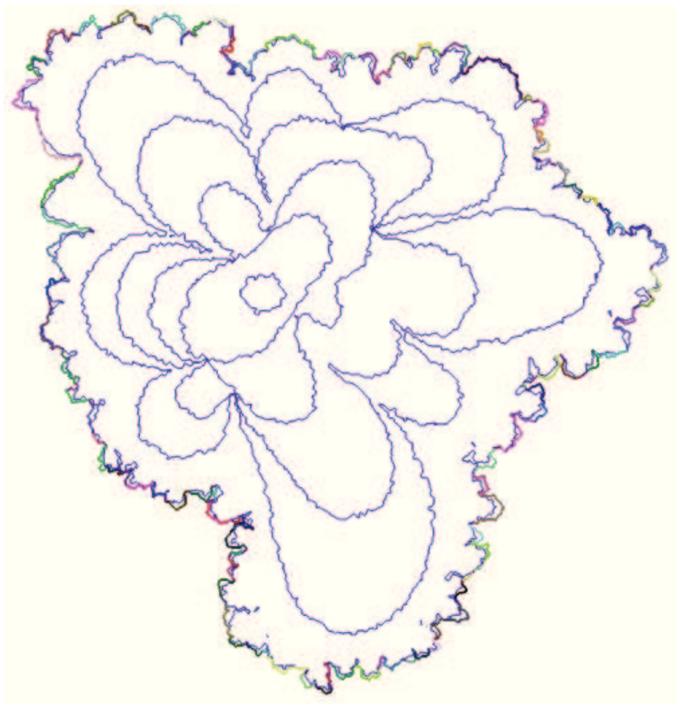


Figure 3.19 A result obtained by controlling the curves' length-limits through Table 3.3. The values for each parameter are shown in Table 3.5.

By adding another entry to the length-limits table shown in Table 3.3, we are able to make the result get a layered pattern mimicking a cumulus cloud or a volcanic ash cloud. Figure 3.20 shows the result obtained by modifying Table 3.3 to add another entry and changing it into Table 3.4. This change will make the parent curves to have a bigger maximum length. However, as the origin of the growth for all the curves is set at a fixed point, this change does not produce arbitrary complex shapes.

We grow more than one object to obtain arbitrary complex shapes. In the next subsection, we will discuss how grouping curves together can help us achieve generating more complex shapes.

Traveled Distance Range	Maximum Weight-Limit
0-50	200
50-200	100000
200-300	500
> 300	25

Table 3.4 Modified version of Table 3.3

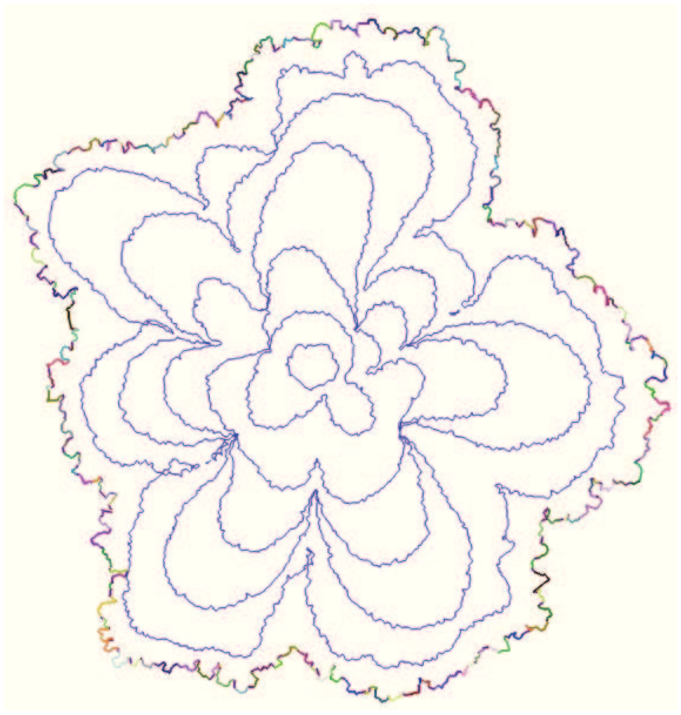


Figure 3.20 A result obtained by controlling the curves' length-limits through Table 3.4. The values for each parameter is shown in Table 3.5.

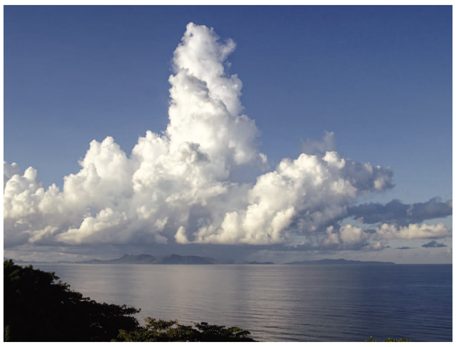
Figures	r	w_s	$[v_{min}, v_{max}]$	c_{min}	G_{max}
Figure 3.19 & 3.20	0.1	1.0	[5.0,30.0]	3.0	10.0

Table 3.5 Parameters used in the simulations

3.5.2 Clusters

We are not able to produce the arbitrary shape of cumulus clouds or the almost vertical shape of the volcanic ash cloud, just by manipulating the limit of length of curves. Figure 3.21 shows images of such arbitrary shapes of cumulus clouds. Our challenge is to make sure that curves stop their growth randomly but still manage to keep a cohesive shape. This means that curves should form separate groups and some groups should stop while others grow. To achieve this we introduce a way of grouping curves together so that we can manipulate the behavior of curves even more to obtain the discussed results.

Clouds
By Gilad Rom is licensed under CC BY 2.0 [21]


Cumulus congestus
By Ian Jacobs is licensed under CC BY 2.0 [28]

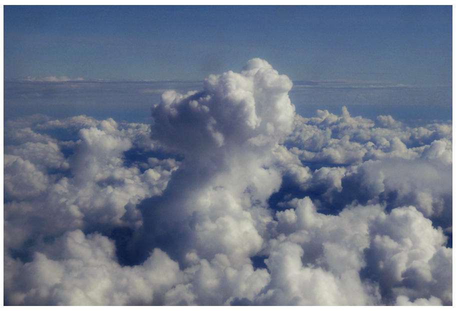

Cumulus mushroom
By Anders Sandberg is licensed under CC BY 2.0 [2]

Figure 3.21 Examples of Clouds' Structures

To produce arbitrary shapes using curves, we group some of the curves into a *Cluster*. We group curves into a cluster gradually. Each cluster is born out of one curve. As that curve splits, the resulting curves are added to the cluster and the same is done for the splitting of the resulting curves. Any new curve created from a long curve breaking is added to the cluster of the said broken curve. Once a snaxel of a curve on a cluster reaches a predetermined traveled distance, the cluster stops. Upon stopping, all curves in that cluster stop. Then, we choose one of the curves of the cluster to start the next cluster. This process is repeated for a predetermined number of times or till the user is satisfied with the result. Figure 3.22 shows a result which was generated through growing 2 clusters using this method. The number of clusters which the system attempts to grow is a parameter called 'n_c'.

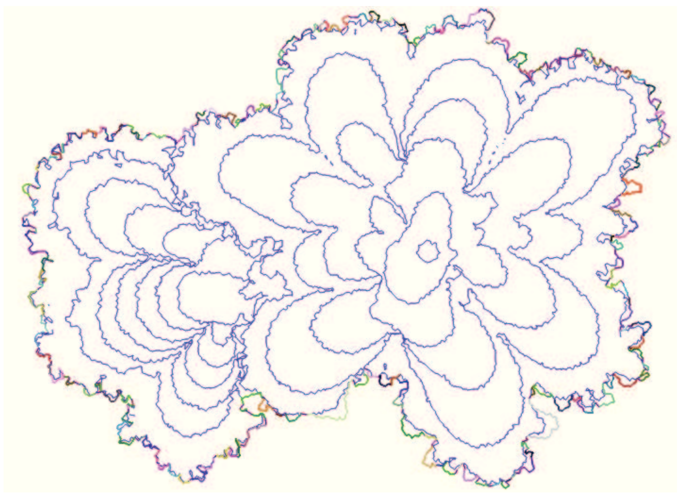


Figure 3.22 A result obtained growing 2 clusters following Table 3.3. The horizontal line at the bottom of the image indicates that the growth has reached the boundaries of the screen. In our implementation, the snaxels are stopped when they reach the boundaries of the screen. The values for each parameter is shown in Table 3.6.

In our simulations, the traveled distance of the snaxels inside the cluster can be chosen randomly therefore, making some of the clusters to grow less than the others. We reset the traveled distance by setting it to a new random value from the interval of '$[t_{min}, t_{max}]$'. Also the amount that a cluster is going to grow can be changed to create clusters which have bigger and more stretched out curves than others. We pick a random allowed growth for each cluster from an interval of '$[g_{min}, g_{max}]$'. If the difference between the traveled distance and the growth allowed is small then the cluster will have a short life. The cluster's irregularity depends on the last curve's length-limit assigned to it.

Figures	r	w_s	$[v_{min}, v_{max}]$	c_{min}	G_{max}	$[t_{min}, t_{max}]$	$[g_{min}, g_{max}]$
Figure 3.22	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,1]	[350,380]

Table 3.6 Parameters used in the simulations. N_s stands for number of snaxels on the snake. N_v stands for number of tessellation's vertices.

3.5.3 Sketch-based

We decided to give the user the ability to draw a shape into the tessellation, since randomization may not capture the artist's imagination and the artist may wish to grow a cloud similar to a target shape of their choosing. Basically, we developed a sketch-based system on to our existing model. It works by the user drawing multiple connected curves, and the system will find the associated triangles in the tessellation and marks them. Then the process starts by the snake engulfing the marked triangles first. Then, the process will follow the length-limits table. The user can control the thickness of snake's initial growth around the drawn shape through the parameter 'c_t'. Figure 3.23 shows a user attempt to model a mushroom cloud. To obtain this result the Table 3.7 was used. The process was stopped by the user.

Traveled Distance Range	Maximum Length-Limit
0-50	500
50-100	50
> 100	50

Table 3.7 The Weight-Limits Table used in figure 3.23

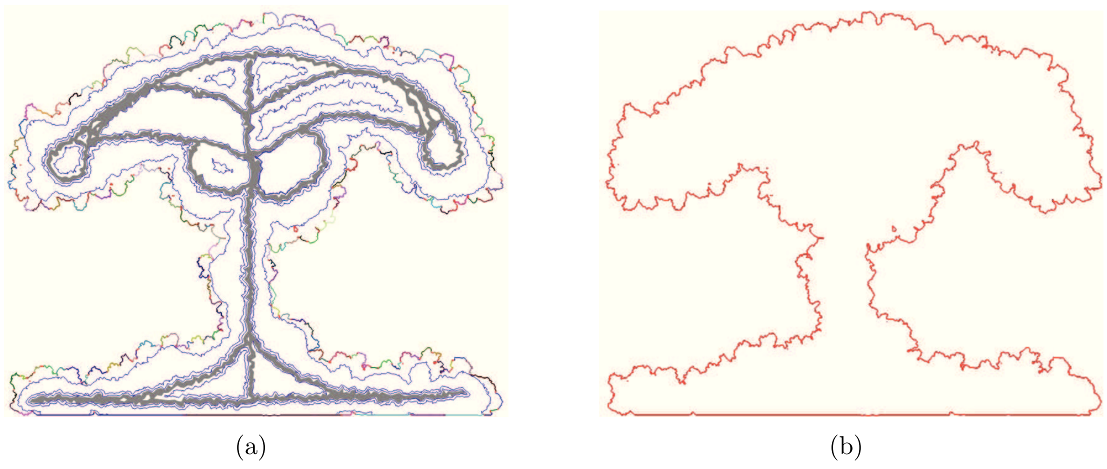


Figure 3.23 User attempt to model a mushroom cloud. Left shows the result and the initial shape drawn by the user. Right shows the final result. The process was stopped by the user. The values for each parameter is shown in Table 3.8.

Figures	r	w_s	$[v_{min}, v_{max}]$	c_{min}	G_{max}	c_t
Figure 3.23	0.1	1.0	[2.0,25.0]	3.0	10.0	1.0

Table 3.8 Parameters used in the simulations. N_s stands for number of snaxels on the snake. N_v stands for number of tessellation's vertices.

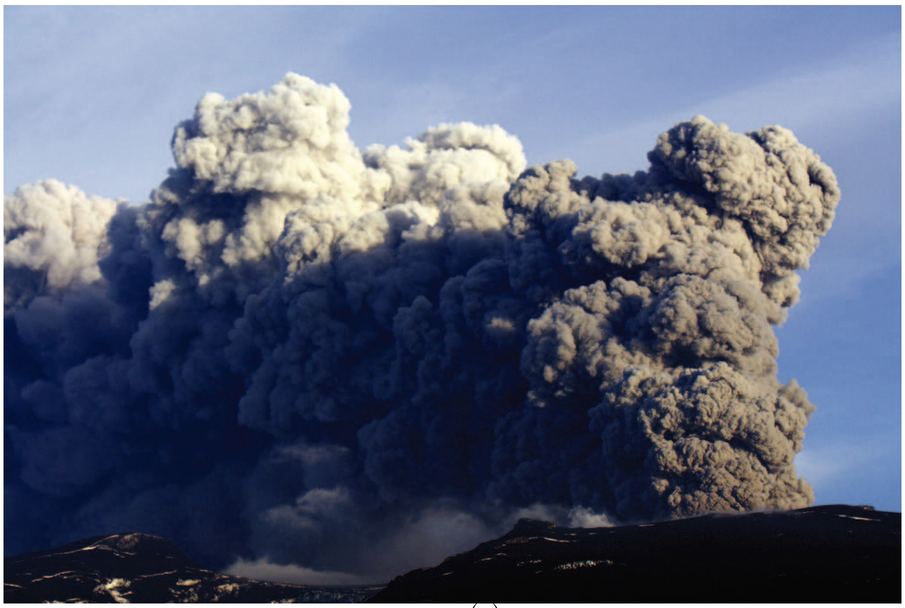
Chapter 4

Results & Discussion

In the previous chapter, we discussed our algorithm and how it generates cloud-like objects. We also discussed our approach and the parameters which can be used to control or influence the result. In this chapter, we will begin by showing our results and discuss and compare them with our target shapes and images. Then, we will show how the parameters can impact the result and cause more variations and why certain values are preferred for some parameters.

Figure 4.1 shows a comparison between a result from our algorithm and a silhouette of a volcanic ash cloud. The result shown in Figure 4.1(b) has features similar to the ones shown in Figure 4.1(c). Smoothed bumps on the bottom section of the result are very similar to the bumps on the silhouette's boundary. Although the silhouette has very few small extrusions on its boundary, few small extrusions on the boundary of the result are similar to the ones on the silhouette. The roundness of large bulges on the result mimic the ones on the silhouette's boundary. This comparison shows our algorithm can capture large and small features of the target image, even though the silhouette's features are not produced from a cross section of the target image.

We used our approach to produce results which capture the surface and shape complexity of cumulus clouds and similar objects. Figure 4.2 shows a comparison between our results and real world cumulus clouds and volcanic ash-clouds. The results are automatically generated and do not attempt to specifically model the target images. They are intended to demonstrate that we have been able to capture the main features of our target objects. We are going to discuss how our parameters capture these features.


Eyjafjallajökull Eruption
By Söring is licensed under CC BY 2.0 [57]
(a)

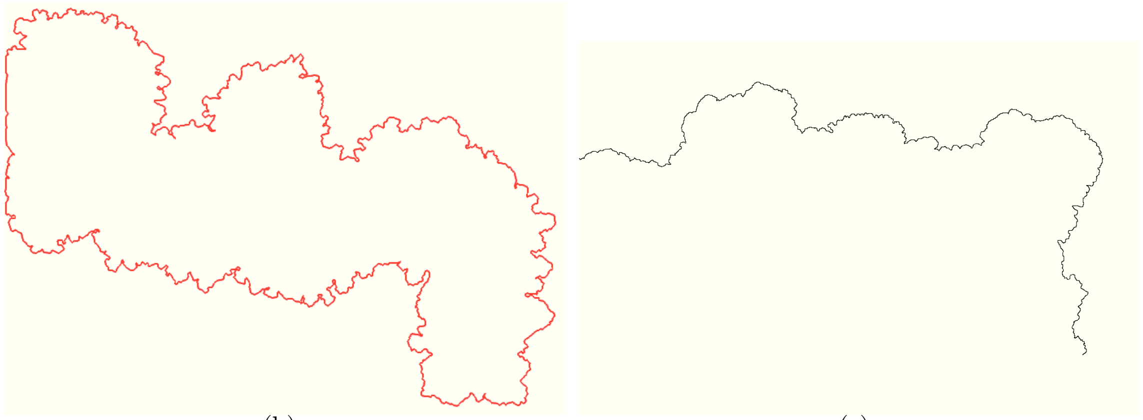
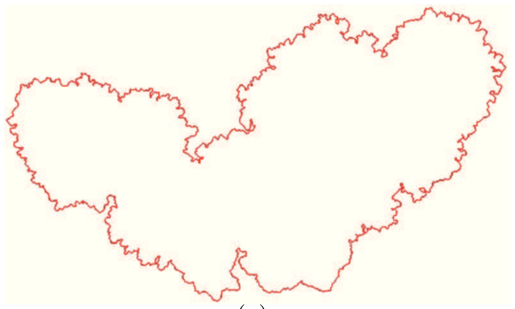

(b) (c)

Figure 4.1 (a) Eyjafjallajökull volcanic eruption in Iceland. (b) One of our results. (c) A 2 dimensional silhouette extracted from the image.

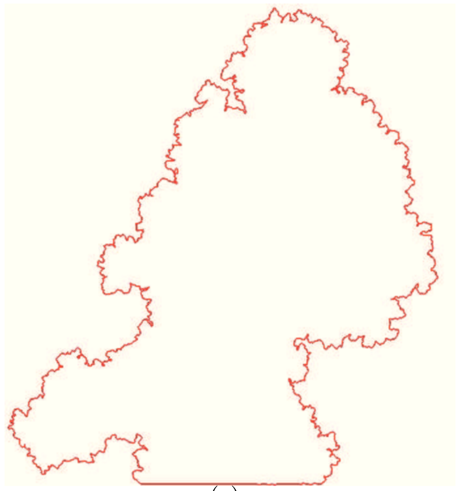
Our method can capture irregular shapes. For instance, Figure 4.2(e) shows a result which has a very irregular shape and detailed surface features similar to the target Figure 4.2(f). The result appears to have small bulged out features similar to the target image. This is due to the variations in the cluster's growth distance. As discussed in previous chapter, the intervals '$[t_{min}, t_{max}]$' & '$[g_{min}, g_{max}]$' control how much a cluster grows. A cluster will shrink in size by resetting the traveled distance to a large value. Therefore, it reaches the final entries faster. If multiple clusters grow on top of each other and each have a small size then the final result will have a more irregular shape. Also the detailed surface is obtained by assigning a low value to the last entry in the length-limits table. This ensures that at the final step, the curves are going to have very short and limited length which results in detailed surface. Later, we will discuss how our parameters control the amount of details on the surface of our results.

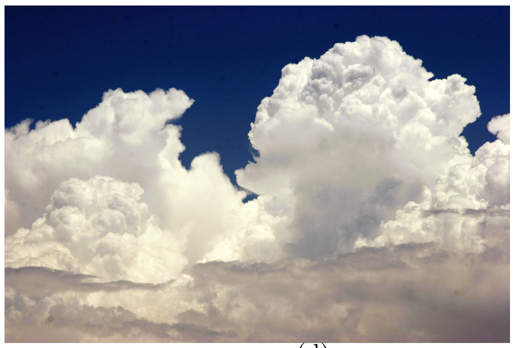
Our method can also capture bulged out features of cumulus clouds or volcanic ash-cloud. For example, Figure 4.2(c) shows a result that has the bulged out features of a cumulus cloud like the one shown in Figure 4.2(d). The result has different sized bulges which have grown out at different distances. This is caused by the length-limits table's values. As discussed in previous chapter, the variety in the table allows for a random structure to form. Then this structure grows to a certain distance from the origin of the growth. Finally, the last entries will add bumps and details to the structure. The random speed values assigned at each curve controls how big the bulges get.

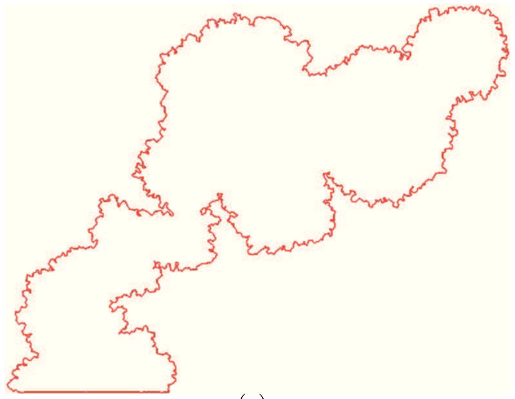
Finally, Figure 4.2(a) shows that our algorithm is able to add varieties of details on the surface of the result. It is very similar to the cumulus cloud in Figure 4.2(b). The result has bulges that appear to grow more aggressively than others and those that appear to be bigger. The lower part of the result has more smoothed bumps than the rest of the body. These changes are due to two separate parameters. As discussed in previous chapter, the interval '$[g_{min}, g_{max}]$' controls how much clusters grow. Therefore, if the last two entries in the length-limits table have their maximum length-limit set to two different values, then the clusters can gain different surface details by assigning the interval '$[g_{min}, g_{max}]$' to cover the required traveled distance in both of those entries. This combination only controls the size of the curves on the clusters. Another parameter in control of this feature is the speed interval '$[v_{min}, v_{max}]$'. As discussed in previous chapter, the speed interval controls the speed of the growing curves. By increasing the interval, curves can vary in their growth and some grow slowly and others more aggressively. This results in a highly irregular surface.

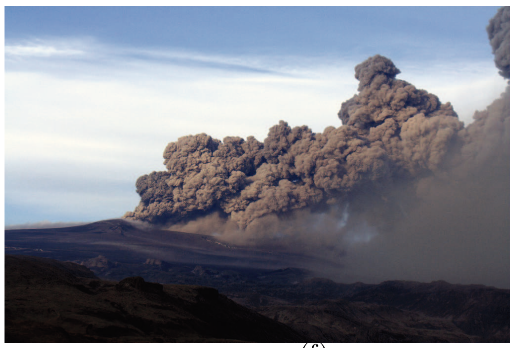

(a)

(b)


(c)


(d)


(e)


(f)

Eyjafjallajökull Eruption
By Söring is licensed under CC BY 2.0 [58]

Figure 4.2 Comparison between our results and real world cumulus clouds and volcanic ash-clouds.

We have obtained varied results even with the same settings, due to the randomness involved in choosing features such as first curve of a cluster or velocity of snaxels. The choice of curve for growing the next cluster seems to heavily impact and influence the result. The results shown in Figure 4.3 are all generated with the same settings and length-limits table. However, they are varied in shape. This is due to the initial curves chosen to grow new clusters. As mentioned in the previous chapter, new clusters grow out of a single randomly chosen curve. For example, Figure 4.3(b) and 4.3(e) both have the same number of clusters. However, Figure 4.3(b) is more elongated than the result shown in Figure 4.3(e).

Traveled Distance Range	Maximum Length-Limit
0-50	400
50-150	1000
150-200	1500
200-250	50

Table 4.1 Length-limits table used in our results

As discussed above, we managed to create clusters which are varied in size. Figure 4.3(f) has a distinct feature which is magnified in Figure 4.4. The upper part of the result seems to get narrow and form a neck. We want to be able to produce such variations. We can mimic similar features and obtain more varied shapes by increasing the number of clusters and increasing the interval for resetting the traveled distance to have higher minimum and maximum. Figure 4.5 shows results with increase in number of clusters. However, obtaining varied shapes with narrow features is not always guaranteed. As the number of clusters grow, they can overlap and simply add to the thickness of the body of the result. Figure 4.6 shows such an object. The number of clusters in this result is similar to the number of clusters in Figure 4.5(a).

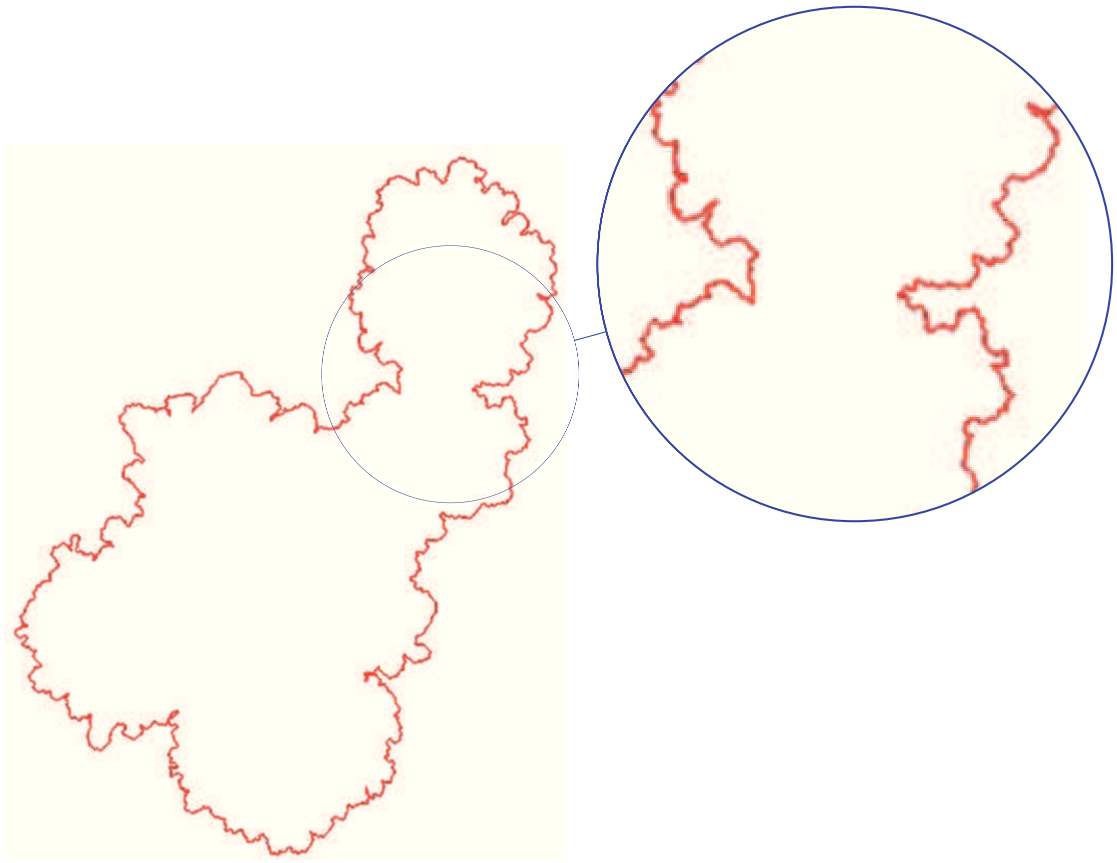
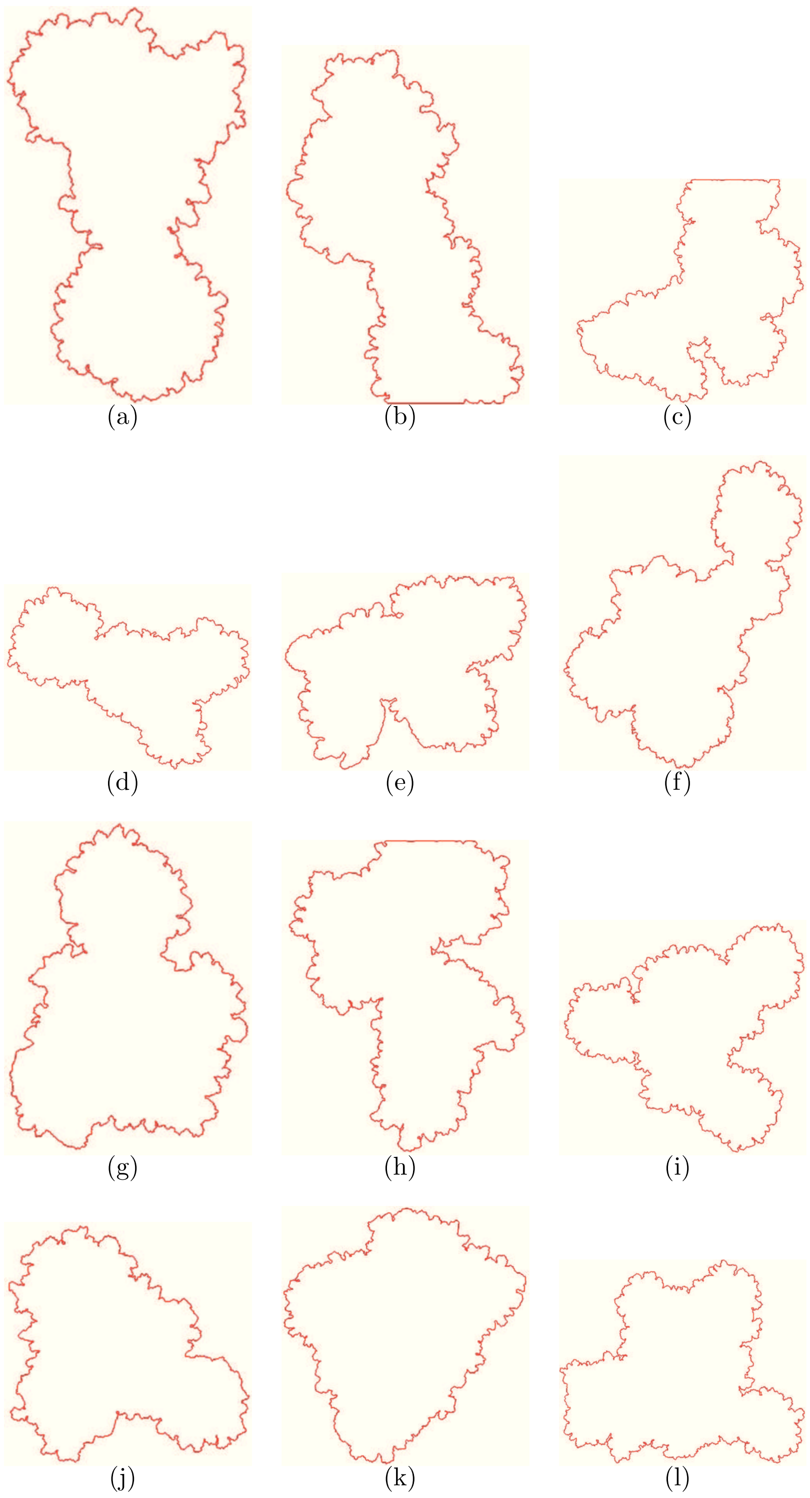


Figure 4.4 A 'neck' feature in one of our results



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l)

Figure 4.3 Results obtained through same settings with exception in number of clusters From left to right column, 2,3 and 4 clusters were used. Table 4.1 was used for length limit manipulations.

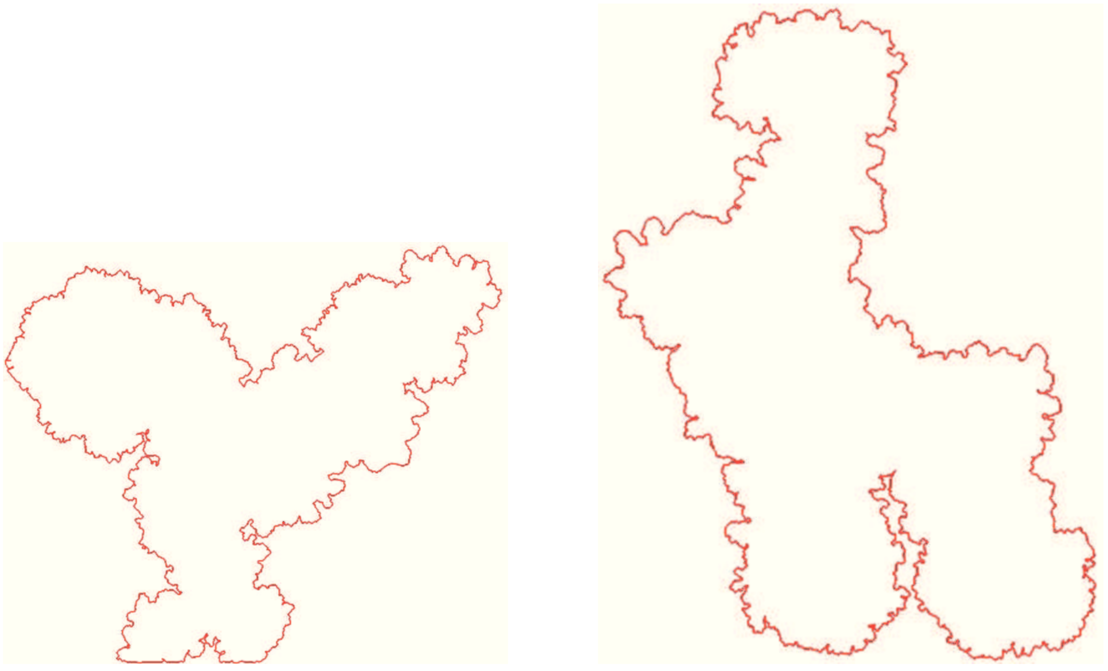

(a) (b)

Figure 4.5 Result obtained by increasing the number of clusters

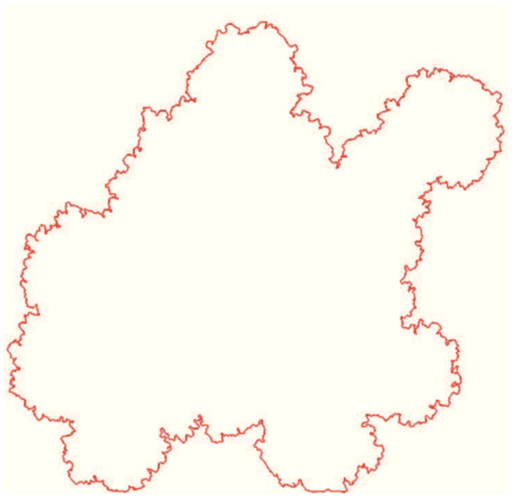


Figure 4.6 A result with high number of clusters

As discussed above, we can add more details by reducing the maximum length-limit in the last entry of the length-limit table. This way the curves are reduced in size and give a more irregular frontier. Figure 4.7 shows results with more details. In

Figure 4.8, some of the detailed features are magnified. The figure demonstrates the curve's maximum speed's influence on the details of the boundary of results. This is caused by the interval which the curve's maximum speeds are chosen from. When the interval gets larger, some curves grow more aggressively than others, resulting in a more irregular boundary.

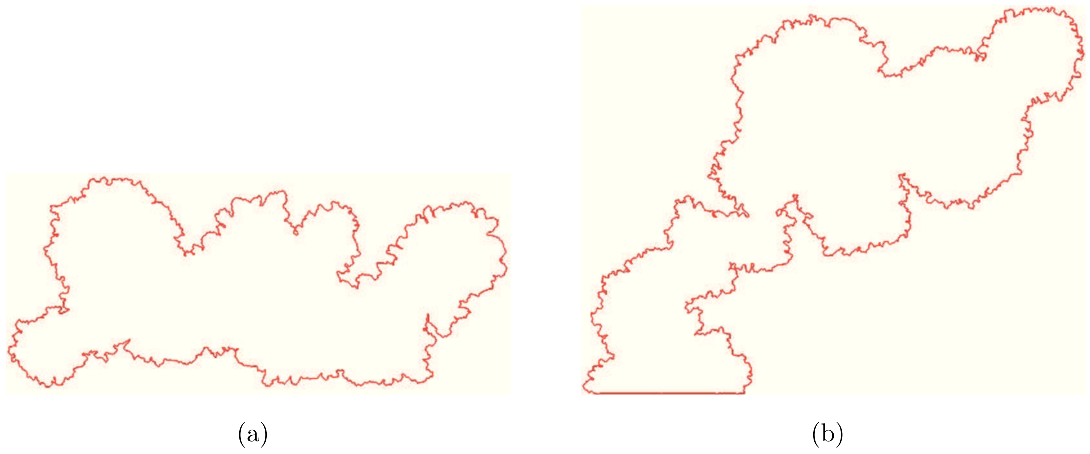


Figure 4.7 Results obtained by reducing the final row of the length-limit used shown in Table 4.1 to 25. Figure 4.7(b) is the same result as the one shown in Figure 4.2(e).

The positioning of clusters controls the overall shape directly. For example, consider Figure 4.9; both are generated with same settings and number of clusters with mere difference in surface details due to last entry in the length-limit table, but the positioning of the clusters has completely changed the type of the result. As discussed above, the curves to grow new clusters out of, are chosen randomly, causing varied shapes. Figure 4.9(b) appears more elongated than Figure 4.9(a).

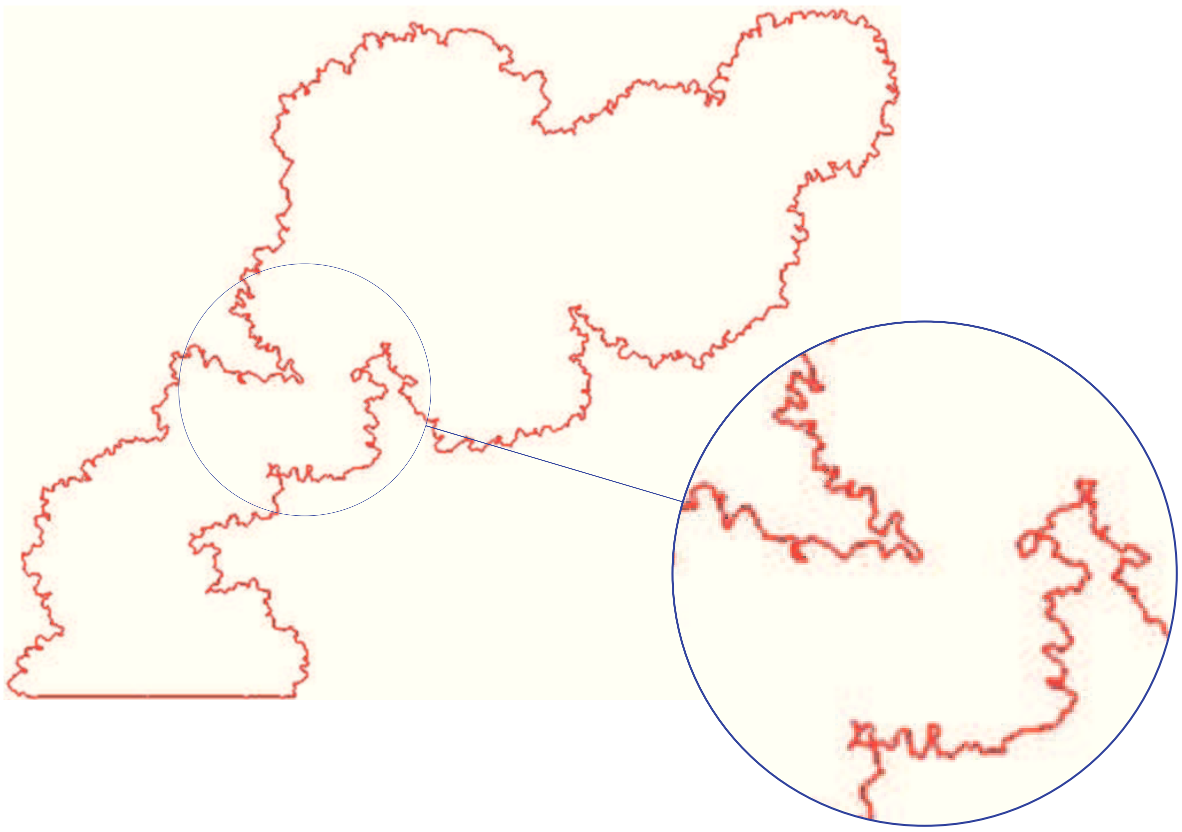
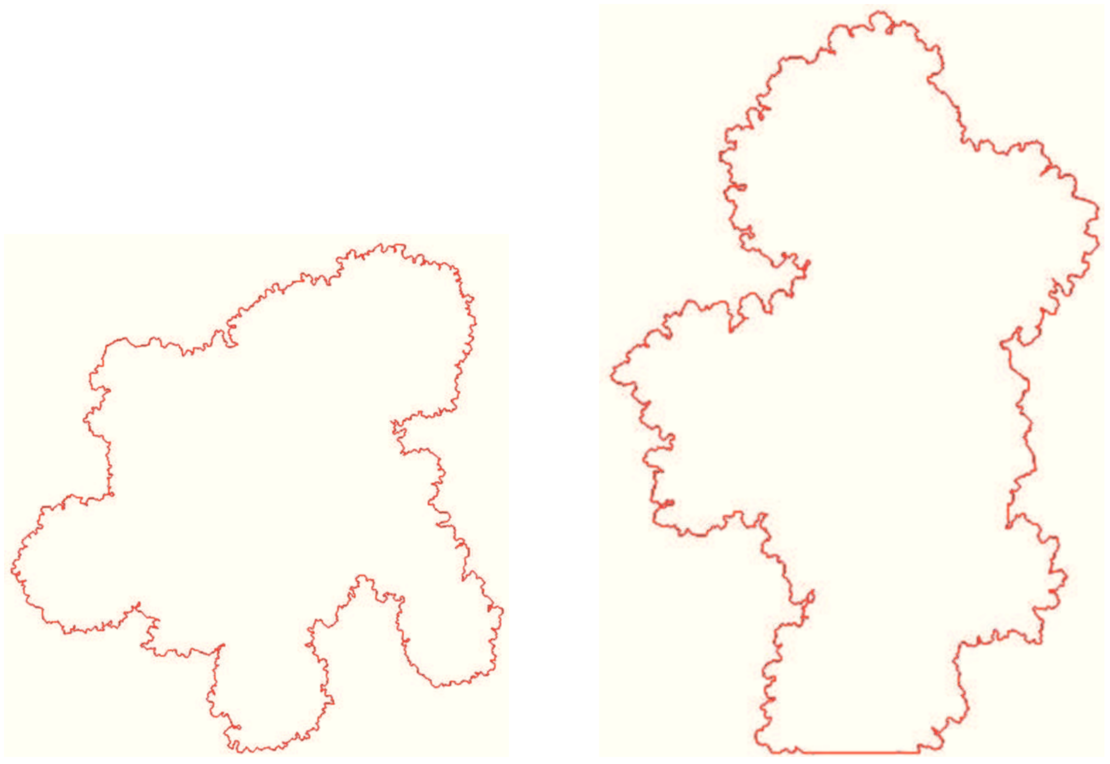


Figure 4.8 Magnifying details on the result in Figure 4.2(e)
;



(a) (b)

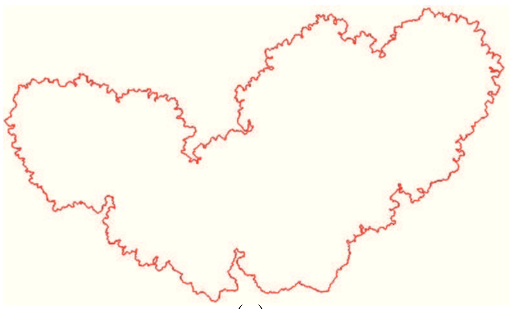
Figure 4.9 Comparison of clusters' locations influence

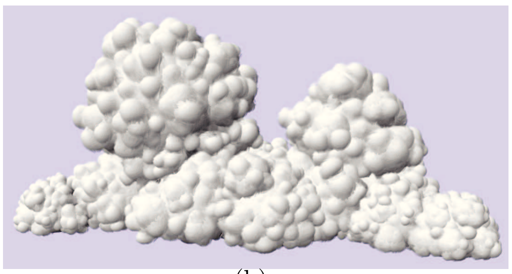
Figure 4.10 shows comparisons between our results and previous works. Contrary to our algorithm, previous works rely on rendering to add details except for Neyret [43]'s result: Figure 4.10(d) which is produced by a simulation of particles. In our work, both shape and details are captured in the modeling phase rather than the rendering.

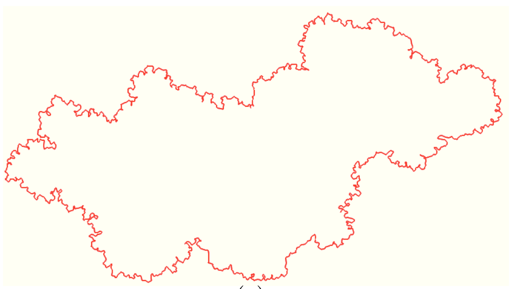
Figure 4.10(a) is one of our results and Figure 4.10(b) is from Bouthors and Neyret [6]. Our result has similar structure to theirs. Their result has small visible spheres on larger ones. This hierarchy of spheres produces regularities at different scales. Our result uses circles only in calculations and concept. The boundary produced by our method has more irregularities than their result. Both results have small bumps on large bulges. However, our result has more details on its boundary.

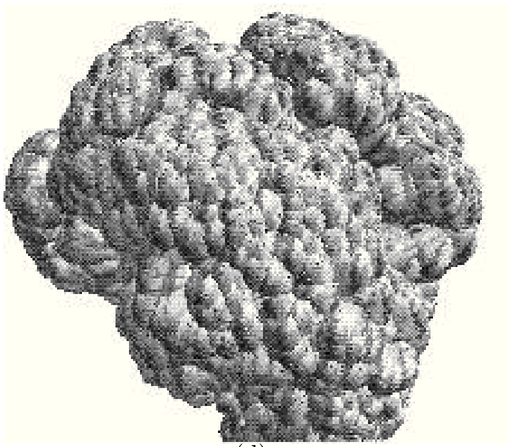
Figure 4.10(c) is one of our results and Figure 4.10(d) is from Neyret [43]. Although our result has similar structure and surface details to Figure 4.10(d), it has more sharp boundary details. The details on the result in Figure 4.10(d) are much more smooth than ours in Figure 4.10(c). Sharp details can be observed on the left side of our result in Figure 4.10(c).

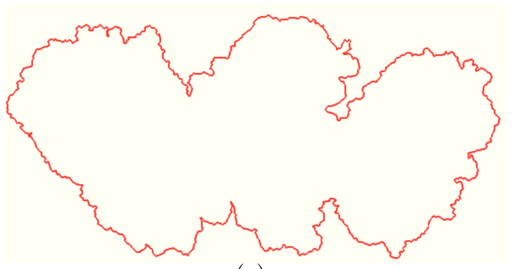
Figure 4.10(e) is one of our results and Figure 4.10(f) is from Trembilski and Broßler [63]. Their result is lit with diffuse reflection. Our method can capture smooth features of clouds similar to theirs. We create smooth features by increasing the length limit of curves. We can also capture shapes with more irregularity. Their method puts spheres on top of one another to try to cover areas of the surface model with low height. This approach reduces the variety of the shapes. Their result has details on large scale. It has no details on small scale. In contrast, our result has both large and small scale details. It appears round in large scale but has details in the small scale.


(a)


(b)


(c)


(d)


(e)

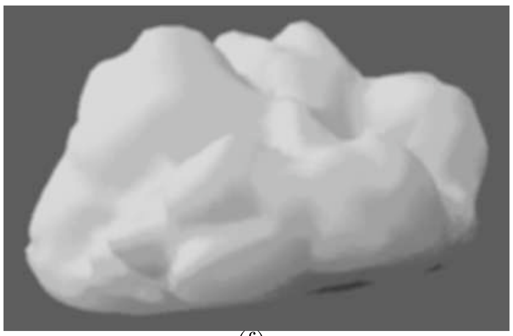

(f)

Figure 4.10 Comparison between our results and previous works. (b) A result from Bouthors and Neyret [6]. (d) A result from Neyret [43]. (f) A result from Trembilski and Broßler [63].

Our results were generated using the parameters and settings in Table 4.2.

Figures	r	w_s	$[v_{min}, v_{max}]$	c_{min}	G_{max}	$[t_{min}, t_{max}]$	$[g_{min}, g_{max}]$	n_c	length-limits table
4.1(b)	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,150]	[250,280]	5	4.4
4.2(a)	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,100]	[240,270]	4	4.3
4.2(c)	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,250]	[250,280]	15	4.4
4.2(e)	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,150]	[240,250]	7	4.5
4.3(a)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	2	4.1
4.3(b)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	3	4.1
4.3(c)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	4	4.1
4.3(d)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	2	4.1
4.3(e)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	3	4.1
4.3(f)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	4	4.1
4.3(g)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	2	4.1
4.3(h)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	3	4.1
4.3(i)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	4	4.1
4.3(j)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	2	4.1
4.3(k)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	3	4.1
4.3(l)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	4	4.1
4.5(a)	0.1	1.0	[5.0,30.0]	3.0	10.0	[50,200]	[240,300]	15	4.6
4.5(b)	0.1	1.0	[5.0,30.0]	3.0	10.0	[100,200]	[240,300]	10	4.6
4.6	0.1	1.0	[5.0,30.0]	3.0	10.0	[50,100]	[200,240]	15	4.7
4.7(a)	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,150]	[240,250]	7	4.5
4.9(a)	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,150]	[240,250]	7	4.5
4.9(b)	0.1	1.0	[5.0,30.0]	3.0	10.0	[0,150]	[240,250]	7	4.1
4.10(c)	0.1	1.0	[5.0,30.0]	3.0	10.0	[50,150]	[200,240]	15	4.8
4.10(e)	0.1	1.0	[2.0,25.0]	3.0	10.0	[0,50]	[240,250]	2	4.9

Table 4.2 Parameters used in the simulations. n_c stands for number of clusters.

Traveled Distance Range	Maximum Length-Limit
0-50	400
50-100	100000
100-200	400
200-250	30
> 250	25

Table 4.3 One of the length-limits table used in our results

Traveled Distance Range	Maximum Length-Limit
0-50	400
50-100	100000
100-200	400
200-260	50
> 260	25

Table 4.4 One of the length-limits table used in our results

Traveled Distance Range	Maximum Length-Limit
0-50	400
50-150	1000
150-200	1500
> 200	25

Table 4.5 Length-limits table used in our results

Traveled Distance Range	Maximum Length-Limit
0-50	400
50-100	1000
100-200	400
200-250	75
250-270	50
> 270	25

Table 4.6 Length-limits table used in our results

Traveled Distance Range	Maximum Length-Limit
0-50	200
50-100	100000
100-150	200
> 150	30

Table 4.7 Length-limits table used in our results

Traveled Distance Range	Maximum Length-Limit
0-50	200
50-100	100000
100-150	200
> 150	25

Table 4.8 Length-limits table used in our results

Traveled Distance Range	Maximum Length-Limit
0-50	400
50-150	1000
150-200	1500
200-220	500
> 220	25

Table 4.9 Length-limits table used in our results

Chapter 5

Future Work

In the previous chapter, we presented our results and compared them to real world clouds and volcanic ash-clouds. We also discussed variations in our results and how our parameterization controls these variations. In this chapter, we will discuss future work and areas that can be investigated further in the 2 dimensional representation of snaxels and extending snaxels into 3 dimensions.

One direction for further investigation is to try to gain control over the overall structure of the results. Curves are picked randomly to grow new clusters. Each choice directly affects the overall structure of the result. Control over this feature may allow for modeling categorically and therefore, only produce results of a certain phenomenon. This would also allow a user to grow numerous cloud like structures of not only cumulus type but other types as well. More research is needed in investigating different approaches to control curves selection for cluster growth.

Another direction for further investigation is to try to produce results of different sizes. Two results can have the same overall structure but differ in their size of bulges. How much a cluster grows can have profound influence on the result. Essentially our results are clusters being stacked up on each other. A process to control how much they grow can impact the result. In our work, clusters grow to a limit determined by an interval. However, future work can explore methods to manipulate the limit of cluster's growth in order to mimic natural-looking results more directly.

Snaxels handle self-collision and point density inherently. As a result, only implementation of a strategy which modifies their behavior, is an issue. This and their simulated growth open up a big space for future works and investigations. One topic that interests us is the modeling of coral reefs and similar objects. Figure 5.1 shows an example of a coral reef. It is given that a lot of coral reefs are fractal looking but there are many of which that have dendritic features and poses an irregular surface with a somewhat regular structure. We believe that an investigation into this problem can be very fruitful.

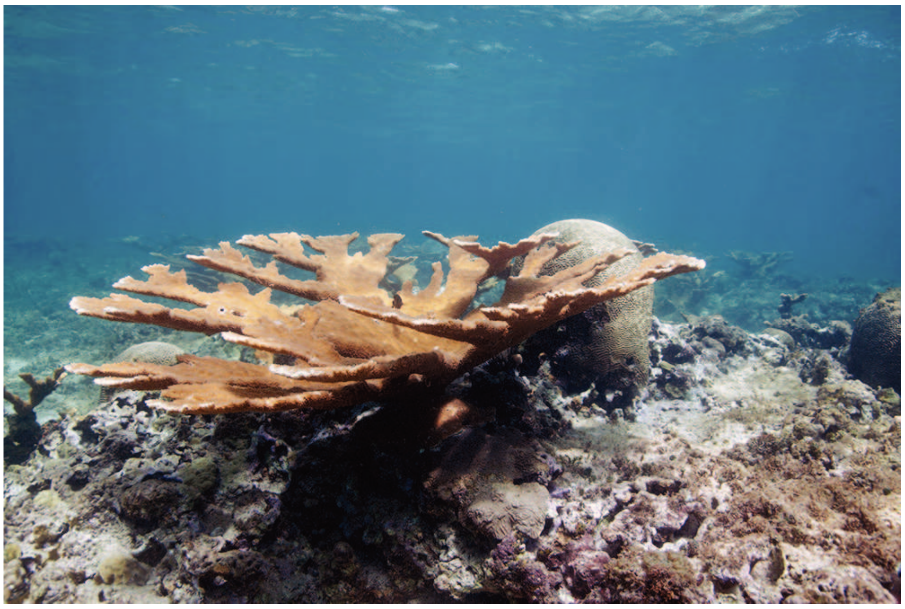


Figure 5.1 Example of a coral reef; Elkhorn coral near Vega Baja, Puerto Rico.

Another topic for future investigations would be different speed schemes. As mentioned, snaxels' propagation is directly influenced by the speeds that are assigned to them. Therefore, changing the scheme with which the speed changes can affect the contour texture. Another future plan would be implementation of snaxels' contraction. In our methodology we do not allow snaxels to move backwards on the tessellation edges. If such a process is allowed, we could investigate creating automatic animations for created models.

5.1 Snaxels in 3D: Mesh Growth Using Snaxels

We have been trying different approaches to try to bring snaxels into 3 dimensions. Appendix A describes the different approaches we pursued. Extending snaxels into

3D is our main goal for future work. We have made some progress on designing an algorithm to handle the snaxels' events. In 3D, snaxels move on a 3 dimensional tessellation or a tetrahedral mesh. The active contour or the snake will be represented as a mesh in 3 dimensions. For us the challenge is to be able to maintain the snaxels' connections with each other after each event that takes place and make sure the snaxels' connections are conforming to the tessellation. The snaxels' events are similar to the ones in 2 dimensions. However, the difference is in the handling of them, particularly birth and merge. The main challenge is in patching the holes generated. In our future work, we plan to work on being able to successfully patch the holes and maintain the snaxels connectivity.

Chapter 6

Conclusion

We introduced a procedural modeling algorithm for modeling cumulus clouds, volcanic ash clouds and similar bodies in 2 dimensions. We also introduced a sketch-based system to model such objects in 2 dimensions. Our system is based on active contour technique, moving on a given tessellation. The contour handles self collision and point density inherently. The resolution of the result is dependent on the tessellation's resolution. The growth of the contour is influenced through manipulating the speed of each snaxel. Our method does this through a scheme which enables mapping each snaxels' speed to a quantity derived from relating snaxels to the circumference of a convex shape. This will force snaxels to mimic the convex shape and therefore, enables us to influence the growing boundary's details. Our method also enables simulating the snaxels' growth by just adjusting a few parameters.

We have also started investigating extending this technique to 3 dimensions. This seems to be a fruitful frontier for future work with promising results. The extension of snaxels to 3 dimension, would allow us to investigate extending our plans and algorithms to the 3 dimensional version.

List of References

[1] Abraxas3d. `IMG_63231`, June 28, 2006. [Online; accessed April 05, 2017].

[2] Anders Sandberg. Cumulus mushroom, August 22, 2014. [Online; accessed March 20, 2017].

[3] Matthew Beardall, Mckay Farley, Darius Ouderkirk, Jeremy Smith, Michael Jones, and Parris K Egbert. Goblins by spheroidal weathering. In *NPH*, pages 7–14, 2007.

[4] Stephan Bischoff, Tobias Weyand, and Leif Kobbelt. Snakes on triangle meshes. In *Bildverarbeitung für die Medizin 2005*, pages 208–212. Springer, 2005.

[5] James F Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *Acm Siggraph Computer Graphics*, volume 16, pages 21–29. ACM, 1982.

[6] Antoine Bouthors and Fabrice Neyret. Modeling clouds shape. In Eric Galin and Marc Alexa, editors, *Eurographics (short papers)*, pages –, Grenoble, France, August 2004. Eurographics Association.

[7] Antoine Bouthors, Fabrice Neyret, and Sylvain Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In Eric Galin and Norishige Chiba, editors, *Eurographics Workshop on Natural Phenomena*, Vienne, Austria, September 2006. Eurographics.

[8] Antoine Bouthors, Fabrice Neyret, Nelson Max, Eric Bruneton, and Cyril Crassin. Interactive multiple anisotropic scattering in clouds. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 173–182. ACM, 2008.

[9] John Brosz, Faramarz F Samavati, and Mario Costa Sousa. Terrain synthesis by-example. In *Advances in Computer Graphics and Computer Vision*, pages 58–77. Springer, 2007.

[10] Matthew T. Cook and Arvin Agah. A survey of sketch-based 3-d modeling techniques. *Interacting with Computers*, 21(3):201 – 211, 2009.

[11] Giliam JP de Carpentier and Rafael Bidarra. Interactive GPU-based procedural heightfield brushes. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 55–62. ACM, 2009.

[12] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th annual conference on computer graphics and interactive techniques*, pages 19–28. ACM Press/Addison-Wesley Publishing Co., 2000.

[13] Yoshinori Dobashi, Tomoyuki Nishita, Hideo Yamashita, and Tsuyoshi Okita. Using metaballs to modeling and animate clouds from satellite images. *The Visual Computer*, 15(9):471–482, 1999.

[14] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Kohling Pedersen. Modeling and rendering of weathered stone. In *ACM SIGGRAPH 2006 Courses*, page 4. ACM, 2006.

[15] David S Ebert. Volumetric modeling with implicit functions: A cloud is born. In *Visual Proceedings of SIGGRAPH*, volume 97, page 147. Los Angeles, CA, USA, 1997.

[16] Pantelis Elinas and Wolfgang Stürzlinger. Real-time rendering of 3d clouds. *Journal of Graphics Tools*, 5(4):33–45, 2000.

[17] Dieter Finkenzeller and Jan Bender. Semantic representation of complex building structures. In *Computer Graphics and Visualization (CGV 2008)-IADIS Multi Conference on Computer Science and Information Systems, Amsterdam, The Netherlands*, 2008.

[18] John Ambrose Fleming. *Magnets and electric currents: An elementary treatise for the use of electrical artisans and science teachers.* E. & F. N. Spon; Spon & Chamberlain, London, 1902.

[19] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982.

[20] Geoffrey Y Gardner. Visual simulation of clouds. In *Acm Siggraph Computer Graphics*, volume 19, pages 297–304. ACM, 1985.

[21] Gilad Rom. Clouds, August 04, 2013. [Online; accessed March 20, 2017].

[22] Prashant Goswami. *Real-time landscape-size convective clouds simulation and rendering.* PhD thesis, INRIA, 2016.

[23] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Real-time procedural generation of ‘pseudo infinite’ cities. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE ’03, pages 87–ff, New York, NY, USA, 2003. ACM.

[24] Eric Guérin, Julie Digne, Eric Galin, and Adrien Peytavie. Sparse representation of terrains for procedural modeling. In *Computer Graphics Forum*, volume 35, pages 177–187. Wiley Online Library, 2016.

[25] Mark J Harris, William V Baxter, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 92–101. Eurographics Association, 2003.

[26] Mark J Harris and Anselmo Lastra. Real-time cloud rendering. In *Computer Graphics Forum*, volume 20, pages 76–85, 2001.

[27] Houssam Hnaidi, Eric Guérin, Samir Akkouche, Adrien Peytavie, and Eric Galin. Feature based terrain generation using diffusion equation. In *Computer Graphics Forum*, volume 29, pages 2179–2186. Wiley Online Library, 2010.

[28] Ian Jacobs. Cumulus congestus, March 24, 2009. [Online; accessed March 20, 2017].

[29] Jonathan E. Shaw. Tavurvur volcano, rabaul, papua new guinea, November 21, 2009. [Online; accessed March 03, 2017].

[30] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[31] George Kelly and Hugh McCabe. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology*, pages 8–16, 2007.

[32] killa uaira. tormenta cloud, December 2013. [Online; accessed April 05, 2017].

[33] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert. Interactive translucent volume rendering and procedural modeling. In *Visualization, 2002. VIS 2002. IEEE*, pages 109–116. IEEE, 2002.

[34] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. In *ACM Transactions on Graphics (TOG)*, volume 28, page 54. ACM, 2009.

[35] Katrin Lang and Marc Alexa. The markov pen: online synthesis of free-hand drawing styles. In *Proceedings of the workshop on Non-Photorealistic Animation and Rendering*, pages 203–215. Eurographics Association, 2015.

[36] Jason Lawrence and Thomas Funkhouser. A painting interface for interactive surface deformations. *Graphical models*, 66(6):418–438, 2004.

[37] lens-flare.de. Cumulus, August 22, 2010. [Online; accessed March 20, 2017].

[38] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[39] Paul Merrell and Dinesh Manocha. Example-based curve synthesis. *Computers & Graphics*, 34(4):304–311, 2010.

[40] Gavin SP Miller. The definition and rendering of terrain maps. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 39–48. ACM, 1986.

[41] R. Miyazaki, S. Yoshida, Y. Dobashi, and T. Nishita. A method for modeling clouds based on atmospheric fluid dynamics. In *Proceedings Ninth Pacific Conference on Computer Graphics and Applications. Pacific Graphics 2001*, pages 363–372, 2001.

[42] F Kenton Musgrave, Craig E Kolb, and Robert S Mace. The synthesis and rendering of eroded fractal terrains. In *ACM Siggraph Computer Graphics*, volume 23, pages 41–50. ACM, 1989.

[43] Fabrice Neyret. Qualitative simulation of convective cloud formation and evolution. In *Computer Animation and Simulation97*, pages 113–124. Springer, 1997.

[44] Fabrice Neyret. A Phenomenological Shader for the Rendering of Cumulus Clouds. Research Report RR-3947, INRIA, May 2000.

[45] T. Nishita and Y. Dobashi. Modeling and rendering methods of clouds. In *Proceedings. Seventh Pacific Conference on Computer Graphics and Applications (Cat. No.PR00293)*, pages 218–219, 326, 1999.

[46] Tomoyuki Nishita, Yoshinori Dobashi, and Eihachiro Nakamae. Display of clouds taking into account multiple anisotropic scattering and sky light. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 379–386. ACM, 1996.

[47] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[48] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[49] Adrien Peytavie, Eric Galin, Jérôme Grosjean, and Stéphane Merillou. Arches: a framework for modeling complex terrains. In *Computer Graphics Forum*, volume 28, pages 457–467. Wiley Online Library, 2009.

[50] Brennan Rusnell, David Mould, and Mark Eramian. Feature-rich distance-based terrain synthesis. *The Visual Computer*, 25(5):573–579, 2009.

[51] Joshua Schpok, Joseph Simons, David S Ebert, and Charles Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 160–166. Eurographics Association, 2003.

[52] Peter Shirley, Michael Ashikhmin, and Steve Marschner. *Fundamentals of computer graphics*. CRC Press, 2015.

[53] Ruben M Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. A survey on procedural modelling for virtual worlds. In *Computer Graphics Forum*, volume 33, pages 31–50. Wiley Online Library, 2014.

[54] Karin Sobottka and Ioannis Pitas. Segmentation and tracking of faces in color images. In *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 236–241. IEEE, 1996.

[55] Anders Soderlund. Procedural modeling of rocks, 2015.

[56] Söring. Eyjafjallajkull eruption (image 1), May 8, 2010. [Online; accessed March 23, 2017].

[57] Söring. Eyjafjallajkull eruption (image 2), May 8, 2010. [Online; accessed April 05, 2017].

[58] Söring. Eyjafjallajkull eruption (image 3), May 8, 2010. [Online; accessed April 05, 2017].

[59] Romain Soulié, Stéphane Mérillou, Olivier Terraz, and Djamchid Ghazanfarpour. Modeling and rendering of heterogeneous granular materials: granite application. In *Computer Graphics Forum*, volume 26, pages 66–79. Wiley Online Library, 2007.

[60] Szymon Stachniak and Wolfgang Stuerzlinger. An algorithm for automated fractal terrain deformation. *Computer Graphics and Artificial Intelligence*, 1:64–76, 2005.

[61] Jos Stam. Stochastic rendering of density fields. In *Proceedings of Graphics Interface*, pages 51–58, Banff, Alberta, May 1994.

[62] Marc Stiver, Andrew Baker, Adam Runions, and Faramarz Samavati. Sketch based volumetric clouds. In *International Symposium on Smart Graphics*, pages 1–12. Springer, 2010.

[63] Andrzej Trembilski and Andreas Broßler. Surface-based efficient cloud visualisation for animation applications. In *WSCG*, pages 453–460. Citeseer, 2002.

[64] Jamie Wither, Antoine Bouthors, and Marie-Paule Cani. Rapid sketch modeling of clouds. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, pages 113–118. Eurographics Association, 2008.

[65] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM, 1996.

[66] Tatsuo Yanagita and Kunihiko Kaneko. Modeling and characterization of cloud dynamics. *Physical Review Letters*, 78(22):4297, 1997.

[67] H. Zhou, J. Sun, G. Turk, and J. M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, July 2007.

Appendix A

Snaxels in 3D: Mesh Growth Using Snaxels

In previous chapters we presented our algorithm and results on modeling cumulus clouds and volcanic ash-cloud and similar bodies by using snaxels and growing them on a tessellation in 2 dimensions. Perhaps our approach can be used to grow objects in 3 dimensions. But to investigate that, first we need to be able to represent and handle growth of snaxels in 3 dimensions. In this chapter we will describe what we have investigated so far in growing snaxels in 3 dimensions. We will describe the obstacles we faced by the different approaches we used and present them for future investigation and researchers.

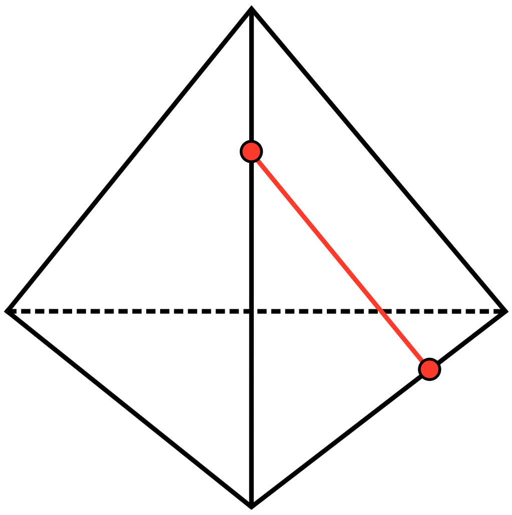
First we are going to discuss the needed elements to transition from 2 dimensional representation of snaxels to 3 dimensional. We will describe the differences and where the difficulties are for handling the events. Next we will present our approaches which we tried and discuss them.

A.1 Transition

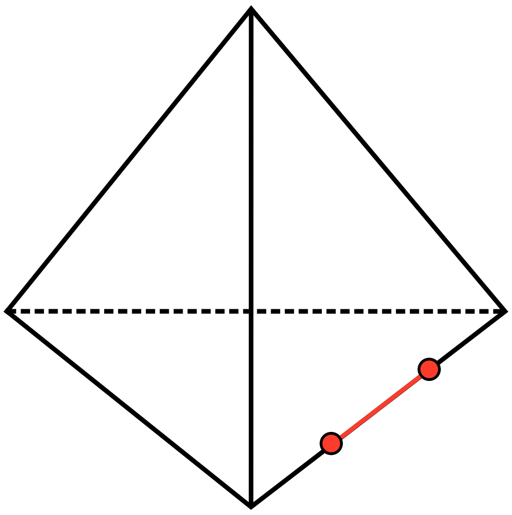
Snaxels propagate on a triangulation in 2 dimension. In 3 dimension it turns into a tetrahedral tessellation and the growing boundary is a water-tight mesh. one dimension lesser than the tessellation it grows on. The growing mesh needs the same constraints as the 2 dimensional version. As discussed, snaxels propagate through the edges of the tessellation and the connections between snaxels will be inside or on the exterior surface of each tetra or tessellation shape. Figure A.1 shows the different types of connections allowed between snaxels. These connections can be categorized as three different types depending where they lie on the host tetra. As discussed in chapter 3, snaxels go through some events as they propagate through the tessellation. In 3 dimensions, the snaxels go through similar events to the ones in 2 dimensions as discussed in chapter 3. However birth and merge require re-stitching and therefore, differ with the 2 dimensional version in how they are handled.

Figure A.2 shows a part of the mesh, where a snaxel on the mesh goes through a birth event. The color green indicates the outside surface of the mesh and the color red indicates the inside surface of the mesh. Birth event, as shown in the figure, creates a cap of children snaxels. The cap is a segment of the mesh formed by connecting the children born in the birth event. The cap is generated by triangulating each 3 children residing on a tetra. And since there is always three children or less on a tetra after a birth event, this action will result in the creation of the cap. With the parent removed and the children connected, two rims are created which need to be stitched to one another. One is the children on the boundary of the cap, and the other is the rim snaxels of the rest of the mesh, which were previously connected to the parent snaxel. The handling of this event is finished by connecting the snaxels of the two rims.

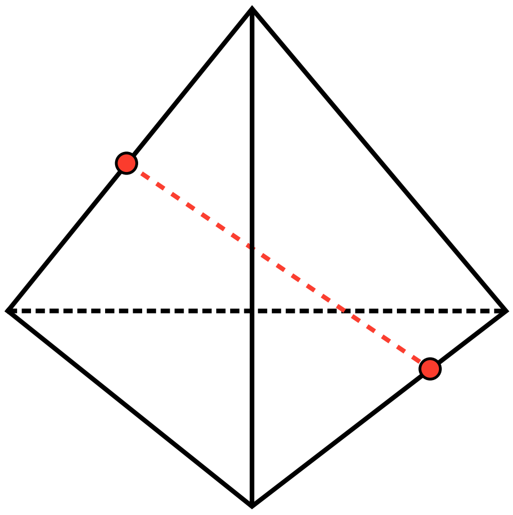
However re-stitching after birth is simply connecting children on the boundary of the cap to rim snaxels. The difficulty is in re-stitching after a merge event. We need to be able to identify the type of the hole generated and successfully patch the hole to complete the water tight mesh. Since a merge event has more variety due to the number and position of neighboring snaxels of the two snaxels involved, the handling of the event becomes more challenging. In fact, the re-stitching needed in a birth event can also be seen after a merge event. Therefore, by focusing on handling a merge event, the birth event is also handled. In the next section we will describe our approaches to tackle this.



(a) A connection lying on a triangular face of the host tetra

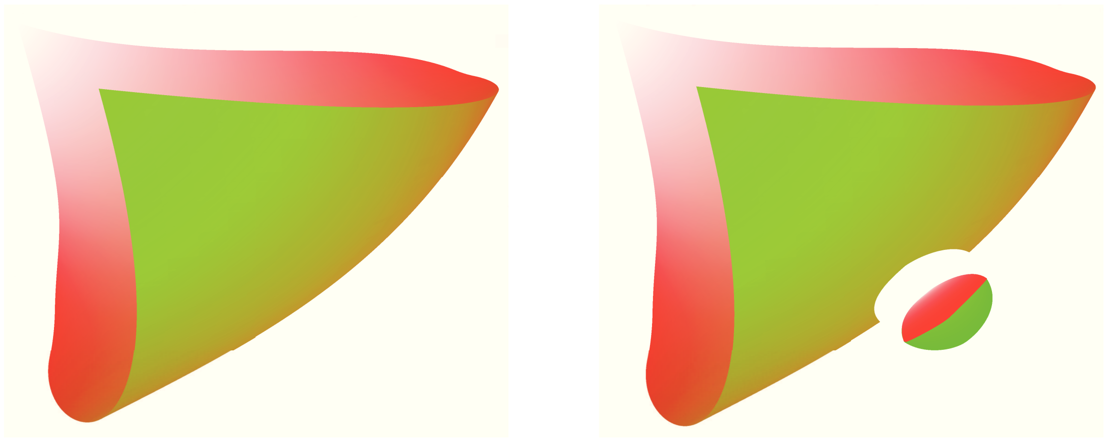


(b) A connection lying on an edge of the host tetra



(c) A connection going through the host tetra

Figure A.1 Different types of connections between snaxels inside tetrahedras. The dashed edges indicate that they are occluded.



(a) Before the event (b) After the event

Figure A.2 Before and after a birth event on a mesh. The mesh before the birth event (a), the parent is removed and the children have formed a cap (b). Note that only a part of the mesh is visualized.

A.2 Case Based Approach

As mentioned in the previous section, in the event of a merge, a re-stitching of the neighboring snaxels of the two snaxels involved in the merge is needed. To approach this problem, we first analyzed each tetra of the neighborhood of the event. Figure A.3 show the top view of the cluster of tetras in the neighborhood of a merge event which is about to happen. As illustrated, the cluster of tetras are captured by taking all the tetras sharing the edge on which the merge is happening. The relationship between the snaxels connected to the merging snaxels can be of three categories only; those connected to the first snaxel, those connected to the second snaxel and those which are shared and connected to both. They are denoted 'N_1', 'N_2' and 'S' respectively.

One approach to stitching is to analyze each tetra by the position of snaxels on its edges and their relationship with the snaxels which merged with one another. However, the relationships between the tetras is largely ignored in this approach. Even if we account for the relationships the space of varieties of type of tetras and how they should be handled simply just gets larger. This space is large enough without accounting for the relationships. Another issue is the number of ways each tetra can be stitched. Some tetras can be stitched in separate ways. For example,

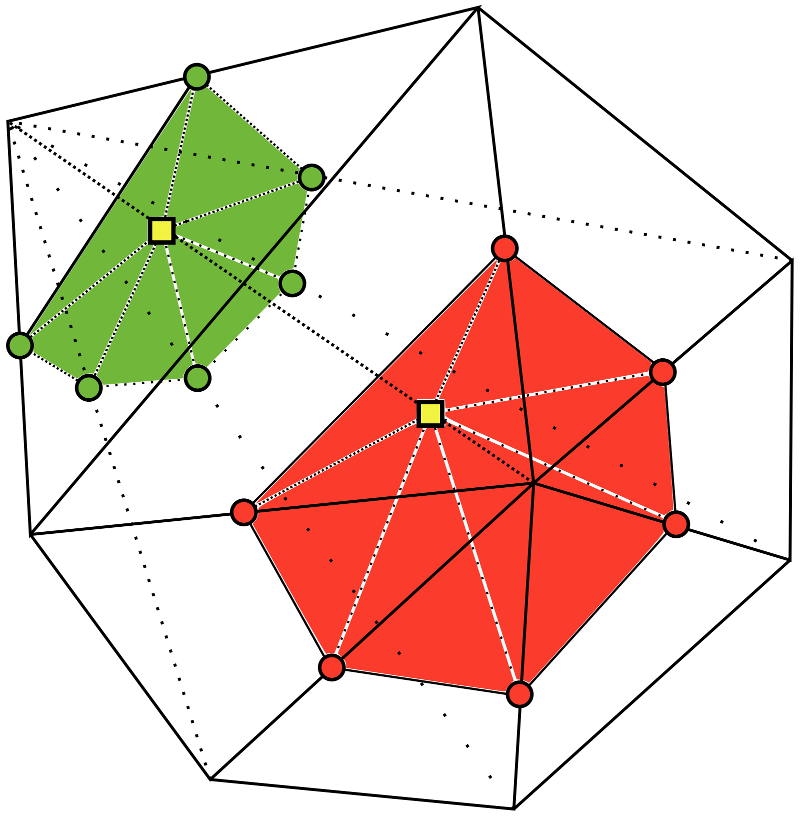


Figure A.3 Before a merge event. The color green indicates the outside surface of the mesh and the color red indicates the inside surface of the mesh.

Figure A.4 shows a tetra in which two snaxels marked as squares, merge with one another and leave behind a structure with a few boundary edges which need stitching. The figures A.5(a) and A.5(b) illustrate two possibilities for stitching.

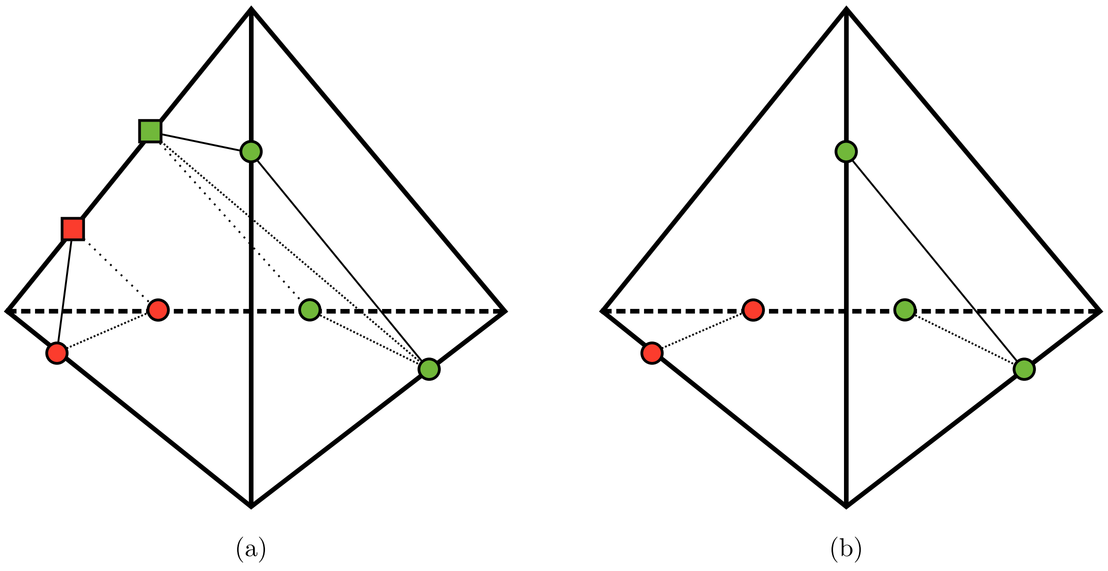


Figure A.4 Before and after a merge event inside a tetra

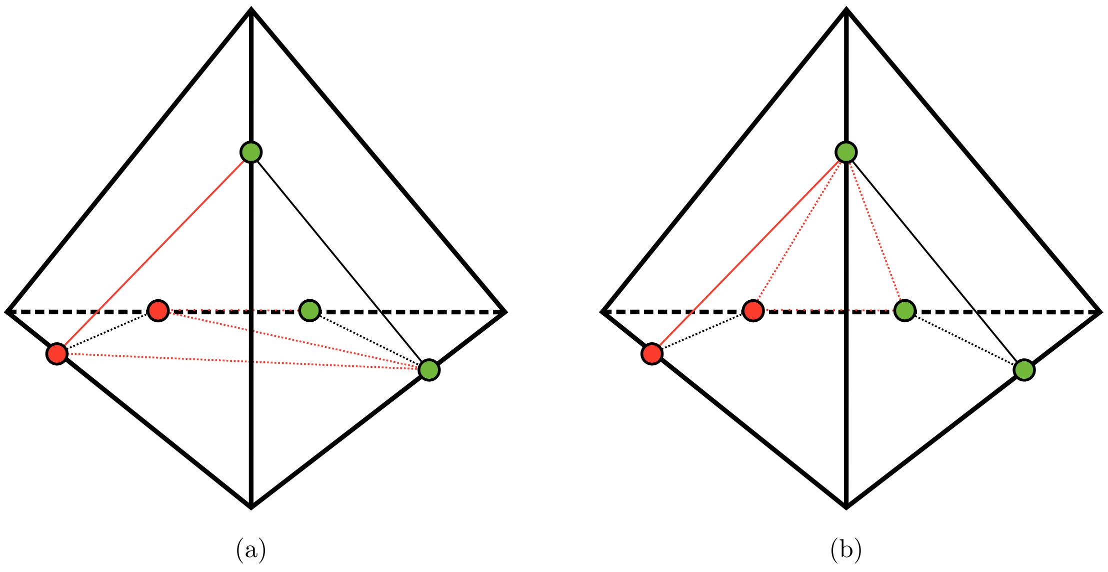
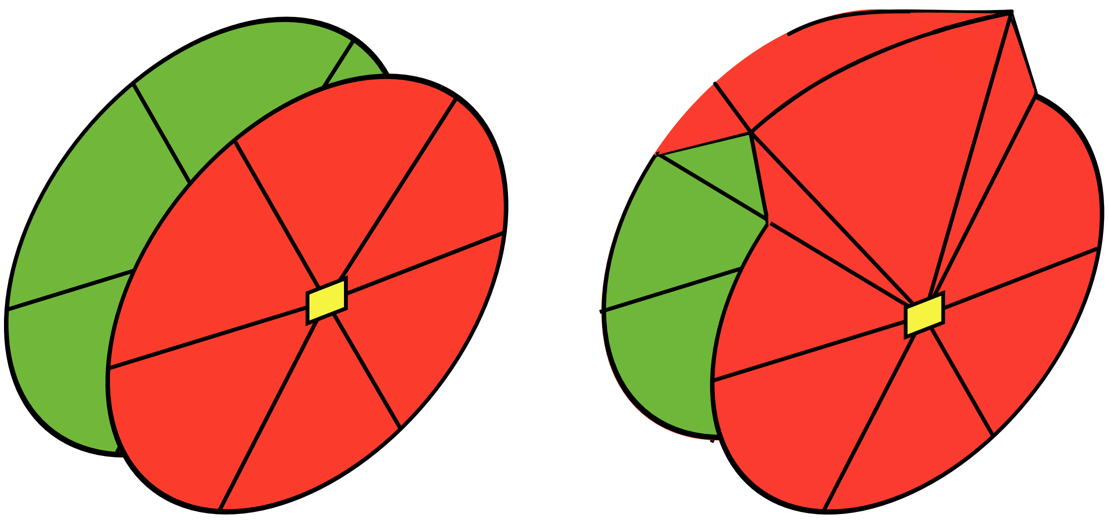


Figure A.5 Example of a re-stitching problem

A.3 Hole Patching Approach

The issues discussed in the previous section, lead us to analyze the hole generated in the mesh instead of each tetra in the cluster around the edge on which the merge event has taken place. Once the merging snaxels are removed from the mesh, depending on the connection between them, one or two holes are generated on the surface of the mesh. Figure A.6 shows the same merge event happening in Figure A.3 in two possible scenarios. Consider each disk to represent one of the merging snaxel plus the triangles connected to it. This makes the circumference of the disk to be the neighboring snaxels of the merging snaxel. As shown, there are two possible scenarios. One scenario shown in Figure A.6(a) is when the two merging snaxels do not share any neighbors. This scenario also indicates that the merging snaxels are not connected to one another. The other scenario shown in Figure A.6(b) is when the merging snaxels share some of their neighboring snaxels. In this case they may be connected to each other. Figure A.7 is an attempt to illustrate two different holes generated in the mesh after the removal of merging snaxels in Figure A.6. Red dashed lines and red and green colors indicate the surface of the mesh.



(a) (b)

Figure A.6 Two different scenarios for a merge event. The color green indicates the outside surface of the mesh and the color red indicates the inside surface of the mesh. Merging snaxels are illustrated by yellow squares.

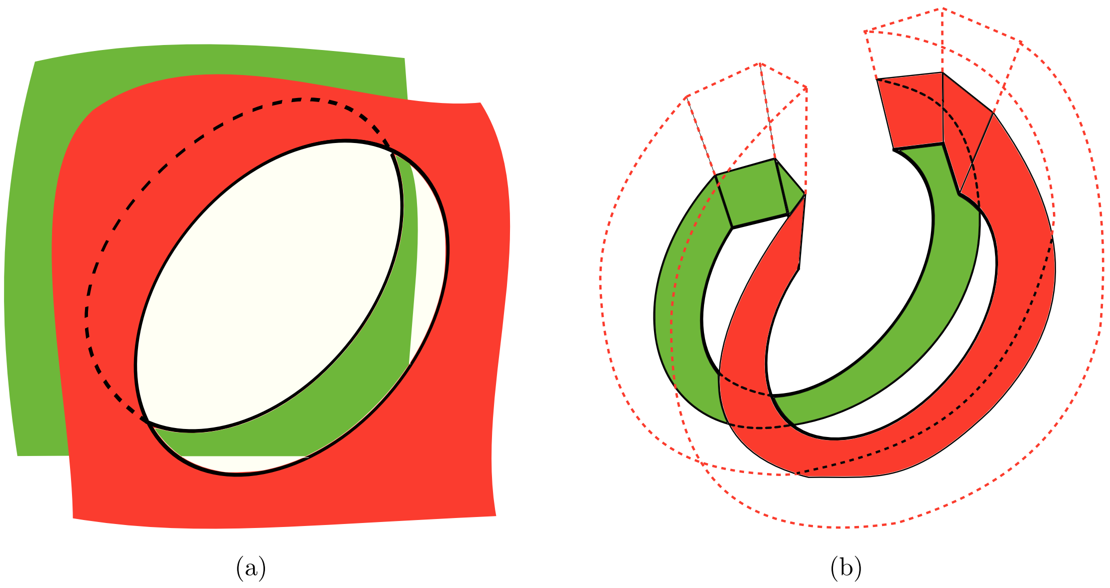
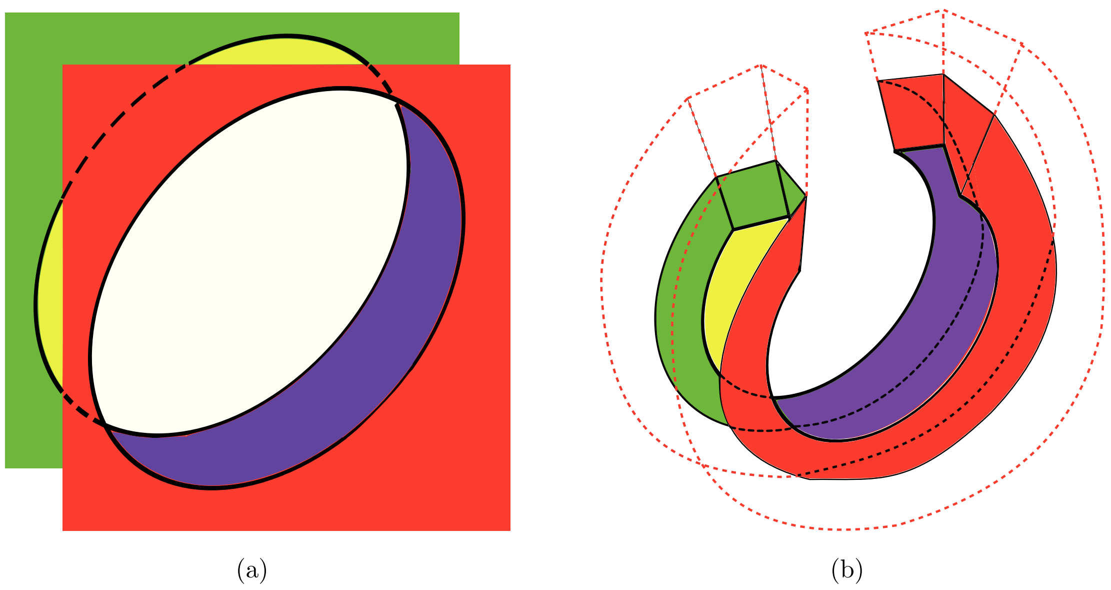


Figure A.7 Two different types of holes. The color green indicates the outside surface of the mesh and the color red indicates the inside surface of the mesh.

Since the hole shown in Figure A.7(a) is convertible to the one shown in Figure A.7(b) by stitching just one or two snaxels from each rim to one another, our focus is to stitch the one shown in Figure A.7(b) by forming a patch shown in Figure A.8. The color yellow indicates the outside surface of the patch and the color purple indicates the inside surface of the patch. To form the patch, we want to pursue an incremental approach to stitching. We start by a shared neighbor of the merging snaxels and using the snaxels at the other end of its boundary edges. First we perform a test to see if all three of them can form a triangle. That means they should all be inside a tetra and should not be connected to each other. Then we proceed to stitch all three together, forming a new triangle and thus forming a new boundary edge. After that, we take the newly formed boundary edge and check to see if we can repeat the process on either of its snaxels. By incrementally repeating this process, we plan to stitch the entire hole.

The approach discussed above is supposed to stitch a given hole. However this implies that the hole does not already have snaxels on its boundary which are connected to one another. If there was already snaxels connected to one another, the incremental stitching algorithm would not be able to complete the stitching. Since two already connected snaxels means that the algorithm will not be able to fit in a patch. If the two snaxels which share an edge were to be used in forming a new triangle, it would violate the mesh's structure. Our focus is to capture such scenarios and handle them properly so that the stitching can be completed.



(a) (b)

Figure A.8 Stitched holes with a patch.