



## Assignment of bachelor's thesis

<b>Title:</b>	Dynamic texture modelling and editing
<b>Student:</b>	Alžběta Mrkvová
<b>Supervisor:</b>	Ing. Radek Richtř, Ph.D.
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Web and Software Engineering, specialization Computer Graphics
<b>Department:</b>	Department of Software Engineering
<b>Validity:</b>	until the end of summer semester 2022/2023

### Instructions

Dynamic textures are textures that evolve over time in some predictable way (they have both spatial and temporal homogeneity).

Typical examples of dynamic textures are clouds.

- 1) Perform research for the topic of clouds, their types and behavior.
- 2) Perform research for dynamic textures, cloud generation, and procedural modeling.
- 3) Analyze existing methods and select the appropriate one.
- 4) Design and implement a cloud generation prototype.
- 5) Compare the results of your generator with existing results and real clouds.





**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

Bachelor's thesis

**Dynamic texture modelling and editing  
Modelování a editace dynamických textur**

*Alžběta Mrkvová*

Department of Web and Software Engineering

Supervisor: Ing. Radek Richtr, Ph.D.

May 12, 2021



---

## **Acknowledgements**

I would first like to thank my supervisor Ing. Radek Richtr, Ph.D. who answered my constant questions and who gave me plenty of advice regarding the thesis. I would also like to thank my family for their great support on all fronts.



---

## Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 12, 2021

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2021 Alžběta Mrkvová. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

### **Citation of this thesis**

Mrkvová, Alžběta. *Dynamic texture modelling and editing*  
*Modelování a editace dynamických textur*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.



---

# Abstrakt

Mraky v přírodě se dynamicky mění ve větru a tuto vlastnost chceme vystihnout v počítačové grafice. Ve své bakalářské práci jsem provedla průzkum na téma mraků, existujících řešení jejich simulace a návrh generátoru mraků jako plugin do Blenderu. Tento plugin pomocí buněčného automatu vytváří volumetrické mraky v klíčových snímcích vhodných pro následné vykreslení.

**Klíčová slova** mraky, dynamické textury, simulace, animace, buněčný automat, Blender

---

# Abstract

Clouds are dynamically changing in the wind and it is desired to mirror this characteristic in computer graphics. In my thesis, I conduct research on the topic of clouds and the existing approaches to their simulation. I propose an implementation of a cloud generator as a Blender plugin. The plugin generates volumetric clouds using cellular automaton and produces a keyframed animation ready for rendering.

**Keywords** clouds, dynamic textures, simulation, animation, cellular automaton, Blender



---

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Goals and objectives</b>	<b>3</b>
<b>2 Research</b>	<b>5</b>
2.1 What are clouds . . . . .	5
2.1.1 Creation of cloud . . . . .	5
2.1.2 Cloud's colours . . . . .	5
2.2 Types of clouds . . . . .	6
2.3 Clouds in computer graphics . . . . .	7
2.3.1 Computer games and films . . . . .	7
2.3.2 Used concepts . . . . .	9
2.3.3 Cellular automaton . . . . .	11
2.3.4 Coupled map lattice . . . . .	13
2.3.5 Particle systems . . . . .	13
2.3.6 Doretto's dynamic textures . . . . .	15
2.3.7 Richtr's dynamic textures . . . . .	15
2.4 Fluid dynamics . . . . .	16
2.4.1 Navier-Stokes equations . . . . .	16
2.4.2 Stable fluids . . . . .	17
<b>3 Analysis and design</b>	<b>19</b>
3.1 Generator . . . . .	19
3.1.1 Cellular automaton . . . . .	19
3.1.2 Coupled map lattice . . . . .	20
3.1.3 Chosen method . . . . .	21
3.2 Visualization . . . . .	21
3.2.1 OpenGL . . . . .	21
3.2.2 Blender . . . . .	21
3.3 Design . . . . .	23

3.3.1	Requirements . . . . .	23
3.3.2	Use cases . . . . .	23
<b>4</b>	<b>Realisation</b>	<b>25</b>
4.1	Chosen method . . . . .	25
4.2	First results . . . . .	25
4.3	First attempts in Blender . . . . .	28
4.4	Blender scripting . . . . .	28
4.4.1	Script sketch . . . . .	28
4.4.2	Initialization of cloud . . . . .	29
4.4.3	Growth of cloud . . . . .	29
4.4.4	Movement of clouds . . . . .	29
4.4.5	Representation of data . . . . .	30
4.4.6	Colours . . . . .	31
4.4.7	Cloud texture . . . . .	31
4.4.8	Technical problems . . . . .	32
4.5	Plugin parameters . . . . .	33
<b>5</b>	<b>Testing</b>	<b>35</b>
5.1	Personas . . . . .	35
5.2	Testing scenarios . . . . .	35
5.3	Conclusion of testing . . . . .	36
<b>6</b>	<b>Results</b>	<b>37</b>
6.1	Version 1.x . . . . .	37
6.2	Version 2.x . . . . .	40
6.3	Version 3.x — Plugin . . . . .	41
6.4	Render speed . . . . .	45
6.5	Comparison of clouds . . . . .	48
	<b>Conclusion</b>	<b>49</b>
	<b>Bibliography</b>	<b>51</b>
	<b>A Acronyms</b>	<b>55</b>
	<b>B Contents of enclosed CD</b>	<b>57</b>

---

## List of Figures

2.1	Classification of clouds . . . . .	6
2.2	The Long Dark (game 2017) . . . . .	8
2.3	Horizon Zero Dawn (game 2017) . . . . .	8
2.4	Castle in the Sky (film 1986) . . . . .	8
2.5	Definition of metaball . . . . .	9
2.6	The rule 30 automaton after 15 steps starting with a single black cell . . . . .	10
2.7	Transition rules . . . . .	12
2.8	Simulation of Bénard cells . . . . .	14
2.9	Dynamic texture examples . . . . .	16
3.1	Animation of natural scenery . . . . .	20
3.2	Use case diagram . . . . .	24
4.1	Static cloud from initial state . . . . .	26
4.2	Animation of cloud CA in Wolfram Mathematica . . . . .	26
4.3	Visualization of cloudy neighbours . . . . .	27
4.4	Colourful sunset . . . . .	31
4.5	Blue sky . . . . .	32
6.1	Version 1.1, moving light with cloud . . . . .	38
6.2	Version 1.2 with and without shadows . . . . .	39
6.3	Version 2.1, radius of spheres = 2 . . . . .	40
6.4	Version 2.06 . . . . .	40
6.5	Version 3.12 . . . . .	41
6.6	Version 3.14, sun from behind . . . . .	42
6.7	Version 3.14, more clouds in scene . . . . .	43
6.8	Version 3.14, growth + movement . . . . .	44
6.9	Version 3.14, particles per cloud = 64 . . . . .	45
6.10	Version 3.14, particles per cloud = 512 . . . . .	46
6.11	Version 3.14 . . . . .	47



---

## List of Tables

4.1	Effect of increasing the size of cloud noise texture: bottom right quadrant of each volume is an example of used noise texture . . . . .	32
4.2	Amount of particles . . . . .	33
4.3	Displacement degree . . . . .	33
4.4	Density . . . . .	34





---

# Introduction

Clouds are natural part of the world. Wherever we found ourselves on the planet it is almost impossible to avoid them – for sky without clouds is not really common.

Therefore, we are expecting clouds maybe even unconsciously in every natural scene: on our Saturday outing when we are enjoying the landscape, in a moment captured by professional photographer, or even in a quick snapshot with our friends during a trip.

The sky is part of even the film industry and the worlds of computer games. Sky full of clouds, grey tumultuous giants, or peaceful little lambs. Clouds are ever-changing and create myriads of interesting shapes in the sky, and similarly captivating is the interplay of their shadows underneath.

That is why generation (and also simulation and animation) of clouds is one of the most important, not to mention interesting, problem in computer graphics. Nature without clouds would be missing something. . . After all, you can't have bad rainy weather without large gloomy clouds. And sunset would be a lot more boring without them.

Clouds always fascinated me. I decided to learn more about generating clouds in computer graphics and tried implementing a solution to this problem.

In theoretical part of my thesis I will introduce existing approaches to cloud generation. I will compare several existing solutions and choose one for my implementation.

The practical part will describe my solution of using cellular automaton as a base for cloud generator. The first attempts that were made using C++ and Wolfram Mathematica as a proof of working concept and rudimentary visualization. The later recreation of the cloud generator as a Blender plugin that allowed for addition of modelling and simulation using Blender API.

The practical part will be concluded by the comparison of my results with real clouds and existing solutions and discussion of my primary objectives.



# Goals and objectives

The main objectives of this thesis are:

- to perform research on the topic of clouds, their types and behaviour
- to perform research on cloud generation, procedural modelling and dynamic textures
- to analyze existing methods and select the appropriate one
- to design and implement a cloud generation prototype
- to compare the results of my generator with existing results and real clouds



---

## Research

### 2.1 What are clouds

Clouds are visually distinct large masses of minute water droplets and ice crystals suspended in the atmosphere. They can also include solid particles that can be found in fumes, smoke or dust.

#### 2.1.1 Creation of cloud

There is number of mechanisms that lead to a formation of a cloud. What they have in common is that cloud (or fog) is formed when air is saturated with water vapor. For air to reach the saturation point, the temperature either has to drop so it can hold lower amount of moisture, or the vapor accumulates until the air can hold no more water.

One of the common examples of formation of clouds is uplift of body of air due to planet's surface heating followed by convection. As the air rises, its temperature and saturation point drops so the moisture in the air becomes visible in the form of clouds.

There are many artificial clouds such as *contrails*<sup>1</sup> and clouds produced by cooling towers of thermal power stations as a part of landscape.

#### 2.1.2 Cloud's colours

Clouds are beautiful. They create complex shapes in the sky that engage the imagination of the observer. Not only their shapes are interesting but also their colour. Colour of the cloud depends mainly on the position of the sun in the sky. Almost the whole colour palette can be found in the sky and clouds throughout the day and night.

From moments before sunrise the sky and the clouds are painted yellow and red by the rising sun. Midday the clouds appear white or gray with the Sun high above

---

<sup>1</sup>Short for condensation trails, line-shaped clouds created by aircraft flying at high altitude.

## 2. RESEARCH

---

the horizon. The sunset colours are tinged red. Higher positioned clouds remain white but those in lower heights take the reddish colour of the sky.

The night clouds can be illuminated by the moon or from below them either by moonlight reflected off of frozen lake, desert sand or by bustling cities to name a few examples. On an overcast night the moon and the stars are covered which results in gray and black clouds.

### 2.2 Types of clouds

There is a great diversity of clouds and multiple systems that categorise them.

One of the most used categorizations is by altitude (Figure 2.1).

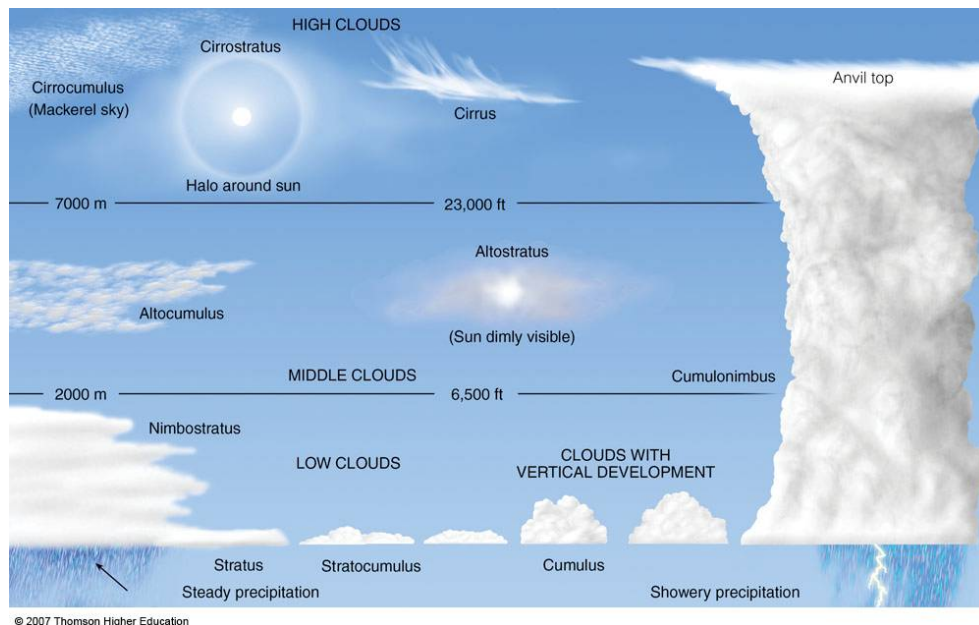


Figure 2.1: Classification of clouds [27]

Low clouds are located up to 2000 metres above ground. *Nimbostratus*, *stratocumulus*, *cumulus* and *cumulonimbus* clouds are included in this category. Middle clouds are up to 7000 metres above ground. We can find *altocumulus* and *altostratus* in this category. High clouds are *cirrus*, *cirrocumulus* and *cirrostratus*.

These are only the core types of clouds. However, there exists an array of varieties to every previously mentioned cloud. Another types of clouds are for example *vertebratus* (riblike patterned cloud), *undulatus* (billow cloud), *lacunosus* (cloud full of holes), *translucidus* (transparent cloud) or *perlucidus* (cloud allowing the passage of light).

*Nimbostratus* and *cumulonimbus* are clouds that most often produce rain. *Nimbus* in their names means rainstorm in Latin. Rain clouds appear darker because they

consist of larger droplets that obscure sunlight.

*Stratus* is Latin word for layer. *Stratus* clouds develop in flat horizontal formations that often cover the whole sky. Low-hanging *stratus* clouds often bring rain, *stratus* cloud located in higher altitudes produce snow.

### 2.3 Clouds in computer graphics

In computer graphics clouds are important subject. That is why there are multiple approaches to generating and simulating clouds, different methods are more suitable for different uses. Some methods are more efficient or produce more realistic looking results.

There are solutions that are able to produce only static images, but with over forty years<sup>2</sup> of interest of many scientists and graphics programmers there are also plentiful solutions regarding dynamic clouds. Many of these approaches can be considered as dynamic textures as stated Haindl [24, 23] and Richttr [31, 26, 30] because they evolve in time according to a set of defined rules.

One of the classifications<sup>3</sup> [16] of clouds in computer graphics:

**Physics-based solutions** simulate the physical process of fluid dynamics. Most methods require a large amount of computational time. Main difference lies in used solvers leading to stability/instability.

**Heuristic solutions** are computationally inexpensive but many parameters need to be found by trial-error method. Heuristic solutions are using fractals, qualitative simulation, stochastic modelling or procedural modelling.

**Image-based solutions** use sets of images for learning. Static images from satellites are used for modelling clouds using metaballs. Video sequences are used for generating infinite textures called dynamic textures.

#### 2.3.1 Computer games and films

It is essential to be able to smoothly display natural phenomena in various animations. The approaches can vary dramatically as illustrated by following examples:

**The Long Dark** game (2017) used smaller sprites moving within the skybox as clouds (Figure 2.2). The sprites for the clouds seem like painted by brush with water colours.

**Horizon Zero Dawn** (2017) is a game with eye-catching graphics. The studio needed convincingly looking evolving clouds in the sky. They were successful with their solution of real-time volumetric clouds (Figure 2.3) which can simulate many types of clouds, e.g. *cirrus* and *cumulus* clouds.

---

<sup>2</sup>For example Csuri's article [1] about interactive animations from 1979.

<sup>3</sup>Doretto's dynamic textures [18, 19] were added to this system.

## 2. RESEARCH

---



Figure 2.2: The Long Dark (game 2017) [29]



Figure 2.3: Horizon Zero Dawn (game 2017) [28]



Figure 2.4: Castle in the Sky (film 1986) [5]



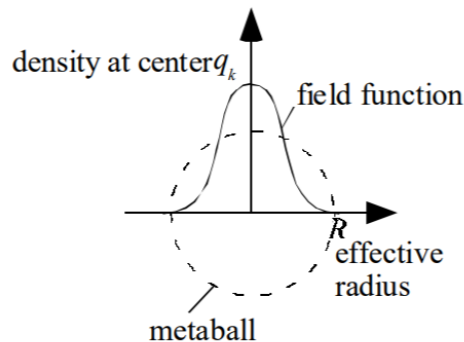


Figure 2.5: Definition of metaball [12]

**Castle in the Sky** (1986) is studio Ghibli's first film. Although it is not an example of computer graphics as it is a hand-drawn film, it is good to remember classical approaches to animation. As the name suggests the film is situated in the sky on a flying island called Laputa (Figure 2.4). The director Hayao Miyazaki, an enthusiastic fan of aircraft, likes to include planes and flying in his animations.

### 2.3.2 Used concepts

**Metaballs** also known as soft objects are solids defined by scalar fields. They are used to model free-form surfaces. Metaballs often need hundreds of data points and it is important to be able to calculate the value of a specific point without having to refer to all of them. For this purpose field functions are used in the metaballs.

Field function is guaranteed to have only a local effect of the metaball. To achieve local control, field function and its derivative must drop to zero at some distance  $R$  called effective radius (or radius of influence). The function is calculated from a set of key points which form a skeleton of the object. Each point can be considered as a source of energy (or density). The energy decreases with increasing distance from the centre of the object (as described in Figure 2.5).

The concept of metaballs and field functions was first introduced in a paper by Wyvill et al. [7].

**Cellular automaton (CA)** is a collection of coloured cells in a grid. The simplest type of grid is one-dimensional but more dimensions are possible. The cells evolve through discrete time steps according to a set of rules that take into consideration states of neighbouring cells. The neighbourhood has to be specified prior to the simulation (Figure 2.6).

In 1984 an article by Wolfram [4] was published describing universality of CA. He included many examples of configurations and images of generated results with detailed analysis of performed steps.

## 2. RESEARCH

---

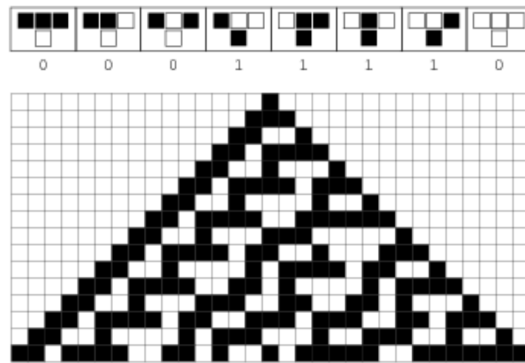


Figure 2.6: The rule 30 automaton after 15 steps starting with a single black cell [17]

**Coupled map lattice (CML)** is an extension of cellular automaton [16]. It was first used by Kaneko in 1990 [6] for simulating physics. The simulation space of CML is subdivided into lattices. Each lattice has several state variables and their status is updated according to the neighbouring lattices. CML uses real-value variables while CA uses discrete variables.

### 2.3.3 Cellular automaton

Dobashi et al. proposed a method for making animation of clouds using the cellular automaton in their 1998 paper [12] and further developed their findings two years later in [15].

Dobashi et al. [12] followed upon the numerical model by Nagel et al. [8] for simulating the growth of clouds using cellular automaton. The space is divided into a three-dimensional grid. For each point (cell) in the grid there are three binary variables: humidity<sup>4</sup> *hum*, cloud *cld* and transition phase<sup>5</sup> *act*.

Since the state is either 0 or 1, the transition rules can be represented as Boolean operations. However, in Nagel's method there were several disadvantages: once *cld* is 1 it remains 1 and the cloud never disappears; the simulation output is binary distribution therefore realistic images cannot be generated.

Dobashi et al. decided to remedy [8] disadvantages and address them as follows: new variables for cloud extinction *ext* and its transition rules are added so the clouds can disappear. For generating more realistic looking clouds continuous distribution using metaballs is calculated in the post process of the simulation.

Metaballs (see 2.3.2) are spheres in which a field function is defined. Each metaball has two parameters: density at the center and effective radius. Metaballs are placed at each point of the grid and the continuous distribution is represented as weighted sum of the field functions. Metaballs are used to obtain the continuous distribution.

Complicated cloud motion can also be simulated by method described in [12]. At every frame the variables are checked if their value is 0. The variables with status 0 are flipped to 1 by random numbers that obey probability function. Animator can control the motion by specifying different probabilities at each grid point at different times.

The newer model [15] is able to cast shadows on the ground, render shafts of light through clouds and clouds can be animated with directional movement.

The whole simulation is computed quickly using graphical hardware through OpenGL. The rendering method is based on splatting algorithm. It is a fast algorithm using billboards placed in the centres of the grid's metaballs. Each billboard has a precomputed texture mapped on it, the positions of colour shades on the texture are based on summed densities of the cloud's metaballs from viewer.

Transition rules<sup>6</sup> used in the growth simulation are as follows:

$$\begin{aligned}
 hum(i, j, k, t_{i+1}) &= hum(i, j, k, t_i) \wedge \neg act(i, j, k, t_i) , \\
 act(i, j, k, t_{i+1}) &= \neg act(i, j, k, t_i) \wedge hum(i, j, k, t_i) \wedge fact(i, j, k) , \\
 cld(i, j, k, t_{i+1}) &= cld(i, j, k, t_i) \vee act(i, j, k, t_i) ,
 \end{aligned} \tag{2.1}$$

<sup>4</sup>Humidity – is there enough vapor to form clouds?

<sup>5</sup>Transition phase – water vapor becomes water droplets or cloud. *act* = 1 means that transition phase is ready to occur.

<sup>6</sup>The rules are taken from [8].

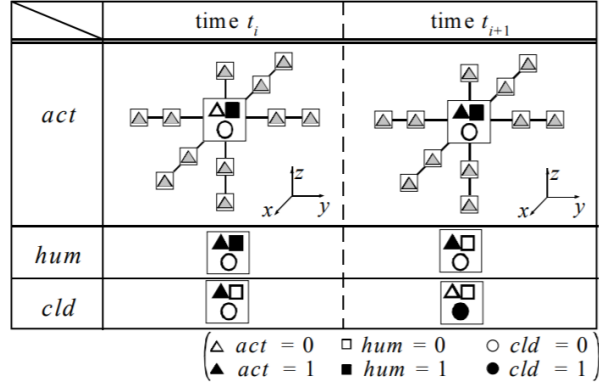


Figure 2.7: Transition rules [15]

where  $f_{act}$  is also a Boolean function, its value is calculated by the status of *act* around the cell:

$$\begin{aligned}
 f_{act}(i, j, k) = & act(i+1, j, k, t_i) \vee act(i, j+1, k, t_i) \vee act(i, j, k+1, t_i) \\
 & \vee act(i-1, j, k, t_i) \vee act(i, j-1, k, t_i) \vee act(i, j, k-1, t_i) \\
 & \vee act(i+2, j, k, t_i) \vee act(i-2, j, k, t_i) \\
 & \vee act(i, j+2, k, t_i) \vee act(i, j-2, k, t_i) \\
 & \vee act(i, j, k-2, t_i) .
 \end{aligned} \tag{2.2}$$

For better understanding of all neighbouring cells participating on extinction and transition phase for one cell look at Figure 2.7.

For stochastic behaviour animator can supply probabilities which are real numbers from interval  $\langle 0, 1 \rangle$ , for extinction  $p_{ext}$ , vapor  $p_{hum}$  and transition phase  $p_{act}$ .

Initial state is

$$\begin{aligned}
 hum &= rand < p_{hum} , \\
 act &= hum \wedge rand < p_{act} , \\
 cld &= 0 .
 \end{aligned} \tag{2.3}$$

At each cell where *cld* is 1 a random number  $rnd \in \langle 0, 1 \rangle$  is generated. Variable *cld* is changed to 0 if  $rnd < p_{ext}$ . Vapor probability  $p_{hum}$  and phase transition probability  $p_{act}$  are used similarly to  $p_{ext}$  to set *hum* and *act* randomly:

$$\begin{aligned}
 hum(i, j, k, t_{i+1}) &= hum(i, j, k, t_i) \vee (rnd < p_{hum}(i, j, k, t_i)) , \\
 act(i, j, k, t_{i+1}) &= act(i, j, k, t_i) \vee (rnd < p_{act}(i, j, k, t_i)) , \\
 cld(i, j, k, t_{i+1}) &= cld(i, j, k, t_i) \wedge (rnd > p_{ext}(i, j, k, t_i)) .
 \end{aligned} \tag{2.4}$$

Animator can design the cloud motion by specifying  $p_{ext}$ ,  $p_{act}$  and  $p_{hum}$ . Ellipsoids are used in this method to simulate the air parcels. When wet air parcels move upward and reach the dew point clouds are gradually formed. The vapor and phase

transition probabilities are assumed to be higher at the center of clouds and cloud extinction probability is higher at their edges. By controlling size and position of ellipsoids different types of clouds can be simulated.

#### 2.3.4 Coupled map lattice

Miyazaki et al. in [16] advanced method for generating clouds by Dobashi et al. [15]. More types of clouds can now be generated: *culumulus*, *cumulonimbus*, *stratocumulus*, *altocumulus*, *cirrocumulus*.

These types of clouds originate due to some factors:

1. viscosity and pressure effect
2. advection by fluid flow
3. diffusion of water vapor
4. thermal diffusion
5. thermal buoyancy
6. phase transition

This method classifies clouds into two categories: those that are created by ascending air currents (*cumulus* and *cumulonimbus*) and those formed by Bénard convection (*stratocumulus*, *altocumulus*, *cirrocumulus*). Bénard convection can be simulated using CML (see 2.3.2). Bénard cells can help to differentiate size and shape of the clouds (Figure 2.8). Large Bénard cells create *stratocumuli*, *cirrocumuli* are generated with smaller Bénard cells.

#### 2.3.5 Particle systems

Another approach to generating dynamic textures are particle systems. The very first attempt for algorithm using particles was done by Csuri et al. in 1979 [1]. They visualized cloud of smoke.

A very detailed description of the concept of particle systems was provided by W. T. Reeves in 1983 [2]. He and his team used the idea for simulating wall of fire in the film *The Wrath of Khan*.

Particle systems use new representation of objects. Objects are clusters of primitive particles which define object volume. The particles are not static entity. New particles are born, they move, change their form, and after their allotted time have passed, they die. Due to their transient nature particle systems are suitable for modelling natural phenomena such as smoke, fire, clouds, water, and even grass.

Objects created with particle systems are not deterministic since its shape and form is not completely specified. Stochastic processes are used to create and to change the properties of the objects.

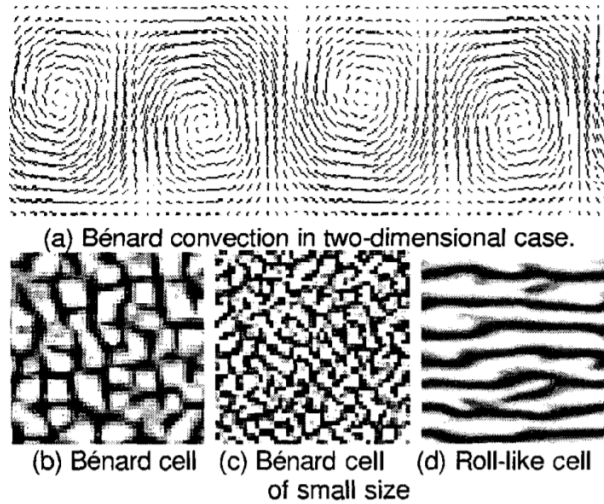


Figure 2.8: Simulation of Bénard cells [16]

An animation using particle systems is as follows: to compute a frame in a motion sequence new particles are generated into the system and they are assigned their initial attributes. The particles that existed over their allotted lifetime are extinguished and the remaining ones are moved and transformed. Finally the image of the particles alive is rendered in a frame buffer.

An animator can manipulate set of parameters to control the shape, appearance and dynamics of the particles. Stochastic processes are constrained by these parameters. Each parameter specifies a range in which a particle's value must lie. Range is defined as its mean value and maximum variance.

The application of the range parameter could be illustrated in the following example featuring simple equation for calculating the number of new particles to be generated at frame  $f$ :

$$N\text{Particles}_f = \text{MeanParticles}_f + \text{Rand}() \cdot \text{VarianceParticle}_f, \quad (2.5)$$

where  $\text{MeanParticles}_f$  and  $\text{VarianceParticle}_f$  is given by animator.  $\text{Rand}()$  is a procedure that returns a uniformly distributed number from  $\langle -1.0, 1.0 \rangle$ .

Speed of particles and other attributes can be computed similarly. Particle motion is achieved by adding its velocity vector to its position vector. For more complex simulation (e. g. such as gravity) acceleration factor can be used.

Particle lifetime is defined as number of frames in which the particle exists. The lifetime is then decremented in each frame. When it reaches zero the particle is killed. Other, more complex mechanisms, can be also put in use in the extinction process. For example if the intensity (which is computed by colour and transparency of the particle) drops below specified threshold it is killed. Another condition can be distance from the system's origin.

In 2010 a scheme for generating three-dimensional clouds in real-time flight simulator [22] was proposed. For this particular purpose very fast modelling and rendering technique was needed. It was built on ideas of Particle systems [2] and the particles used transition rules based on Boolean operations from [15].

### 2.3.6 Doretto's dynamic textures

Another approach to simulating dynamic clouds is called dynamic textures. Doretto et al. devoted their research to this method [19, 18].

Doretto's dynamic textures are based on sequence of images of scenes such as sea-waves, smoke and foliage in the wind.

The textures use image-based rendering technique (IBR). IBR generates synthetic sequences of images without building physical model of the process that generate the scene. Among IBR there are procedural techniques that forego models altogether concentrating on concatenation and repetition of data instead. Another IBR techniques use model of the image sequence itself – model of visual signal. Such models capture mixture of shape, radiance and motion as an image. This approach is called video-based modelling.

Common feature of all IBR techniques is their inflexibility. Editing is a challenge because there is a lack of physical parameters that can be manipulated. This challenge was accepted by Doretto with his team [19].

The proposed system for generating and editing dynamic textures is composed of several modules. Finite video of a dynamic texture and set of values for editing parameters have to be given as input data. The input video is processed by learning module. The module produces a representation of a dynamic texture in the form of a parametric dynamical model.

The next module is editing which allows to interactively modify the temporal characteristics of the simulation. For example, the speed of the texture can be altered or the intensity of a pattern can be changed.

### 2.3.7 Richtr's dynamic textures

There is no exact definition of visual (or multispectral [32]) texture, neither of dynamic (or video, temporal) texture [31, 30]. Dynamic textures (Figure 2.9) are characterized by some spatially invariant statistics (like other visual textures). Additionally to that they require some temporal invariance. Dynamic textures can be represented as a realization of a four-dimensional stochastic random field.

There are two main approaches that can be applied to modelling dynamic textures: a solution built on a mathematical model and a solution based on measured data sampling. Modelling by some mathematical probabilistic model is by far less demanding on memory. However it often produces results with low visual quality. The intelligent sampling methods produce better results but they need to store material patches.



Figure 2.9: Dynamic texture examples [26]

Richtr stated in [30] that three major properties are present in dynamic textures. They define the static texture itself with global and local dynamics. The static texture represents a structure of the scene and its objects. The global motion is describing moving of the whole scene (e. g. rotating or sliding camera) and the local motion dynamics are for small motions, oscillation, overlapping or disappearing of small objects.

Dynamic textures are synthesized by three-dimensional toroidal patch which allows to create realistic results very effectively and in short time. However this approach is very limited and it works well only in some types of dynamic textures, like natural scenes, grass, etc. It is not suitable for modelling clouds.

## 2.4 Fluid dynamics

The following section will focus on topics of Navier-Stokes equations and Stam's Stable fluids [14] that are partly based on the equations.

### 2.4.1 Navier-Stokes equations

*"There is a consensus among scientists that Navier-Stokes equations are a very good model for fluid flow."*[14]

Navier-Stokes equations describe the three-dimensional motion of viscous fluid. The equations consist of conservation of mass and conservation of momentum.

Conservation of mass:

$$\nabla \cdot \vec{u} = 0 \quad , \quad (2.6)$$



Conservation of momentum:

$$\rho \frac{D\vec{u}}{Dt} = -\nabla p + \mu \nabla^2 \vec{u} + \rho F , \quad (2.7)$$

$$\text{mass} * \text{acceleration} = \text{force} . \quad (2.8)$$

These equation can be used to model the weather, air flow, ocean currents and water flow. They help with the design of cars, airplanes, nuclear reactors, etc.

Simplified version is called Reynolds-averaged equation using dimensionless Reynolds numbers as an average and simplification. Reynolds number is defined as

$$Re = \frac{\text{inertia forces}}{\text{viscous forces}} . \quad (2.9)$$

### 2.4.2 Stable fluids

It is in high demand to convincingly mimic fluid-like behaviour like water, smoke or fire in many different branches of computer graphics. For example in painting programs where the emulation of traditional methods (i. e. watercolours or oil paint) is desired.

Physical models are great for this task. The interaction of flows with objects and forces is handled elegantly. It also permits an animator to create interesting, swirling-like fluid behaviors. However, the models prior to [14] use unstable explicit schemes to solve the physical equations which result in limited application. The results tend behave erratically and an animator has to reset the initial values which complicates the interactivity and speed.

Jos Stam [14] decided to follow up on the work of Foster and Metaxas [11]. They use explicit solver based on three-dimensional Navier-Stokes equations. The instability is mainly due to large time-step, but it can be caused also because of other parameters. Stam uses both Lagrangian<sup>7</sup> and implicit methods for solving the equations. The implicit method is in diffusion step. Foster and Metaxas used simpler, discretized solution.

Stam's simulation consists of four steps. Each simulation step needs to be completed during one time-step of the simulation. From an initial state:

- add force, assuming that the forces does not change much during the time-step
- perform advection (or convection) of the fluid on itself<sup>8</sup>
- apply the effect of viscosity, diffusion
- apply projection step

This solver can be implemented in both two- and three-dimensions. The motion is modelled by adding density into the fluid or by applying forces.

<sup>7</sup>It is combination of kinetic  $T$  and potential  $V$  energies:  $L \equiv T - V$  [21].

<sup>8</sup>The term  $-(u \cdot \nabla)u$  makes the equation non-linear. Thanks to Stam's method of characteristics, it results in unconditionally stable solver.



---

## Analysis and design

There are many factors that have to be considered when selecting a method creating for dynamic texture clouds. The main sub-problems are choice of method for generating clouds and visualization of chosen method.

Different goals come with their inherent requirements and that is reflected in used methods. A flight simulator will weigh more on the interaction of air currents and the render speed of clouds while a film animation will focus more on detail and visuals of a scene. For the first case very fast generating and rendering method has to be chosen. For the second case there are no limitations to the time allotted for generating and rendering the clouds therefore any method can be used. The latter application is closer to the focus of this thesis.

### 3.1 Generator

The following section will be focused on comparison of cellular automaton with coupled map lattice.

#### 3.1.1 Cellular automaton

One of the methods of generating clouds as dynamic textures is a heuristic approach using cellular automaton [12] (see 2.3.3). The main idea consists of several transition rules for three state variables in cells placed in three-dimensional grid. It results with very nice *cumulus* clouds (see Figure 3.1).

The advantage of this generator is its simple implementation. The grid is defined by three dimensions: number of cells in X-, Y- and Z-axis represented by variables  $X$ ,  $Y$  and  $Z$  respectively. The variables for each cell of the grid can be represented as Booleans and the transition rules can be computed using Boolean operations.

The memory complexity is  $O(2 \cdot N) \sim O(N)$ , where  $2 \cdot N$  represents storage of the current grid's variables and the computed new generation.

The time complexity of computing one new generation of the cloud varies on data structure used for storing the cloud's information. Each cell of the grid has

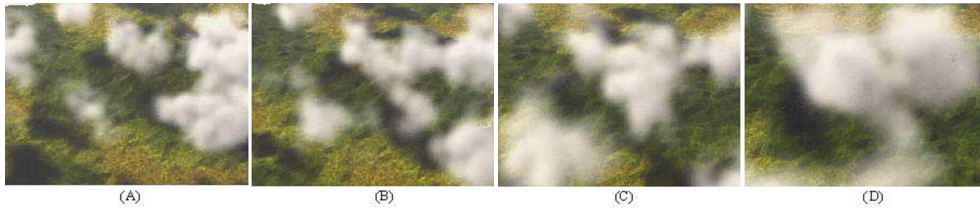


Figure 3.1: Animation of natural scenery [12]

to be recomputed and one of the cell's variables is dependent on up to eleven cells in its neighbourhood. It is appropriate to select data structure where getting and setting items is done fast. In the case of using list for storing the cell's data the time complexity<sup>9</sup> for these two operations is  $O(1)$ . In the case of using hashed map or dictionary the complexity for average case is also  $O(1)$ , but the amortized worst case can be  $O(N)$ , where  $N$  is number of cells in the grid.

The resulting time complexity for computing one new generation of the cloud is then

$$O(N) \tag{3.1}$$

in case of using list. If hashed map or dictionary is used as the storage the time complexity is in the worst case scenario  $O(N^2)$ .

Initialization of state variables in the grid is very important phase. It can be done randomly or with preset chunks of clouds. From the initial state the next generations are then computed using the transition rules without further need for user input. The state variables can be partly decided by random numbers and probabilities of occurrence of the state variables. The random elements secure irregularities in the cloud's growth.

### 3.1.2 Coupled map lattice

Coupled map lattice is an extension of CA. The idea of this method is much more complex than the approach using cellular automaton. It takes into consideration physics of the cloud creation and forces influencing the cloud. Different clouds are formed due to different conditions. This method classifies the clouds into two types based on the way they are formed. The first category is clouds that are formed by strong ascending air currents, *cumulus* and *cumulonimbus*. The second category is clouds formed by Bénard convection. The clouds have cell-like or roll-like shapes, e. g. *cirrocumulus* or *stratocumulus*.

CML method uses real-value variables. Pressure, density and the viscosity of the fluid have to be considered for simulating fluid flow. The method uses Navier-Stokes equations at its basis. Solving them can be very time-consuming. However, this method uses approximated models that have computationally smaller cost.

---

<sup>9</sup><https://wiki.python.org/moin/TimeComplexity>

The memory complexity of this method is comparable to the method using CA. The simulation space is divided into three-dimensional lattices and in each lattice a set of equations is computed.

The time complexity is more difficult to compute due to more complex equations (than Boolean operations of cellular automaton) are used during the cloud generation. There are also several phases of cloud formation: computing viscosity and pressure, advection by fluid flow, diffusion of water vapor, thermal diffusion, thermal buoyancy and phase transition.

### 3.1.3 Chosen method

The method using cellular automaton is more user friendly for design than the approach using CML although CML can generate more types of clouds. CML achieves this variety due to up to six phases of cloud generation that are computed with sets of physical equations. CA on the other hand uses only Boolean operations in one generation phase. The chosen approach was to use CA as a base for my generator implementation for its ease of use.

## 3.2 Visualization

Visualization of clouds is yet another large topic. The colour and transparency of cloud is dependent on position of main light sources and intensity of emitted light with its direction. Multiple studies ([3, 7, 10]) have been written about cloud illumination methods. Some methods are adjusted for more efficient rendering at run time, some give more emphasis on accuracy.

### 3.2.1 OpenGL

OpenGL is used in many solutions of generating clouds, e. g. [15, 20, 22]. It uses fast visualization with precomputed textures mapped on billboards. This method is fast because it utilizes graphical hardware and the rendering pipeline.

### 3.2.2 Blender

Blender is a free software suitable for modelling, sculpting, animating and rendering objects, visual effects and creating computer games. To this day at least fifteen<sup>10</sup> open films were created using Blender.

Blender provides favourable conditions for scripting and writing plugins. Almost every action performed in Blender by mouse and keyboard produces a record in a log which can be directly used in the script. It is easy to manipulate objects in a scene – creating objects, translating and rotating them, changing their materials or textures, or hiding the objects. Lighting with ray-tracing is also possible. If graphical card

---

<sup>10</sup><https://cloud.blender.org/films/>

### 3. ANALYSIS AND DESIGN

---

is present Blender can utilize it effectively. There exist solutions for natural-looking sky to be added as a background.

For the aforementioned reasons Blender has been chosen for both cloud visualization and simulation.

## 3.3 Design

The following section will list requirements for the cloud generator and in Python<sup>11</sup>.

### 3.3.1 Requirements

**Non-functional** requirements are:

- realized as a plugin for Blender version 2.92
- program written in Python

**Functional** requirements are for:

**Cloud generation** Cloud generator should be able to create at least one type of cloud. The generator should be operating according to set of parameters specifying size of initial cloud and number of iterations when the cloud grows.

**Visualization** User should be able to specify parameters regarding to appearance of clouds.

**Animation** The generator should also create an animation of dynamically growing cloud with possible motion in direction of wind.

### 3.3.2 Use cases

The functional requirements need more detailed explanation leading to use cases (see Figure 3.2<sup>12</sup>) and input parameters.

The cloud is represented in three-dimensional space. The initial dimensions can be represented as the grid height, depth and width, i. e. number of cells in X-, Y- and Z-axis respectively.

The number of times the cloud generator will run depends on number of frames and frequency of growth.

The visualization requirement is connected to specification of rendering quality. The information needed is: number of particles in the cloud, degree of cloud's particle displacement due to noise texture and density of cloud particles. After researching and testing different values of density and displacement the most suitable values were set as the plugin's default.

The cloud motion is realized based on directional vector and frequency of the wind. The frequency determines in which frames the cloud moves in the direction of the vector.

The plugin produces an animation of dynamically changing cloud which requires that total number of frames, frequency of wind and frequency of growth (in frames) are mandatory parameters.

---

<sup>11</sup>Blender plugins are written in Python.

<sup>12</sup>The diagram was drawn in <https://app.diagrams.net>.

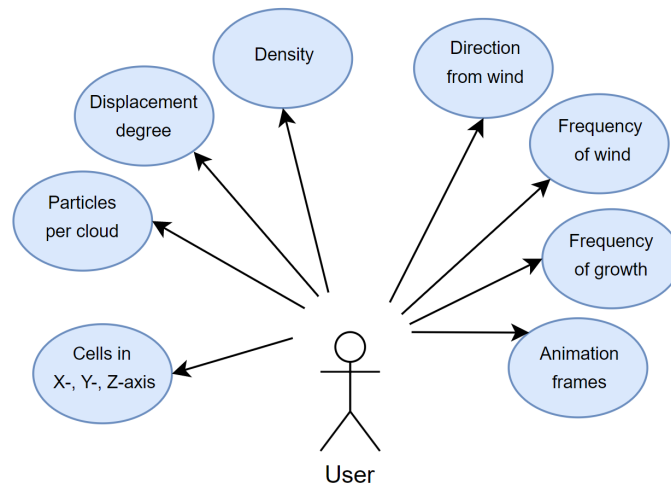


Figure 3.2: Use case diagram

The reason for using frequency in number of frames instead of speed is that it is easier to imagine when the cloud grows or moves.

The use cases<sup>13</sup> are:

- **Cells in (X, Y, Z)-axis:** specifying number of cells in X-, Y- and Z-axis
- **Particles per cloud:** specifying number of particles in a cloud
- **Displacement degree:** specifying degree (strength) of particle displacement in cloud due to noise texture
- **Density:** density of cloud
- **Direction from wind:** directional vector from wind
- **Frequency of wind (in frames):** frequency of cloud motion, e. g. every frame the cloud moves in direction from wind if this frequency is 1
- **Frequency of growth (in frames):** frequency of cloud growth, e. g. every frame the cloud grows according to CA's rules if this frequency is 1
- **Animation frames:** specifying number of animation frames

More parameters for the displacement texture are unnecessary because the used cloud texture can be changed at any time by user.

---

<sup>13</sup>Input parameters in the plugin are based on these use case names.



---

# Realisation

Before settling on using Blender plugins OpenGL was considered to render the generated clouds. However, this method has proven to be problematic. Using OpenGL would require Visual Studio on Windows as the programming environment which is a program with complex setup. It is much less user-friendly than Blender and with steeper learning curve. Furthermore this solution would not provide the desired outputs by itself and a separate visualization program would be needed.

## 4.1 Chosen method

The method that was ultimately chosen for realisation was generator using cellular automaton for cloud growth.

The solution of cloud generator that was decided to be implemented is a combination of [12, 15]. The transition rules are directly used from there, but the visualization method is different. The method using CA is easy to implement but there are problems with the visualization.

## 4.2 First results

Version 0.1 was written in C++ language. The preliminary code tests were carried out in text form in console and later visualized in

Wolfram Mathematica<sup>14</sup>.

There is even a rudimentary function for cellular automaton in Wolfram Mathematica, however it is inadequate for purposes of this thesis, because it is only for two-dimensional grids with predefined sets of rules.

The data to be visualized were supplied from a file generated by the program. The resulting image for a static cloud with random noise and a sphere is Figure 4.1. The cells with  $cld = 1$  are blue points.

---

<sup>14</sup>Piece of trivia: Stephen Wolfram wrote a paper on cellular automata, four years later he released this program. I realized it later on while writing the explanation on CA.

#### 4. REALISATION

---

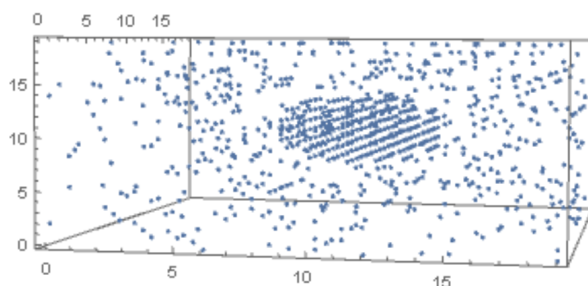


Figure 4.1: Static cloud from initial state

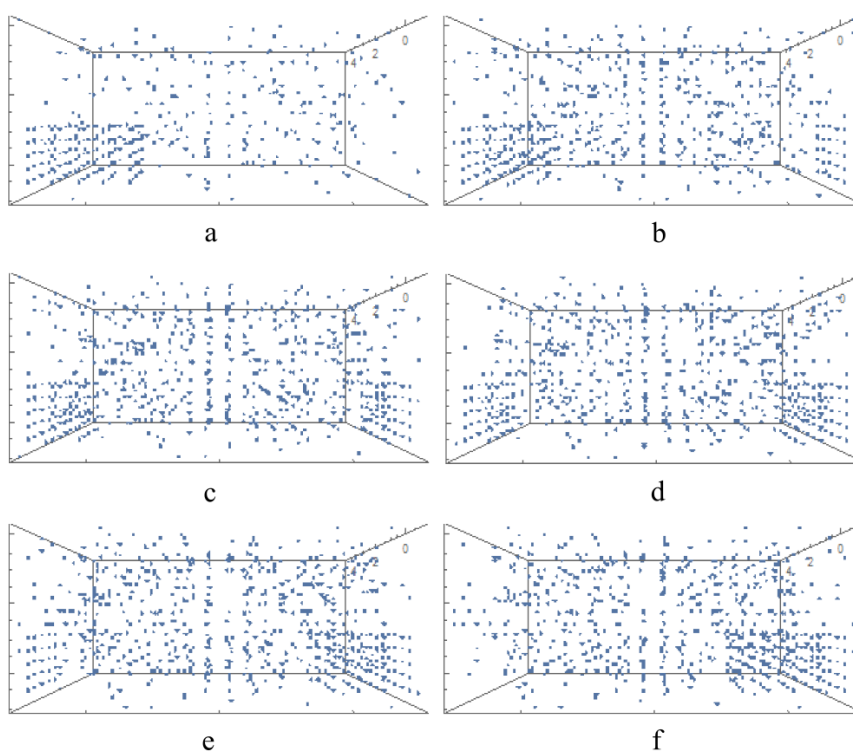


Figure 4.2: Animation of cloud CA in Wolfram Mathematica

An example of an animation of six cloud generations with motion in  $x$ -axis is in Figure 4.2. The initialization was done with a cube and noise. The cube is visible in the animation and the direction of motion can be derived from the cube's position. For this animation grid boundaries were set that when the cloud leaves the grid from one side of an axis it appears on the other side of the axis.

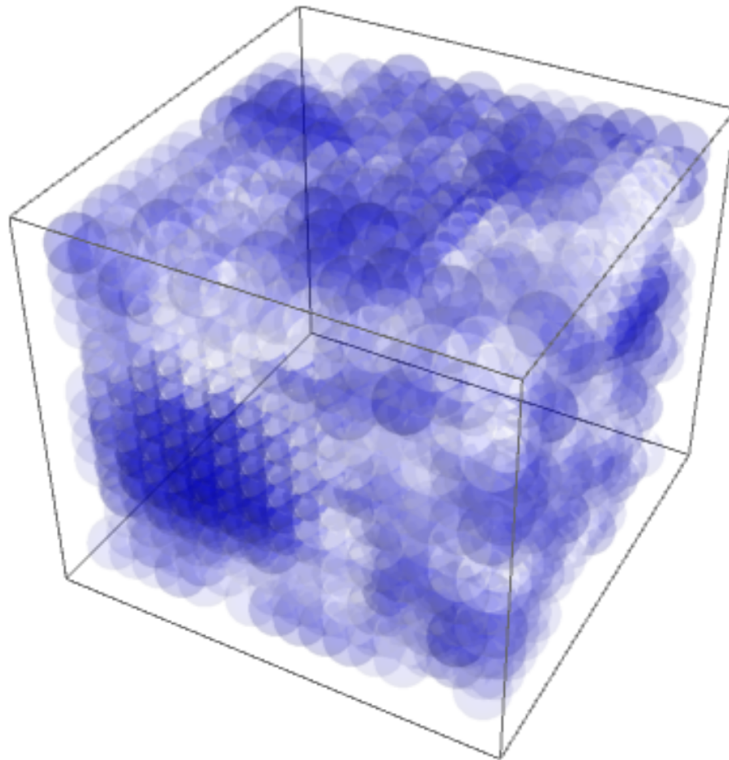


Figure 4.3: Visualization of cloudy neighbours

An alternative visualization of 11-neighbourhood (see 2.7). In Figure 4.3 is displayed the sum of neighbours with  $cld = 1$  for each cell of the grid. The sum cell is represented as an opacity of blue ball in three-dimensional grid. It is directly proportional to the count of cell's cloud neighbours and its opacity. The use of opacity has proven to be more reasonable than size.

### 4.3 First attempts in Blender

First attempts using Blender were in Blender 2.79. As the very first try, *obj* object of vertices generated by the previously mentioned program was imported into Blender. It was for a static cloud. This was a dead end.

The second attempt was made with python script. The script could generate transparent spheres into a scene. It remained as a loader of static cloud because the rendered images were not appealing.

### 4.4 Blender scripting

Blender 2.9x makes it possible to work with particle volumes. As the starting point of work in Blender could be considered consultation of the video<sup>15</sup> on topic of creating volumetric clouds in Blender.

At this point the decision was made to write a script for generating particle volumes in a three-dimensional grid and animating it by using cellular automaton rules.

#### 4.4.1 Script sketch

This is the pseudocode of the script used as a base for the plugin:

```
start(init, frames, wind, speed_wind, speed_growth):
    iters = compute_iters(frames, speed_growth)
    total = compute_dimns(init, iters)

    grid = init_grid(total)
    grid = init_cloud(grid, init)
    animate(grid, frames, wind, speed_wind, speed_growth)

    clean_up(grid)
```

The parameters *init* and *total* are three-dimensional vectors for user's defined initial cloud and total number of cells in the grid respectively.

Function *clean\_up* deletes ellipsoids that form the cloud from the scene. Only joined objects and particle volumes remain.

```
animate(grid, frames, wind, speed_growth, speed_wind):
    for f in frames:
        if f % speed_wind == 0:
            grid = translate_grid(grid, wind)
        if f % speed_growth == 0:
            grid = new_generation(grid)
```

---

<sup>15</sup>Video from channel by Blender Tutor:  
[https://www.youtube.com/channel/UCn6kWatTy0\\_ayxgJ3MqWaag](https://www.youtube.com/channel/UCn6kWatTy0_ayxgJ3MqWaag)

### 4.4.2 Initialization of cloud

A cloud is generated on the position of the cursor. The initial cloud is randomly set in a grid with user specified dimensions. The grid is then expanded to account for the space needed for the cloud to grow.

User specifies number of frames in which the cloud will be keyframed and speed of evolution for which the generator will run.

The number of CA iterations is computed as

```
iters = ceil(frames / speed_growth)
```

The dimensions of the extended grid are then

```
increment = Vector(iters * 2, iters * 2, iters + 4)
total_dimens = init_dimens + increment
```

The cloud with the initial dimensions is created from index

```
start_x = centre(total_dimens.x, init_dimens.x)
start_y = centre(total_dimens.y, init_dimens.y)

start_index = Vector(start_x, start_y, 2)
```

Function `centre` computes starting position from centre of the grid

```
centre = floor(total_dimens / 2) - floor(init_dimens / 2)
```

### 4.4.3 Growth of cloud

The cloud evolves and grows while keyframing into animation. The growth depends on the cellular automaton rules from [15]. It is to a certain degree randomized by probabilities for *hum* and *act* and randomly generated numbers. When computing new generation of the cloud there is also probability for cloud extinction that is chosen randomly from interval (0.14, 0.43) and multiplied by the shortest distance from grid borders. If *cld* in a cell changes to False it gets second chance for resurrection with probability of 0.11.

Each growth iteration the cells with *cld* = True are duplicated and joined into one mesh which is linked to a particle volume visible for this particular generation. All meshes stay hidden.

### 4.4.4 Movement of clouds

Blender is equipped with physics simulation engine with options for force field such as vortex, magnetic, fluid flow, wind and many more. One of the earlier ideas was to generate the clouds with the plugin and leave the user to add wind to the scene. The wind would create the motion of the clouds.

Rigid body modifier has to be applied for the cloud mesh to be responsive to forces. After multiple tweaks of different masses and wind strengths the combination of  $mass = 0.05$  kg and  $strength = 1$  was settled on. The rendered short animation was of growing cloud with the wind placed in the scene and the resulting movement was unexpected. Working with wind forces would be difficult and hard predict or debug.

For that reason it was decided to simulate movement in the plugin with directional vector from wind and frequency of the wind (used for keyframing frequency). When the time for the cloud translation comes the whole grid and the cloud mesh is moved in the direction of the wind.

The cloud evolution is computed while moving and making the animation frames. That means that new position of each cell is computed as

$$\text{position}_{new} = \text{position}_{old} + \overrightarrow{wind}. \quad (4.1)$$

#### 4.4.5 Representation of data

The cloud is being generated into a three-dimensional grid. The entire grid is stored as a dictionary with cell position as a key. Under the key there are stored mesh (ellipsoid) and Boolean variables  $cld$ ,  $hum$ ,  $act$  (with the same meaning as in [8, 12, 15]).

The growth of the cloud can produce more objects into scene, but the size of the dictionary (i. e. number of keys and items) stays the same as in the beginning when it was initiated.

Each growth of the cloud the cells with  $cld = \text{True}$  are duplicated and joined into a mesh and the new volume is linked to the mesh. The meshes and volumes can be stored in a list or a dictionary but it is not necessary.

First versions of the script generated spherical meshes with particle volumes linked to them. It was done for each cell of the grid in the beginning before creating the animation. The animation was done by hiding or showing the volumes (the meshes stayed hidden) while keyframing them. The resulting clouds had sharp shadows and the regular structure of the grid was obvious (Figure 6.2 on the left). The solution was slow because it generated a lot of unutilised objects into the scene.

Memory complexity of the final solution is in the worst case scenario double the amount of grid cells.

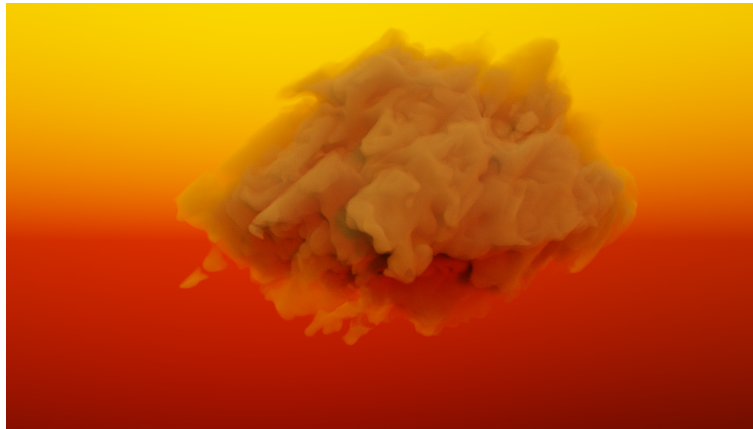


Figure 4.4: Colourful sunset

#### 4.4.6 Colours

The colour of the cloud is computed by illumination.

It is possible to have clouds in blue sky, sunset or sunrise. Several versions of sky were tried: light blue texture, photograph of sky, but the best results were with using Blender Sky texture nodes.

There are three types of Sky nodes, each of them based on different sky model. The sky used in the initial images was from Hošek and Wilkie [25]. Their sky's colour changes depending on Turbidity and Ground Albedo. The Sky texture based on paper by Preetham et al. [13] is quite similar. One minor disadvantage of these Sky textures is that there needs to be a light source added and set up.

The last Sky texture node<sup>16</sup> is based on Nishita's seminal paper [9]. Quite interesting results can be achieved with the parameter `Air` (which is for density of air molecules). By increasing this parameter the sky reddens and with it the particle volumes darken in the scene (Figure 4.4).

This sky texture also takes as parameters `Sun Elevation` and `Sun Rotation`. This sun removes the need for artificial sun in the scene (Figure 4.5). For visual check of this node's parameters Cycles render<sup>17</sup> has to be set up in Blender.

#### 4.4.7 Cloud texture

Cloud texture is a type of noise texture. The cloud texture is used for displacement of particles in generated clouds. In the plugin the parameter displacement degree is used as strength of the noise texture in the particle volumes.

User can change several of the texture parameters. The change will propagate also into existing clouds, because the same texture is identical for every run of the

<sup>16</sup>It is a method for displaying the earth from outer space.

<sup>17</sup>Cycles render is not default option after starting new Blender file. Running the plugin for the first time sets the Cycles as active render.



Figure 4.5: Blue sky

plugin. Most observed differences are made by changing noise scale of the texture – parameter named Size. Increasing the size creates blurred regions in the noise. It is made by enlarging small area of the initial noise texture (generated by random numbers) into bigger dimension and subsequently blurring the image. Table 4.1 shows rendered clouds with different sizes of noise textures and bits of the textures displayed on the spheres.

Table 4.1: Effect of increasing the size of cloud noise texture: bottom right quadrant of each volume is an example of used noise texture

Parameters					
Size	0.1	0.2	0.4	0.6	1
Particles			128		
Displacement			2.5		
Density			1		

#### 4.4.8 Technical problems

Considerable time was spent trying to resolve technical problems associated with Blender 2.9x. Several unsuccessful installations on both Linux and Windows were



experienced. One of which was on a virtual machine provided by school. The problem stemmed from obsolete components and a lack of support of newer versions of OpenGL which is required for rendering.

## 4.5 Plugin parameters

Effect of every parameter linked to cloud appearance can be difficult to understand theoretically. The following tables contain examples to demonstrate sample degrees of freedom of the presented model.

Table 4.2: Amount of particles


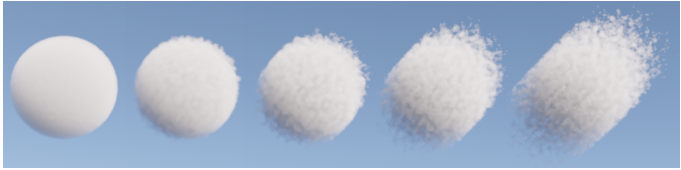
Parameters					
Particles	32	64	128	256	512
Displacement			0.5		
Density			1		


Table 4.3: Displacement degree

Parameters					
Particles			128		
Displacement	0	0.5	1.3	2.5	4
Density			1		

#### 4. REALISATION

---

Table 4.4: Density

Parameters					
					
Particles					128
Displacement					0.5
Density	0.1	0.3	0.6	1	4

---

# Testing

This section will present the conclusions of tests of the plugin's user interface. The comments of the testers were taken as an initiative for improving documentation.

## 5.1 Personas

Two abstract personas were selected to cover most of the user base.

**Persona A** is based on a 50-year-old man. He is an amateur meteorologist and he knows types of clouds and likes to photograph them. He works with computer on a daily basis. Every year he makes New Year postcards in Photoshop or in Gimp. He never worked with any graphical program for creating three-dimensional models or animations but he was curious after hearing about the plugin and decided to install Blender.

**Persona B** is modelled on a 24-year-old student. He learnt to work with Maya and AutoCAD for creating school projects. He uses Blender to model objects and creating short animations in his free time.

## 5.2 Testing scenarios

The testing subjects had to complete several tasks. First task was to install plugin into Blender and run the plugin with default settings to generate a cloud. Second task included generating a cloud based on more detailed information. Specified information were size of initial cloud, wind vector, frequency of cloud motion and number of cloud particles.

### **5.3 Conclusion of testing**

Testing subject A had minor problems with installing the plugin. He suggested README to the plugin where the installation process would be described. He would also welcome instructions for setting up sky and lighting the clouds in the scene. The subject A also noted that the names of some parameters are unclear and not intuitive and that a description of the parameters in the plugin menu would be helpful. He tried to generate several clouds with different values of parameters. He was quite content with the resulting clouds.

Testing subject B did not have any problems with the plugin installation. He tried the plugin on his notebook. He found it to be hardware demanding. There are known performance issues that could be optimized later by further development.

---

## Results

### 6.1 Version 1.x

From version 1.0 of the cloud generator the growing clouds were generated by script (as a pre-phase of the plugin) in Blender 2.92.0. Versions 1.x create whole three-dimensional grid of spherical objects. Centres of adjacent cells are 1 unit apart. There is a sphere with radius of 1 unit in each cell and it has its own particle volume. The cloud generation is very slow and the Blender file sizes have tendency to quickly grow out of control. Sky texture used as a background was based on Hošek/Wilkie and each cloud has to have its own light that has to be moved with the moving cloud for better illumination of the particle volumes (Figure 6.1). The last image of Figure 6.1 has visible artifacts due to render image computation. Exact origin of this artefact is unknown.

The grid composed of spherical cells was noticeably obvious when shadows were computed. When shadows were disabled it created cartoon-like clouds (Figure 6.2).

## 6. RESULTS

---



Figure 6.1: Version 1.1, moving light with cloud

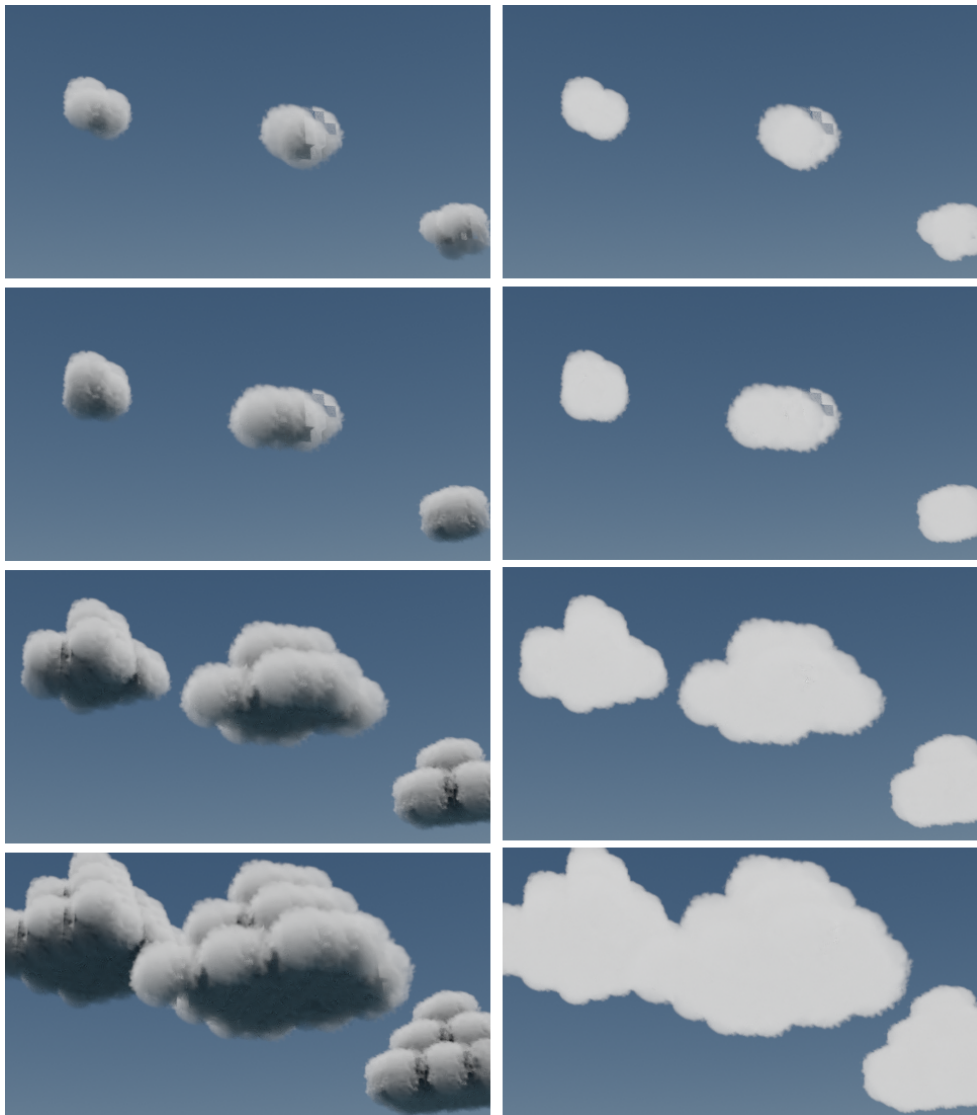


Figure 6.2: Version 1.2 with and without shadows

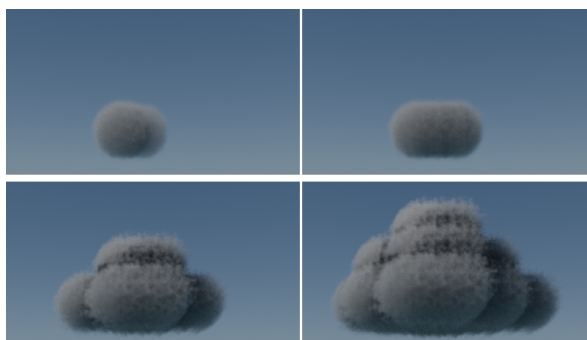


Figure 6.3: Version 2.1, radius of spheres = 2



Figure 6.4: Version 2.06

### 6.2 Version 2.x

In version 2.0 and later, spheres were added into the cloud during each step of the growth phase as they were required. Radius of cell spheres is randomized to be in interval  $(1, 2)$ . Later this value was set to 2 in hopes of eradicating most of the shadows and improving the cloud shape (Figure 6.3). This has proven insufficient and was discarded in later versions.

Nishita's sky was welcomed as a background because it provided more comfortable solution of cloud illumination (Figure 6.4).





Figure 6.5: Version 3.12

### 6.3 Version 3.x — Plugin

Version 3.0 brought more changes to the generator. In this version the spheres that form the basis for clouds were replaced by ellipsoids with randomized radius from interval  $(1,2)$ . Another big change was that only one particle volume was assigned to a cloud growth iteration. This change finally solved the problem of evident grid structure on the pictures (Figure 6.11).

More time was spent with setting up different parameters for better visuals of the clouds, e. g. displacement degree of particles in a cloud and size of the displacing cloud texture. And lastly the script was finalised and turned into a plugin.

## 6. RESULTS

---

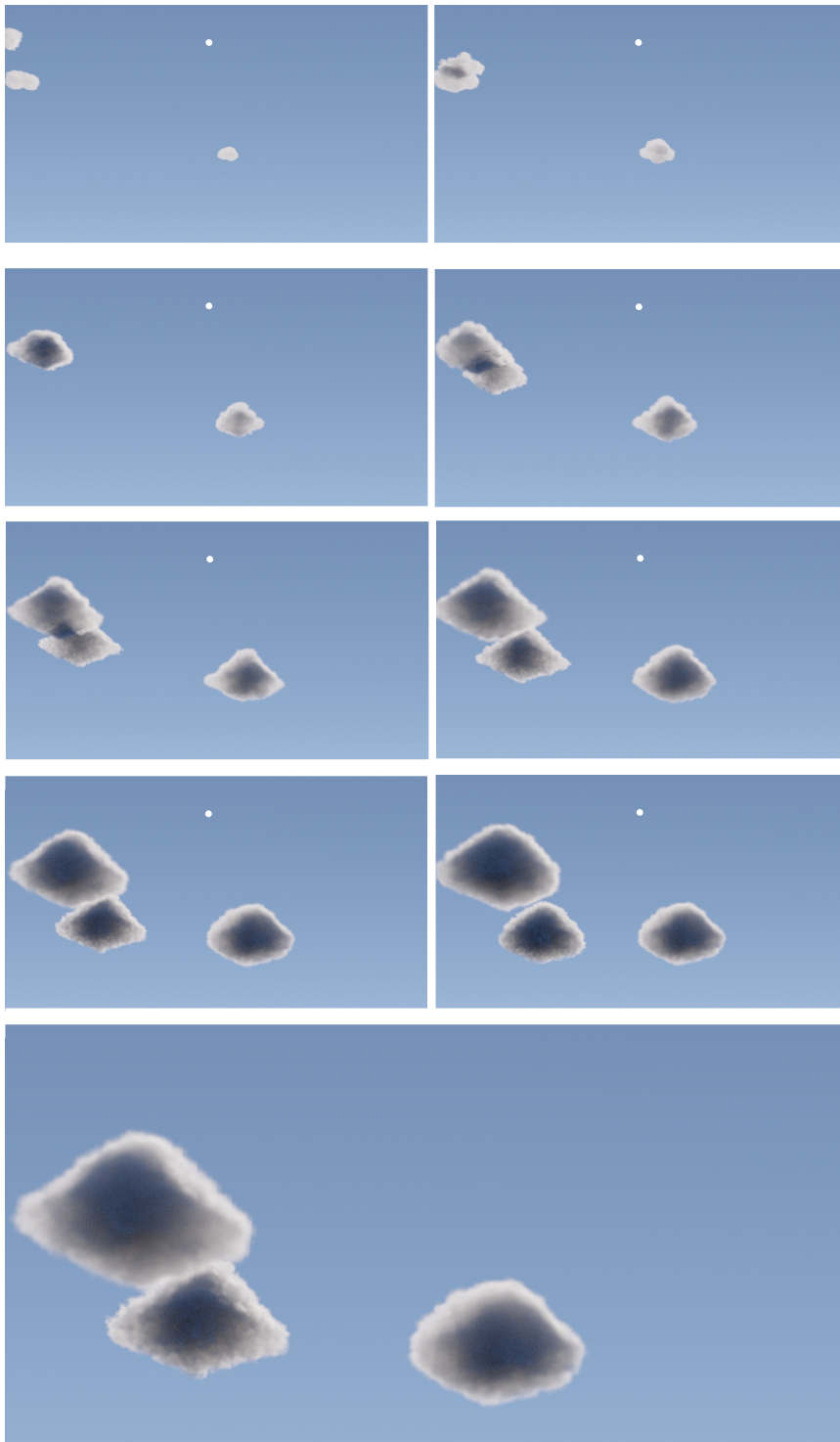


Figure 6.6: Version 3.14, sun from behind



Figure 6.7: Version 3.14, more clouds in scene

## 6. RESULTS

---



Figure 6.8: Version 3.14, growth + movement



Figure 6.9: Version 3.14, particles per cloud = 64

## 6.4 Render speed

The render of five-framed animation with Full HD resolution took 4m 17s when the cloud had 64 particles per volume (Figure 6.9). The render took 24m 20s for cloud with 512 particles per volume (Figure 6.10). Rendering was done on a machine with CPU Ryzen 5 5900X, RAM 96GB, GPU RTX 3080.

## 6. RESULTS

---



Figure 6.10: Version 3.14, particles per cloud = 512



Figure 6.11: Version 3.14

### 6.5 Comparison of clouds

The cloud generator has undergone great development. Clouds generated in early stages of the plugin development had evident regular structure because of layered volumes of particles. The models and rendered images only remotely resembled clouds with their colour and particle structure. The final version of plugin generate clouds that look like *cumuli* clouds on a nice sunny day.

The clouds cast shadows on each other and other objects in their line of illumination. They are modelled as a part of the scene therefore they obstruct the rays of light when shadows are computed for rendering in Cycles render in Blender. The clouds and their shadows in articles [12, 15] (where similar method for creating clouds was used) have to be computed in advance and stored in a buffer for later blending because OpenGL computes illumination for every object separately from each light and the rest of the scene is ignored.



---

## Conclusion

This thesis outlines different types of clouds and existing approaches to simulating them. I read articles on the topic of creation of clouds, their colours and types. Writing of my thesis required extensive observation of clouds in their natural habitat, the sky.

Several papers written on different approaches to generating clouds and their visualization in computer graphics were consulted. This thesis only manages to scratch the surface. The approaches studied were heuristic procedural solutions with stochastic variables, image-based processing methods and methods based on physical processes of fluid dynamics. The approaches with three-dimensional models were more in kind with my imagination.

I analysed methods that use cellular automaton and methods that use coupled map lattices. A comparison of these two methods was done based on their time complexity, memory complexity and diversity of resulting clouds.

Finally I decided to implement a cellular automaton for generating dynamic clouds as a Blender plugin.

The resulting clouds are from the *cumulus* family. By variation of the lighting white puffy clouds or stormy giants can be depicted.

This cloud generating plugin is intended for work in Blender which is ideal for making animations. Even though the generator can produce only one type of cloud with this plugin, many distinct clouds can be placed into a scene where they can move and evolve individually over time.

Although my solution requires high performance machine or long render times it can produce complex cloud systems with different vectors of motion and speed.

During the work on this thesis I got acquainted with new version of Blender which I found much more elegant than the older versions.

Every time I look up on the sky I think about what could have been done differently in my solution and I would like to someday revisit this topic and try to implement a different approach and develop it more.



---

## Bibliography

1. CSURI, C.; HACKATHORN, R.; PARENT, R.; CARLSON, W.; HOWARD, M. Towards an Interactive High Visual Complexity Animation System. In: *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques*. Chicago, Illinois, USA: Association for Computing Machinery, 1979, pp. 289–299. SIGGRAPH '79. ISBN 0897910044. Available from DOI: 10.1145/800249.807458.
2. REEVES, William T. Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics*. 1983, vol. 2, pp. 359–376.
3. KAJIYA, James T.; VON HERZEN, Brian P. Ray Tracing Volume Densities. *SIGGRAPH Comput. Graph.* 1984, vol. 18, no. 3, pp. 165–174. ISSN 0097-8930. Available from DOI: 10.1145/964965.808594.
4. WOLFRAM, Stephen. Computation theory of cellular automata. *Communications in mathematical physics*. 1984, vol. 96, no. 1, pp. 15–57.
5. *Laputa: Castle in the Sky* [film]. Director Hayao MIYAZAKI. Japan: Studio Ghibli, 1986.
6. KANEKO, Kunihiko. Simulating physics with coupled map lattices. In: *Formation, Dynamics And Statistics Of Patterns: (Volume 1)*. World Scientific, 1990, pp. 1–54.
7. WYVILL, Geoff; TROTMAN, Andrew. Ray-tracing soft objects. In: *CG International'90*. Springer, 1990, pp. 469–476.
8. NAGEL, Kai; RASCHKE, Ehrhard. Self-organizing criticality in cloud formation? *Physica A: Statistical Mechanics and its Applications*. 1992, vol. 182, no. 4, pp. 519–531.
9. NISHITA, Tomoyuki; SIRAI, Takao; TADAMURA, Katsumi; NAKAMAE, Eihachiro. Display of the earth taking into account atmospheric scattering. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 1993, pp. 175–182.

10. MAX, Nelson. Efficient light propagation for multiple anisotropic volume scattering. In: *Photorealistic Rendering Techniques*. Springer, 1995, pp. 87–104.
11. FOSTER, Nick; METAXAS, Dimitris. Modeling the motion of a hot, turbulent gas. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 181–188.
12. DOBASHI, Yoshinori; NISHITA, Tomoyuki; OKITA, Tsuyoshi. Animation of clouds using cellular automaton. In: *Proceedings of Computer Graphics and Imaging*. 1998, vol. 98, pp. 251–256.
13. PREETHAM, A. J.; SHIRLEY, Peter; SMITS, Brian. A Practical Analytic Model for Daylight. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 91–100. SIGGRAPH '99. ISBN 0201485605. Available from DOI: 10.1145/311535.311545.
14. STAM, Jos. Stable Fluids. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128. SIGGRAPH '99. ISBN 0201485605. Available from DOI: 10.1145/311535.311548.
15. DOBASHI, Yoshinori; KANEDA, Kazufumi; YAMASHITA, Hideo; OKITA, Tsuyoshi; NISHITA, Tomoyuki. A Simple, Efficient Method for Realistic Animation of Clouds. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 19–28. SIGGRAPH '00. ISBN 1581132085. Available from DOI: 10.1145/344779.344795.
16. MIYAZAKI, R.; YOSHIDA, S.; DOBASHI, Y.; NISHITA, T. A method for modeling clouds based on atmospheric fluid dynamics. In: *Proceedings Ninth Pacific Conference on Computer Graphics and Applications. Pacific Graphics 2001*. 2001, pp. 363–372. Available from DOI: 10.1109/PCCGA.2001.962893.
17. WEISSTEIN, Eric W. Cellular automaton. <https://mathworld.wolfram.com/>. 2002.
18. DORETTO, G.; CHIUSO, A.; WU, Y. N.; SOATTO, S. Dynamic textures. *ijcv*. 2003, vol. 51, no. 2, pp. 91–109.
19. DORETTO, G.; SOATTO, S. Editable dynamic textures. In: *cvpr*. Madison, Wisconsin, USA, 2003, vol. 2, pp. 137–142.
20. LIAO, Horng-Shyang; CHUANG, Jung-Hong; LIN, Cheng-Chung. Efficient Rendering of Dynamic Clouds. In: *Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry*. Singapore: Association for Computing Machinery, 2004, pp. 19–25. VRCAI '04. ISBN 1581138849. Available from DOI: 10.1145/1044588.1044591.

21. MORIN, David. *Introduction to classical mechanics: with problems and solutions*. Cambridge University Press, 2008.
22. LIU, Zhenbao; WANG, Zhongsheng; ZHANG, Chao. *Scheme of Dynamic Clouds Generation for 3D Real Time Flight Simulation*. 2010. Available from DOI: 10.1109/ICCMS.2010.24.
23. HAINDL, Michal; FILIP, Jiri. *Visual texture: Accurate material appearance measurement, representation and modeling*. Springer Science & Business Media, 2013.
24. HAINDL, Michal; RICHTR, Radek. Dynamic texture enlargement. In: *Proceedings of the 29th Spring Conference on Computer Graphics*. 2013, pp. 5–12.
25. WILKIE, Alexander; HOŠEK, Lukas. Predicting sky dome appearance on earth-like extrasolar worlds. In: *Proceedings of the 29th Spring Conference on Computer Graphics*. 2013, pp. 145–152.
26. RICHTR, Radek; HAINDL, Michal. Dynamic texture editing. In: *SCCG*. 2015, pp. 133–140.
27. EMERY, William; CAMPS, Adriano. Chapter 8 - Atmosphere Applications. In: EMERY, William; CAMPS, Adriano (eds.). *Introduction to Satellite Remote Sensing*. Elsevier, 2017, pp. 597–636. ISBN 978-0-12-809254-5. Available from DOI: <https://doi.org/10.1016/B978-0-12-809254-5.00008-7>.
28. GUERRILLA. *Horizon Zero Dawn* [comp. software]. 2017. Available also from: <https://www.playstation.com/en-us/games/horizon-zero-dawn/>.
29. HINTERLAND STUDIO INC. *The Long Dark* [comp. software]. 2017. Available also from: <https://www.thelongdark.com/>.
30. RICHTR, Radek. *Modelování dynamických textur*. 2018. PhD thesis. Výpočetní a informační centrum, České vysoké učení v Praze.
31. RICHTR, Radek; HAINDL, Michal. Dynamic Texture Similarity Criterion. In: *2018 24th International Conference on Pattern Recognition (ICPR)*. 2018, pp. 904–909.
32. CHU, Rui Jian; RICHARD, Noël; CHATOUX, Hermine; FERNANDEZ-MALOIGNE, Christine; HARDEBERG, Jon Yngve. Hyperspectral Texture Metrology Based on Joint Probability of Spectral and Spatial Distribution. *IEEE Transactions on Image Processing*. 2021, vol. 30, pp. 4341–4356. Available from DOI: 10.1109/TIP.2021.3071557.



## Acronyms

**CA** Cellular automation

**CML** Coupled map lattice

**IBR** Image-based rendering technique





---

## Contents of enclosed CD

	readme.txt.....	the file with CD contents description
	src .....	the directory of source codes
	plugin.....	the directory of implementation
	thesis.....	the directory of L <sup>A</sup> T <sub>E</sub> X source codes of the thesis
	text.....	the thesis text directory
	thesis.pdf.....	the thesis text in PDF format
	visualization.....	the directory of described visualization methods
	Blender.....	Blender file with example scene
	WolframMathematica .....	the directory of examples in WM
	renders.....	the directory of high resolution renders