# Precipitation generation appropriate to simulated fluid dynamic clouds generated in real-time.

**Jake Morey**

**BSc(Hons) Computer Games Technology**
**University of Abertay Dundee**
**School of Arts, Media and Computer Games**
**May 2014**

**University of Abertay Dundee**

Permission to copy

Author: Jake Morey
Title: Precipitation generation appropriate to simulated fluid dynamic clouds generated in real-time.

Degree: BSc(Hons) Computer Games Technology

Year: 2014

(i) I certify that the above mentioned project is my original work

(ii) I agree that this dissertation may be reproduced, stored or transmitted, in any form and by any means without the written consent of the undersigned.

Signature

……..

Date ……..

## Abstract

# Contents

**Abstract** i

**List of Figures** ii

**List of Tables** iii

**1 Introduction** 1

1.1 Weather: . . . 1

1.2 Research Question: . . . 2

**2 Literature** 4

2.1 Background: . . . 4

2.2 Cloud Generation: . . . 4

2.2.1 Artist Created: . . . 5

2.2.2 Cellular Automata (CA): . . . 5

2.2.3 Fluid Dynamics: . . . 6

2.3 Cloud Rendering: . . . 9

2.3.1 Single Scattering: . . . 9

2.3.2 Multiple scattering: . . . 10

2.3.3 Photon Mapping: . . . 10

2.4 Rain Rendering: . . . 10

**3 Methodology:** 12

3.1 Practical Methodology: . . . 12

3.2 Evaluation Methodology: . . . 18

**4 Results:** 20

4.1 Visual Comparison: . . . 20

4.2 Resource Usage: . . . 20

**5 Discussion:** 23

**6 Conclusions:** **26**

6.1 Further Work: . . . . . 26

**Appendix** **27**

**References** **32**

# List of Figures


Figure 1 Left: Ouranos! (1980), Right: Dear Ester (2012) . . . . . 1

Figure 2 Left: rain (2013), Right: Heavy Rain (2010) . . . . . 2

Figure 3 A screenshot from Tomb Raider (2013) . . . . . 4

Figure 4 Main CUDA Frame Pseudo Code . . . . . 14

Figure 5 Render Pseudo Code . . . . . 15

Figure 6 Volume Rendering Pseudo Code . . . . . 15

Figure 7 Advection Pseudo Code . . . . . 16

Figure 8 Vorticity Confinement Pseudo Code . . . . . 16

Figure 9 Vorticity Confinement Pseudo Code . . . . . 17

Figure 10 Buoyancy Force Pseudo Code . . . . . 17

Figure 11 Water Continuity Pseudo Code . . . . . 17

Figure 12 Temperature Pseudo Code . . . . . 18

Figure 13 Divergence and Jacobi Pseudo Code . . . . . 18

Figure 14 Final Pseudo Code . . . . . 18

Figure 15 Dobashi *et al.* (2000) . . . . . 19

Figure 16 $t = 773s$ . . . . . 20

Figure 17 Harris *et al.* (2003) . . . . . 27

Figure 18 Miyazaki *et al.* (2001) . . . . . 28

Figure 19 Fedkiw, Stam and Jensen (2001) . . . . . 29

Figure 20 Harris *et al.* (2003) . . . . . 30

Figure 21 Visual Evaluation . . . . . 31

# List of Tables

Table 1 Texture Grid Size Performance Analysis . . . . . 21

Table 2 CUDA Kernel Resource Usage . . . . . 21

# 1 Introduction

## 1.1 Weather:

Since the early 1980's a major part of games have been there use of weather. The weather used in games generally has one of two different uses. The first use of weather in games is using the weather to affect gameplay such as Ouranos! (1980) which uses it as the main game mechanism . More recently Dear Ester (2012) used the weather for the second reason which is as a way to add to the immersion the player feels when playing the game. Figure 1 contains screenshots from both games and showcases how far the weather has come in games. There are some games that utilise both types of weather in games, including Microsoft Flight Simulator X (2003).


Figure 1: Left: Ouranos! (1980), Right: Dear Ester (2012)

Looking closer at the types of weather used in games the use of clouds and rain stand out as the most recurring and notable aspects of weather as well as the most versatile examples of its use in games. For example rain plays different roles in two games, Heavy Rain (2010) and rain (2013). However, without it, the games would not be as compelling as the currently are. Figure 2 shows screenshots from both games. rain (2013) shows how rain is used to affect gameplay by allowing the player only to see the character in the rain, while Heavy Rain (2010) shows how the use of rain adds a depth that increases the film noir feel of the game.

When creating these games a number of different techniques were used to create and render the weather. Ouranos! (1980) uses ASCII because of the limitations

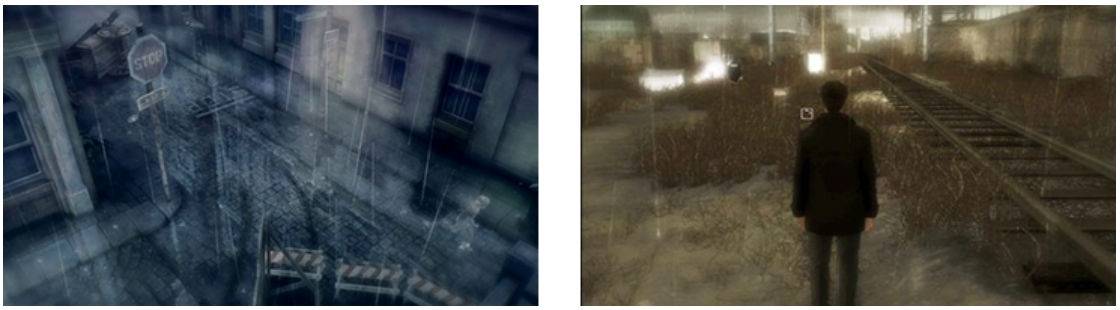


Figure 2: Left: rain (2013), Right: Heavy Rain (2010)

at the time. Games such as Super Mario Bros (1985) use 2D sprites to render clouds on to the background and games like Tomb Raider (2013) uses 3D scripted clouds. 2D games are also more likely to use 2D scrolling rain texture while 3D games are more likely to use a particle system to generate the rain. Some games can use location data to simulate the correct weather at a given location such as NCAA Football 14 (2013). These games usually have a backup dynamic weather system which will be used if the game can't connect to the internet to collect data. Most games created use artistic representation of clouds instead of creating them in real time using equations. Fluid dynamic equations can be used to represent the movement of any object made up of liquid or gas. Basic clouds can be modelled by fluid dynamic equations, more advance clouds need extra equations for water continuity, thermodynamics, and buoyancy. Using equations to generate the clouds will mean that the cloud will behave more like a real cloud than an artist's interpretation of a cloud. When creating the cloud using these equations rain generation can be based proportionally on cloud size or by adding an extra equation to water continuity equations so rainfall is included.

## 1.2 Research Question:

The project aims to create realistically moving and looking clouds in real time using fluid dynamic equations. Another aim is to create rain in locations and an appropriate amount that relates to the clouds created. The last aim is to compute the equations in the GPU so that the equations can be computed efficiently. These

aims lead to the research question:

***How can the amount of Precipitation generated in a game be related to realistic simulated clouds generated in real-time using fluid dynamic equations?***

Answering the research question results in a number of objectives the need to be completed including using a number of equations, mainly fluid dynamic equations in 3D, space to generate and move clouds. To use these clouds to generate the rain in a 3D scene, the fluid dynamic equations will need to be optimized to run as smoothly as possible in the application whilst producing realistic, or at least plausible, effects.. The final application will be analysed visually and numerically to test how the clouds grow and move over time, as well as generating rain in the most efficient way.

# 2 Literature

## 2.1 Background:

This section will look at how clouds and rain are generated currently in games. Weather plays a huge impact to the how a game feels, as Barton (2008) wrote about how "It was a dark and stormy night" not only sets the time and weather of the scene but also sets the tone. Wang (2004) agrees by saying that one of the most fascinating parts of a scene could be the clouds. An example of this is the game Tomb Raider (2013) which at numerous points in the game the user can see a vast sky full of clouds as seen in Figure 3.

Figure 3: A screenshot from Tomb Raider (2013)

## 2.2 Cloud Generation:

There are a number of different methods for generating clouds from cellular automata, to fluid dynamic equations, to importing 3D objects.

### 2.2.1 Artist Created:

This technique works not by creating the clouds at run time but instead creates the clouds as models and loads them into the game when needed. Wang (2004) describes a version of this which allows artists to create boxes in 3DS Max in which a plug-in will then generate clouds inside the box. She explained how this method was used when creating Microsoft Flight Simulator 2004: A Century of Flight (2003). A similar method can be used in the CryEngine 3 SDK (2013) which allows the user to alter the properties of the clouds created in real-time within the editor. This method can create very realistic looking clouds but doesn't allow for realistic movement or generation.

### 2.2.2 Cellular Automata (CA):

A Cellular Automaton can be described as a regular shaped structure which consists of identical cells that are computed synchronously depending on the state of the cell and its neighbours (Dantchev, 2011). Dobashi *et al.* (2000) used a cellular automaton model when generating the clouds which involved giving each cell a number of boolean states that, when coupled with the rules generated clouds.

This method was extended by Miyazaki *et al.* (2001) who used the Coupled Map Lattice (CML) method which is described as "an extension of cellular automaton, and the simulation space is subdivided into lattices". Miyazaki *et al.* (2001) also goes on to explain that the CML model differs from the CA model by using continuous values instead of discreet values. This CML model uses very simple equations for viscosity and pressure effects, advection, diffusion of water vapour, thermal diffusion and buoyancy, and the transition from vapour to water.

Cellular Automaton gives a lot more control to the physics of clouds because of the equations used to define them compared to clouds created by artists. However these equations are not as accurate as using fluid dynamic equations to move and generate clouds.

### 2.2.3 Fluid Dynamics:

As clouds can be described as an incompressible fluid it can be simulated via the fluid dynamic equations. The Navier-Stokes Equations are used for a "fluid that conserves both mass and momentum." (Stam, 1999). In the Navier-Stokes equations $\rho$ is the density, $\mathbf{f}$ represents all external forces and $\nu$ is the kinematic viscosity of the fluid. The velocity and pressure are defined as $\mathbf{u}$ and $p$ respectively. The second equation is the continuity equation which means the fluid is incompressible.

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\,\mathbf{u} - \frac{\nabla p}{\rho} + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

The Navier-Stokes equations (1) and (2) can be simplified to Euler's Equation when "the effects of viscosity are negligible in gases" (Fedkiw, Stam and Jensen, 2001). This makes the equations for generating the clouds less computationally heavy and can be shown in equation (3) which has no $\nu \nabla^2 \mathbf{u}$. The continuity equation has not changed and can be seen from (4) being the same as (2).

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\,\mathbf{u} - \frac{\nabla p}{\rho} + \mathbf{f} \tag{3}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{4}$$

Fedkiw, Stam and Jensen (2001), Harris *et al.* (2003), and Overby, Melek and Keyser (2002) all used work created by Stam (1999) on stable fluid simulations. Overby, Melek and Keyser (2002) used the actual solver created by Stam (1999) in the application to solve the fluid dynamic part of creating clouds. Whereas Fedkiw, Stam and Jensen (2001) and Harris *et al.* (2003) used the theory in the creation of the smoke and clouds respectively. Even though all three used the same start for simulating cloud generation they have different methods for assigning values to the other equations needed.

The Fedkiw, Stam and Jensen (2001) model uses a Poisson equation to compute

the pressure of the system and two scalar functions for advecting the temperature and density. This model also uses a function built up of the temperature, ambient temperature, density, and two other positive constants to create a buoyancy effect. The model also simulates velocity fields, which are dampened out on the coarse grid, by finding where the feature should be and then creating a realistic turbulent effect.

Overby, Melek and Keyser (2002) computes the local temperature based upon the heat energy and the pressure. The pressure is calculated from ground level to the tropopause by an exponentially decreasing value (Overby, Melek and Keyser, 2002). The buoyancy in this model is created using the local temperature, the surrounding temperature and a buoyancy scalar. Relative humidity is calculated based upon current water vapour and saturated water vapour. Water condensation is then calculated based upon relative humidity, hygroscopic nuclei, and a condensation constants. The final equation to be calculated is the latent heat which is calculated by the water condensation and a constant.

The Harris *et al.* (2003) model uses equations for water continuity, thermodynamics, buoyancy, and a Poisson equation for fluid flow. This model also creates velocity fields using the same process as Fedkiw, Stam and Jensen (2001). This model uses more complicated equations than the previous two models to more accurately simulate the creation of clouds. For example this model uses gravity, the mass mixing ratio of hydrometeors and virtual potential temperatures, whereas the previous models use scalars or other constants with the temperature to create the buoyancy force.

The vorticity confinement as defined by Harris *et al.* (2003) model is defined in equation (5). $\boldsymbol{\omega}$ is the vorticity, defined by $\boldsymbol{\omega} = \nabla \times \mathbf{u}$, and $\mathbf{N}$ is the normalized vorticty vector field and points from areas of lower vorticiy to areas of higher vorticity which is defined by equation (6). With $h$ is the grid scale and $\epsilon$ is a scale parameter.

$$\mathbf{f}_{vc} = \epsilon h \left( \mathbf{N} \times \boldsymbol{\omega} \right) \tag{5}$$

$$\mathbf{N} = \frac{\nabla \left| \boldsymbol{\omega} \right|}{\left| \nabla \left| \boldsymbol{\omega} \right| \right|} \tag{6}$$

The buoyant force is defined in equation (7) where $g$ is the acceleration due to gravity. $q_v$ is the mixing ratio of hydrometeors and in this case is defined as $q_c$, the mixing ratio of liquid water. $\theta_v$ is the virtual potential temperature and is defined in equation $\theta_v \approx \theta(1 + 0.61 q_v)$. $\theta_{v0}$ is the reference potential temperature and is between 290 and 300K as defined by Harris *et al.* (2003).

$$B = g \left( \frac{\theta_v}{\theta_{v0}} - q_h \right) \tag{7}$$

The water continuity in Harris *et al.* (2003) model is based upon the Bulk Water Continuity model which described by Houze (1994) as "the simplest type of cloud is a warm non-precipitating cloud". Houze (1994) describes the model as a set of categories in which clouds can be created. The categories for this simple type of cloud model are vapour $q_v$ and cloud liquid water $q_c$ and are described in equations (8) by Houze (1994). $C$ is the condensation rate.

$$\frac{Dq_v}{Dt} = -C, \frac{Dq_c}{Dt} = C \tag{8}$$

The thermodynamic equation defined in Harris *et al.* (2003) model can be seen in equation (9). $c_p$ is the specific heat capacity of dry air at constant pressure in this case $1005 Jkg^{-1}K^{-1}$. $L$ is the latent heat of vaporization of water which is $2.501 Jkg^{-1}$. The part of the equation in brackets on the right hand side of the equation can be exchanged for the condensation rate from the water continuity equations.

$$\frac{\partial \theta}{\partial t} + \left( \mathbf{u} \cdot \nabla \right) \theta = \frac{-L}{c_p \Pi} \left( \frac{\partial q_v}{\partial t} + \left( \mathbf{u} \cdot \nabla q_v \right) \right) \tag{9}$$

$\Pi$ is called the Exner function and is defined in equation 10. Where $p_0$ is the

pressure at the surface, usually taken as $1000hPa$; $R_d$ is the gas constant for dry air a can be taken as $287Jkg^{-1}K^{-1}$; $c_p$ is the heat capacity of dry air at constant pressure, and $p$ is pressure.

$$\Pi = \left(\frac{p}{p_0}\right)^{\frac{R_d}{c_p}} \tag{10}$$

With these extra equations using fluid dynamics for generating and moving clouds will give more accurate simulations compared to the previously mentioned processes. There is a draw back for using fluid dynamic equations as it will use more computing power compared to artistically created, and Cellular Automaton methods.

## 2.3 Cloud Rendering:

Due to the nature of clouds when light passes through them it becomes scattered. The majority of the models looked at the use two different techniques to accomplish this effect: single scattering and multiple scattering. These models may render clouds using these scattering techniques directly or may use scattering inside other rendering processes such as photon mapping. A number of these models also use billboards or imposters when rendering the clouds as this saves on computation. This section will look at single scattering, multiple scattering, and photon mapping.

### 2.3.1 Single Scattering:

Harris and Lastra (2001) describe single scattering as a model that simulates scattering in a single direction that is usually the direction leading to the point of view. There are debates concerning whether or not this type of rendering is detailed enough for rendering clouds. Miyazaki *et al.* (2001) states the main topic of his model is the cloud shapes so using single scattering is enough to check the shape of the cloud. Bohren (1987) describes single scattering as insufficient when describing common observations. Due to this being a simpler and less computa-

tional heavy method of rendering clouds compared to other methods this process could be best for cloud generated using a number of complex equations.

### 2.3.2 Multiple scattering:

"Multiple scattering models are more physically accurate, but must account for scattering in all directions ... and therefore are much more complicated and expensive to evaluate" (Harris and Lastra, 2001). Harris *et al.* (2003) uses a version of multiple scattering which is called multiple forward scattering, this differs from the original by instead of calculating scattering in all directions it calculates scattering in the forward direction only. This means the algorithm is less computationally heavy. Fedkiw, Stam and Jensen (2001) describe the multiple scattering of light as necessary for objects made from water vapour, which clouds are.

### 2.3.3 Photon Mapping:

"Photon mapping is a variation of pure Monte Carlo ray tracing in which photons (particles of radiant energy) are traced through a scene" (Jensen (1996), cited in Harris (2003)). Harris (2003) describes the process of photon mapping as storing position, incoming direction, and radiance of each photon landing on a nonspecular surface that has been traced from the light source. Fedkiw, Stam and Jensen (2001) use photon mapping when rendering smoke and describe the process as a two pass algorithm, one where a volume photon map is built and the second as a rendering pass using a forward ray marching algorithm.

## 2.4 Rain Rendering:

"Rain is an extremely complex natural atmospheric phenomenon" (Puig-Centelles, Ripolles and Chover, 2009). Puig-Centelles, Ripolles and Chover (2009) describe two main techniques for rendering rain to a scene scrolling textures where a texture the size of the screen scrolls in the direction of the rain, and a particle system where each rain drop is represented as a particle in the system. Tariq (2007) writes

"animating rain using a particle system is more useful for realistic looking rain with lots of behaviour (like changing wind)." Puig-Centelles, Ripolles and Chover (2009) sates that texture scrolling "is faster than particle systems, but it does not allow interaction between rain and the environment."

An extension to the Bulk Water Continuity model which was described in section 2.2.3 allows a warm precipitating cloud with rain as an additional category, Houze (1994). Now instead of two equations there are three which are shown in equations 11 - 13.

$$\frac{dq_v}{dt} = -C + E_c + E_r \tag{11}$$

$$\frac{dq_c}{dt} = C - A - K - E_c \tag{12}$$

$$\frac{dq_r}{dt} = A + K + F - E_r \tag{13}$$

In the above three equations $C$ represents the condensation of vapour into cloud water. $A$ is the autoconversion which is the rate cloud water decreases as particles grow to raining size. $E_c$ and $E_r$ are evaporation variables, the former of cloud water and the latter is evaporation of rainwater. $K$ is the collection of cloud water and $F$ represents the rain fallout of the model. Adding the third equation and the extra variables to the water continuity from section 2.2.3 will allow for more realistic rain generation with a particle system.

# 3 Methodology:

## 3.1 Practical Methodology:

In answering the research question an application has been created showing the generation of clouds in real time as well as generating different amounts of rain falling from these clouds when the correct amount of precipitation is generated. The application has been written using C++ using the Visual Studio 2012 (2013). This application has been built upon a free tutorial framework called RasterTek (2014), As well as using C++ and the Raster Tek framework NVIDIA (2013a) CUDA general-purpose GPU computing language will be used to simulate the growth and dissipation of the clouds in real time. The DirectX 11 (2012) API will be used to render, this data created from the CUDA processes, to the screen.

The application consists of three parts first is an uneven terrain which is loaded in from a height map texture. The second part of the application is the cloud layer which is generated whilst the application is running. The last part is which is also generated when the application is running is the rain particle system. Much like the model used by Dobashi *et al.* (2000) which is shown below in Figure 15. Snapshots from Harris *et al.* (2003) and Miyazaki *et al.* (2001) are both shown in the appendix as Figure 17 and Figure 18 respectively. The clouds have been generated using fluid dynamic equations as well as thermodynamic and water continuity equations. The fluid equations like mentioned in section 2.2.3 are described in equations (14) and (15).

$$\frac{\partial \mathbf{u}}{\partial t} = -\left(\mathbf{u} \cdot \nabla\right)\mathbf{u} - \frac{\nabla p}{\rho} + \mathbf{f} \tag{14}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{15}$$

$$\mathbf{f} = \mathbf{f}_{vc} + \mathbf{B} \tag{16}$$

Where $\rho$ is the density, $\mathbf{f}$ represents all external forces. The velocity and

pressure are determined as $\mathbf{u}$ and $p$ respectively. The second equation is the continuity equation which means the fluid is incompressible. The forces are made up of two separate equations as shown in equations (16). Equation (17) is the vorticity confinement which is defined by Harris *et al.* (2003). The variables are defined as $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ and $\mathbf{N}$ as the normalization of this variable.

$$\mathbf{f}_{vc} = \epsilon h \left( \mathbf{N} \times \boldsymbol{\omega} \right), \mathbf{N} = \frac{\nabla \left| \boldsymbol{\omega} \right|}{\left| \nabla \left| \boldsymbol{\omega} \right| \right|} \tag{17}$$

$$B = g \left( \frac{\theta_v}{\theta_{v0}} - q_h \right) \tag{18}$$

The other equation (18) is the buoyant force equation and uses the virtual potential temperature, $\theta_{v0}$, and $\theta_v$ as the reference potential temperature. While $g$ is the acceleration due to gravity and $q_h$ is the mixing ratio of hydrometeors. The mixing ratio of hydrometeors is calculated by the water continuity equations (11 - 21) and are solved with $C$ the condensation of vapour, $A$ as the autoconversion rate, $E_c$ and $E_r$ are evaporation variables. $K$ and $F$ are the collection of cloud water and the rain fallout respectively.

$$\frac{dq_v}{dt} = -C + E_c + E_r, \tag{19}$$

$$\frac{dq_c}{dt} = C - A - K - E_c \tag{20}$$

$$\frac{dq_r}{dt} = A + K + F - E_r \tag{21}$$

As well as needing the water continuity equation to solve the buoyant force equation the thermodynamic equation (22) is used to solve the virtual potential temperature each frame. $L$ is the latent heat, $c_p$ is the heat capacity of dry air, and $\Pi$ is the Exner function.

$$\frac{\partial \theta}{\partial t} + \left( \mathbf{u} \cdot \nabla \right) \theta = \frac{-L}{c_p \Pi} \left( \frac{\partial q_v}{\partial t} + \left( \mathbf{u} \cdot \nabla q_v \right) \right) \tag{22}$$

Each frame of the application the equations from the previous section are solved and the velocity vector for the fluid equation gets updated these values are then used to render the cloud. Figure 4 shows pseudo code for the CUDA kernel calls for each frame. It starts by first advecting the velocity and temperature resources which are used as part of the fluid dynamic equations and thermodynamic equation respectively. Next a check to see if boundary variables need updating after allotted amount of time.

```
\\After initialising the textures once now advect velocity and temperature
    textures
advect velcoity();
advect temperature();
if(timer){
        //update boundary variables to be passed to correct functions
        update variables;
}
//calculate forces
calculate vorticity();
calculate buoyancy();
//update advect velocity with the recently computed forces
update advect velocity();
//update water continuity and temperature values
update water continuity();
update temperature();
//compute divergence of advect velocity
compute divergence();
//Jacobi solver to solve Poission pressure equation
for 20 times{
        jacobi solver();
}
//update velocity texture by subtracting the pressure texture from the divergence
    texture
update velocity();
//flatten the 3D rain texture to a 32*32 texture to be used to calculate the rain
    position and amount
flatten rain();
//copy the CUDA textures to a DirectX texture for rendering and a float array for
    rain generation
copy velcoity texture();
copy rain texture();
//update rain particles based upon data received from the rain texture
update rain system();
```

Figure 4: Main CUDA Frame Pseudo Code

Continuing from this the extra forces are calculated, vorticity confinement and the buoyant force, and are added to the advected velocity field. Next the water continuity and thermodynamics are updated for the next iteration. The divergence of advected velocity field is calculated next with a Jacobi solver used to calculate the pressure resource. The new velocity data is calculated by subtracting the pressure resource from the divergence resource. The rain data is now flattened to

a smaller texture to be copied to an array used by the particle systems. As well as coping the rain to an array the velocity resource is copied to a DirectX texture.

```
//render terrain and particles
render terrain();
render particles();
//render cloud using volume rendering
//start by rendering front and back facing bounding cube textures
render front cube();
render back cube();
//use front and back cube to calculate ray direction and iterate through volume
    updating final colour
render volume();
```

Figure 5: Render Pseudo Code

Figure 5 shows the rendering function used each frame. Firstly the terrain is rendered using a simple texture shader. Next the particles are render if there is any rain in the system to be rendered. Then the volume is rendered to the scene.

```
//calculate projective texture coordinates
//used to project the front and back position textures onto the cube
Set Texture Position Coordinates;
Normalize Direction between back and front textures;
Set starting position to be at the front texture;

Set source colour;
Set final colour;
Set temp colour;
Set step size for stepping through ray;

for 0 to iteration{
        set temp colour from volume texture;
        set source colour based upon temp colour;
        Update final colour based upon source colour;
        if final colour alpha > 1{
                exit for loop;
        }
        update position based upon step size;
        if pos > then texture coord{
                exit for loop;
        }
}
return final colour;
```

Figure 6: Volume Rendering Pseudo Code

For the volume rendering of the cloud a process from Hayward (2009) is used which is called Volume Ray Casting. This process consists of two passes that are computed each frame. Firstly two textures are created one with front facing faces turned on and the second with back faces turned on. These two textures are passed to the second part of the process as defined by Figure 6 where the front facing

cube is the starting location of the ray and its direction is calculated by taking away the front facing cube from the back facing cube. This process creates a ray which when iterated through will blend the colour at each step resulting in a 3D representation of the cloud.

A pseudo representation of the advection CUDA kernel for the velocity can be seen in Figure 7. It starts by calculating where the velocity will be transported to at the next time step. Using this position in the texture an interpolation is done using the surrounding array elements. The boundaries for velocity are also set here with the bottom boundary being a no-slip boundary condition, which means that the velocity equals zero. The boundary for the top however is a free-slip boundary where when the top two levels are added together equal zero. The remaining boundaries are all randomised values indicating wind.

```
//update position based upon where velocity will be next time step
Update Position;
\\Get the surrounding values
Interpolate values to get advection velocity;
Update Boundaries;
```

Figure 7: Advection Pseudo Code

The Figure 7 will be very similar for advecting the temperature with the main difference being the different boundary conditions. In advecting the temperature all but the bottom boundary equals the ambient temperature, which has a predetermined constant value.

```
Get surrounding values;
Perform curl on the vector;
Create a normalized version of the vector;
Store curl vector and magnitude to texture;
```

Figure 8: Vorticity Confinement Pseudo Code

Figure 8 and 9 shows the two kernel used for solving the vorticity confinement. The first kernel is used to calculate the curl of the velocity and the normalization of this vector. These two variables are passed two the next kernel as a texture and are then used to calculate the final vorticity confinement vector which is added to

the advected velocity field.

```
Compute Divergence of the magnitude;
Calculate Cross product of Curl vector and divergence;
Multiply value by time step and grid scale
Add final value to advection velocity;
```

Figure 9: Vorticity Confinement Pseudo Code

```
Divide potential temperature by ambient temperature;
Minus water condensation rate;
Multiply by gravity and time step;
Update Advected velcoity field;
```

Figure 10: Buoyancy Force Pseudo Code

The next kernel in the process of simulating the clouds s the buoyant force kernel and can be seen in Figure 10. This kernel takes in two textures as inputs for variables and the advected velocity for an output. Figure 11 and Figure 12 show the pseudo code for updating water continuity textures and the temperature texture. The mixing ratios for water vapour, cloud water and rain water are all updated as described by Houze (1994). In this kernel the boundaries are also calculated with the mixing ratio for cloud water as zero. The boundaries for the mixing ratio of water vapour are defined as zero at the top, a constant at the bottom of the system, while the sides of the system are randomized.

```
Calculate K;
Calculate F;
Calculate A;
if water condensation > threshold value{
        A = (water condensation - threshold value)*alpha
}
Calculate temperature;
Calculate C;
Update Water Vapour;
Update Water Condensation;
Update Rain Mixing ratio;
Store Water Continuity variables;
Update Boundary conditions;
```

Figure 11: Water Continuity Pseudo Code

The temperature is calculated by solving the equation (9) from section 2.2.3 with the condensation rate being exchanged for the part in brackets.

```
Calculate Latent Heat divide by heat capacity;
Calculate Exner function;
Divide by Exner function;
Multiply by condensation rate and time step
Update temperature with advection temperature + previous variable
```

Figure 12: Temperature Pseudo Code

The Figure 13 shows the combination of the divergence kernel and the Jacobi kernel. The divergence kernel calculates the divergence of the avected velocity texture. The main body of the Jacobi kernel is located in a for loop which runs a number of different times. Next the boundaries for the pressure is calculated this is done by setting the end value to be the same as the next value in for example the pressure at the bottom of the system is the same as the pressure one level up. Central Finite Difference is used to calculate the pressure.

```
Calculate divergence of advection velocity;
for number of iterations{
        Update pressure boundaries;
        Compute Central Finite difference on pressure field;
        Store updated value;
}
```

Figure 13: Divergence and Jacobi Pseudo Code

The final Figure 14 shows the process for calculating the new velocity by taking away the pressure from the advected velocity. Finally the value used for the volume rendering is calculated by calculating the magnitude of the velocity vector.

```
Calculate div of pressure;
Calculate advection - div pressure;
Update velocity;
Store magnitude of vector as alpha channel;
```

Figure 14: Final Pseudo Code

## 3.2 Evaluation Methodology:

Evaluating the application is done by two methods both based upon methods found in the literature review. The first is a visual test at time steps during the

running of the program. This will showcase the formation and deformation of clouds and the generation of rain. Figure 15 shows Dobashi *et al.* (2000) results in the manner describe here. In the appendix Figure 18 to Figure 20 show the results from other models. This method has been chosen because it shows of the purpose of the project which is to have visually realistic clouds simulated at real time with the creation of rain.

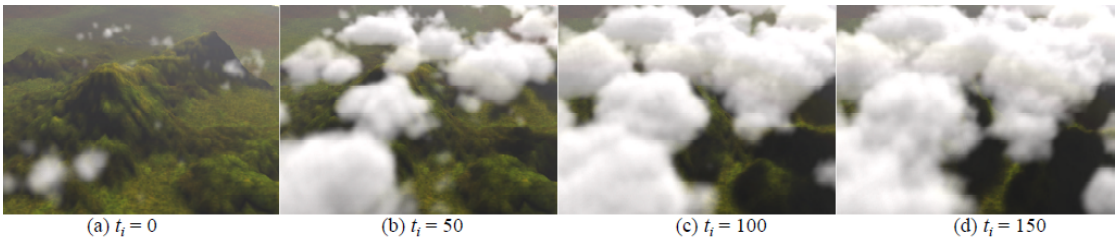


Figure 15: Dobashi *et al.* (2000)

The other method for testing the application is to test how much resources are being used by the application. Harris and Lastra (2001), and Elek *et al.* (2012) both used this method when evaluating their models. This form of evaluation tests how efficient the application is when running in real time as well as concerns like the size of grid that can be used. This can be accomplished using Nsight Visual Studio Edition (2013b) for debugging CUDA and profiling the application.

# 4 Results:

As described in section 3.2 the application has been tested in two different ways the first visually at a number of different time steps. Secondly by evaluating the resources the application uses whilst running. For testing the application a laptop running Windows 8.1 Pro 64-bit, and containing and Intel Core i7 3632QM @ 2.20GH CPU with 8.00GB DDR3 @ 788MHz memory and a 2GB NVIDIA GeForce GT 740M graphics card.

## 4.1 Visual Comparison:

Figure 16 shows the application after 773 seconds, in the Appendix Figure 21 shows the application at four different time steps. The cloud texture size in these screenshots is $64 * 64 * 64$.

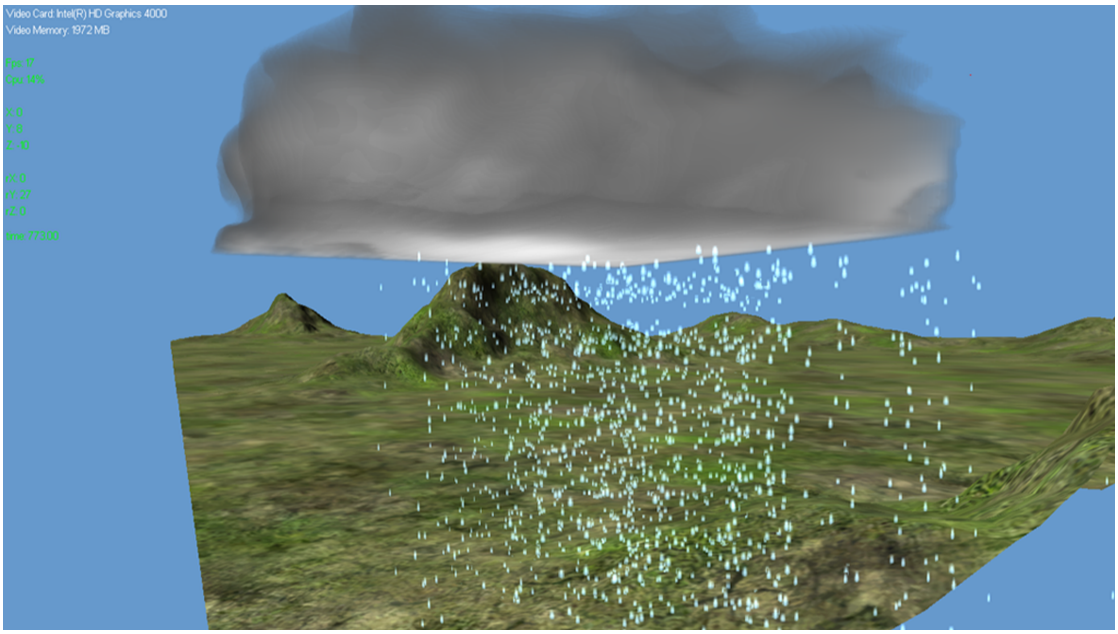


Figure 16: $t = 773s$

## 4.2 Resource Usage:

Table 1 shows the application using a number of different grid sizes for the textures used in the application. For a number of these different grid size the amount of

Table 1: Texture Grid Size Performance Analysis

| Grid Dimensions (Width * Height * Depth) | Jacobi Iterations | FPS(avg) | GPU (%) | CPU (%) |
|---|---|---|---|---|
| 32*32*32 | 16 | 60 | 25.8 | 46.6 |
| 32*32*32 | 20 | 57 | 32.2 | 57.0 |
| 32*32*32 | 32 | 50 | 32.3 | 55.4 |
| 64*64*64 | 16 | 30 | 41.6 | 58.3 |
| 64*64*64 | 20 | 26 | 45.9 | 64.4 |
| 64*64*64 | 32 | 18 | 51.0 | 61.9 |
| 128*128*128 | 16 | 5 | NA | 54.8 |
| 64*64*32 | 16 | 38 | 37.1 | 51.2 |
| 128*128*32 | 16 | 16 | 31.6 | 39.6 |
| 128*128*64 | 16 | 10 | 29.8 | 32.5 |

Jacobi Solver iterations is slightly different to see how changing this impacts the efficiency of the application. The values for the different iterations are 16, 20, and 32 and they are used for the $32 * 32 * 32$ and $64 * 64 * 64$ grids.

With the different grid sizes Table 1 shows how many frames per second (FPS) the application runs at. This was calculated by running the application a number of different times and averaging the FPS. The GPU and CPU columns are values taken from the Nsight Visual Studio Edition performance analyser. These values represent the utilization of either the GPU or CPU while the application is running. These values may not be exact to when the application is running due to Nsight performance analyser using resources that wouldn't normally be used.

Table 2: CUDA Kernel Resource Usage

| **CUDA Kernals** | **Duration (avg $\mu s$)** | **Occupancy (%)** |
|---|---|---|
| Advect Velocity | 3835 | 62.5 |
| Advect Temperature | 2079 | 100 |
| Vorticity Confinement | 1284 | 100 |
| Forces | 1816 | 100 |
| Buoyancy Force | 3222 | 100 |
| Water Continuity | 10125 | 75 |
| Update Temperature | 2584 | 100 |
| Divergence | 839 | 100 |
| Jacobi Solver | 18490 | 100 |
| Cloud Velocity Update | 1577 | 100 |
| Flatten Rain | 223 | 100 |

Table 2 shows the time on average each kernel takes every frame whilst the application is running. The time take is in $\mu s$ and is taken from the Nsight performance analyser using a texture size of $64 * 64 * 64$ and 20 Jacobi Solver iterations. The Occupancy column shows how well each kernel uses the resources of the device. NVIDIA (2012) states that "higher occupancy does not always equate to higher performance ... low occupancy always interferes with the ability to hide memory latency, resulting in performance degradation." There are some kernels not in the table such as the initialise kernels, as they do not affect the running of the application at each frame.

# 5 Discussion:

This section will discuss the visual and quantitative results gathered in the previous section. Starting with the visual results seen in Fig 16 and 21 show that the fluid equations, with the added water continuity and thermodynamic equations completes the objectives that are need to answer the research question mentioned in section 4.1. As seen in the screenshots the clouds created grow, move and dissipate over time. As well as this the equations also allow the creation of rain depending on the cloud properties. This is not to say that the clouds are perfect, for example they are quite dark at times because of the rendering technique used but could be improved by using either single scattering or multi-scattering as described in section 2.4. The reason behind not using them is because the either of the process would use more resources that will slow down the application further.

The rain used in the application is a very basic particle system that uses billboard texture and instancing to allow for a large number of rain particles to be on screen at once with the amount of rain at each located is based upon the mixing ratio of rain calculated using Houze (1994) Bulk Water Model. Even though the amount of rain is directly related to the cloud system improvements could still be made to the rendering technique used. For example light scattering could be added to the rain so that it starts to look more realistic. As well as improving the rendering of the rain improvements could be made to the physical simulation of the rain for example the addition of wind and other external forces. These forces could directly relate to the forces that move the cloud around the 3D scene.

Whilst comparing the visual aspects of the application shows how using fluid dynamic equations allow for the creation and movement of clouds in real-time as well as the creation of precipitation directly related to this system, the quantitative data shown in section 4.2 show how efficient the application is.

In Table 1 as mentioned in section the last two columns are the GPU and CPU Utilization respectively. As shown the utilization increase with both the grid size and the number of Jacobi iterations. Comparing this the frame per second

column the most efficient without the loss of detail grid is the $64*64*64$ grid with 16 Jacobi iterations as it sits around 30 frames per second. That being said the current cloud system is limited by the size of the texture it can used. With graphic cards that have more memory bigger textures can be used and in turn bigger grid sizes can be used. This will allow for either more accurate simulation or allow for a bigger cloud system.

One of the values in the table appears as NA, this is because when using the Nsight Visual Studio Edition performance analyser no value was given. The most likely reason was due to the size of the grid the application is using most of the computers resources which meant the profiler didn't have a enough resources to calculate the percentage. This is the most likely reason as while the application ran the cloud system still updated frame by frame.

The next table, Table 2, as mentioned in section 4.2 states the average duration of each kernel whilst the application is running, as well as stating how close the kernel runs towards the devices threads and registers. Looking at the occupancy first all but two of the kernels run at 100This however doesn't mean that these kernels aren't as efficient as the other kernels. The reason behind this is most likely because of the complexity of the two kernels and the amount of variables need to complete their corresponding process. The two kernels in question are Advect Velocity and Water Continuity. The most likely cause for the Advect Velocity kernel not to have 100% occupancy is because of the interpolation the kernel performs, which requires looking up other parts of the array. The cause for the Water Continuity kernel not having 100% occupancy is most likely that in solve the condensation rate a number of different to the power of functions and exponential functions are called. These functions take time to compute and can slow down the kernel which is why the Water Continuity kernel has the second highest average duration value.

The kernel with the highest duration is the Jacobi Solver kernel. The reason behind this is because inside that kernel is a for loop which performs the Jacobi

Solver 20 times. The reason the Jacobi Solver was chosen over other solvers, such as a multi-grid solver is because it is a relative simple solver. However if the gird size were to increase then "the multi-grid method shows promise for large-grid simulation on the GPU", (Harris *et al.*, 2003).

# 6 Conclusions:

## 6.1 Further Work:

# Appendix:

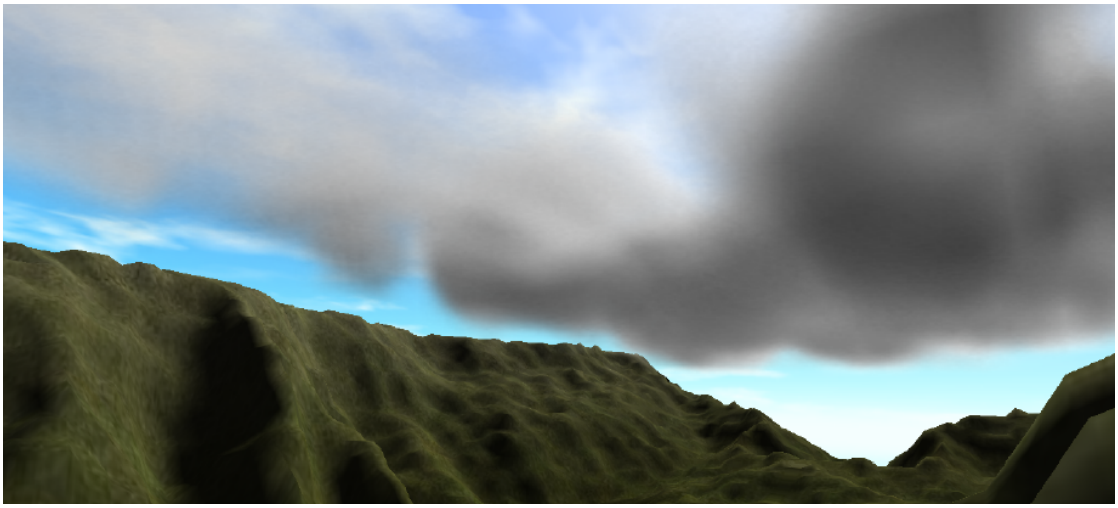

Figure 17: Harris *et al.* (2003)

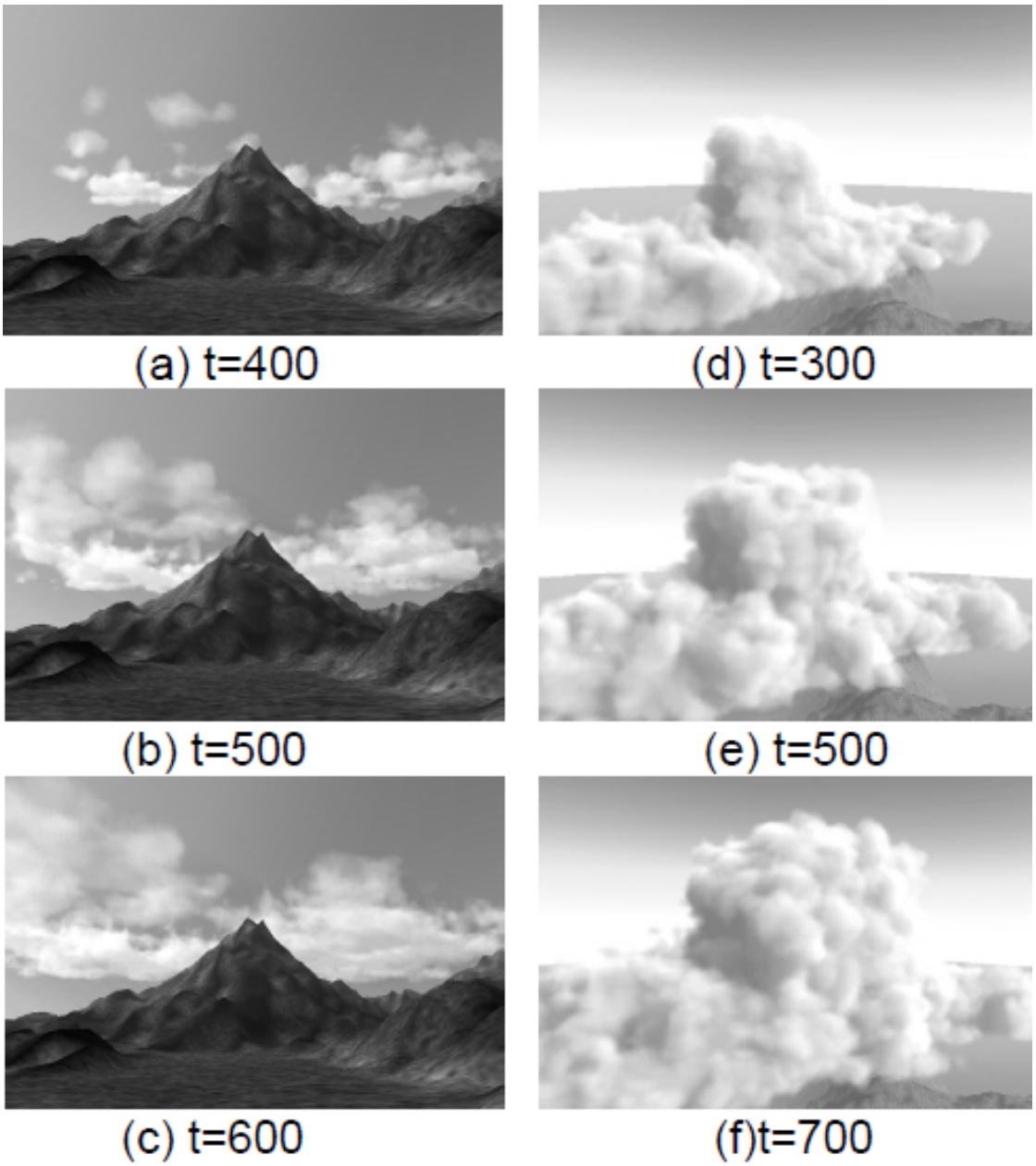


Figure 18: Miyazaki *et al.* (2001)

Figure 19: Fedkiw, Stam and Jensen (2001)

Figure 20: Harris *et al.* (2003)

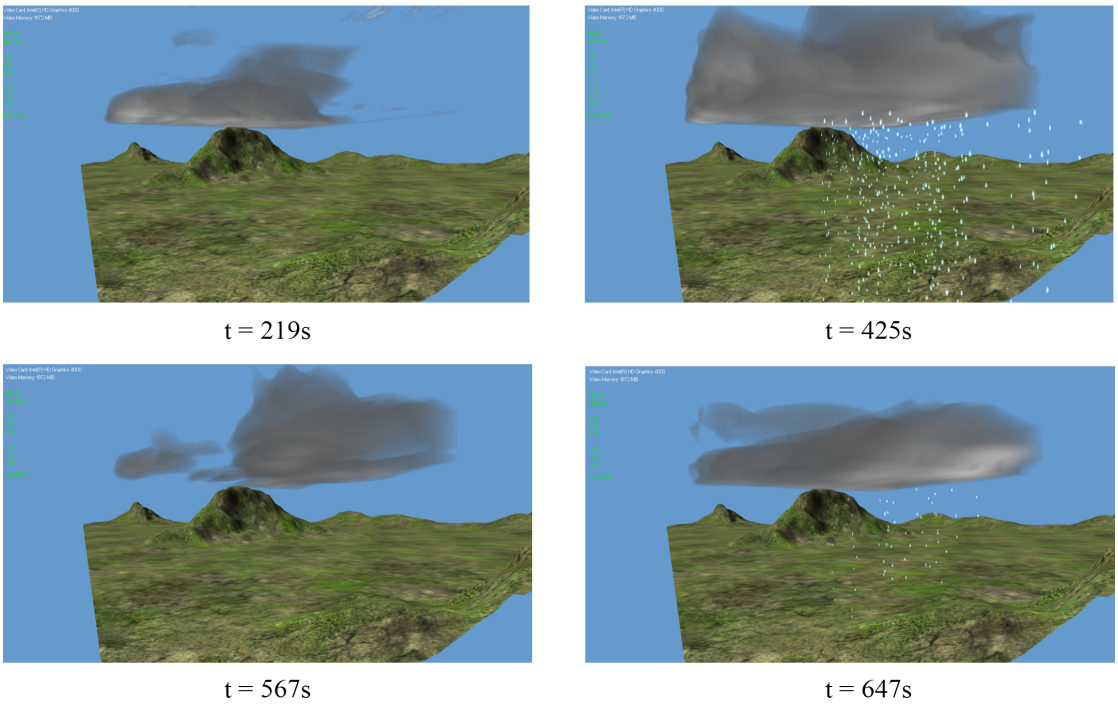


Figure 21: Visual Evaluation

# References


Barton, M. 2008. How's the weather: Simulating weather in virtual environments 8. Available from: http://gamestudies.org/0801/articles/barton. 2.1

Bohren, C. F. 1987. Multiple scattering of light and some of its observable consequences. *Am.J.Phys* 55(6). pp. 524–533. 2.3.1

Corporation, N. 2012. *Profile cuda settings*. [Online]. Available from: http://http.developer.nvidia.com/NsightVisualStudio/2.2/Documentation/UserGuide/HTML/Content/Profile_CUDA_Settings.htm. 4.2

Crytek 3 SDK. 2013. *CryEngine* [software] Version 3. Available from: http://mycryengine.com/. 2.2.1

Dantchev, S. 2011. Dynamic neighbourhood cellular automata. *Computer journal* 54(1). pp. 26–30. 2.2.2

*Dear Esther*. 2012. [Online]. PC. The Chinese Room. (document), 1.1, 1

Dobashi, Y. et al. 2000. A simple, efficient method for realistic animation of clouds. ACM Press/Addison-Wesley Publishing Co. pp. 19–28. (document), 2.2.2, 3.1, 3.2, 15

Elek, O. et al. 2012. Interactive cloud rendering using temporally coherent photon mapping. *COMPUTERS and GRAPHICS-UK* 36(8). pp. 1109–1118. 3.2

Fedkiw, R., Stam, J., and Jensen, H. 2001. Visual simulation of smoke. ACM. pp. 15–22. (document), 2.2.3, 2.2.3, 2.3.2, 2.3.3, 19

Harris, M. et al. 2003. Simulation of cloud dynamics on graphics hardware. Eurographics Association. pp. 92–101. (document), 2.2.3, 2.2.3, 2.2.3, 2.2.3, 2.3.2, 3.1, 3.1, 5, 17, 20

Harris, M. J. 2003. *Real-time cloud simulation and rendering.*. [Online]. 2.3.3

Harris, M. and Lastra, A. 2001. Real-time cloud rendering. *Computer Graphics Forum* 20(3). pp. C76–C76. 2.3.1, 2.3.2, 3.2

Hayward, K. 2009. *Volume rendering 101*. [Online]. Available from: http://graphicsrunner.blogspot.co.uk/2009/01/volume-rendering-101.html. 3.1

*Heavy Rain*. 2010. [Disk]. PlayStation 3. Sony Computer Entertainment. (document), 1.1, 2

Houze, R. 1994. *Cloud Dynamics*. International Geophysics. Elsevier Science. 2.2.3, 2.4, 3.1, 5

Jensen, H. W. 1996. *Global illumination using photon maps*. Springer. pp. 21–30. Rendering Techniques 96. 2.3.3

Microsoft. 2012. *DirectX 11.1* [software] Version 6.02.9200.16384. Available from: http://www.microsoft.com/en-gb/download/details.aspx?id=10084. 3.1

Microsoft. 2013. *Visual Studio 2012* [software] Version no 11.0.61030.00 Update 4. Available from: http://www.visualstudio.com/. 3.1

*Microsoft Flight Simulator 2004: A Century of Flight*. 2003. [Disk]. PC. Microsoft. 1.1, 2.2.1

Miyazaki, R. et al. 2001. A method for modeling clouds based on atmospheric fluid dynamics. In: *Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on*. IEEE. pp. 363–372. (document), 2.2.2, 2.3.1, 3.1, 18

*NCAA Football 14*. 2013. [Disk]. PlayStation 3. EA Sports. 1.1

NVIDIA. 2013a. *CUDA* [software] Version no 5. Available from: http://www.nvidia.com/object/cuda_home_new.html. 3.1

NVIDIA. 2013b. *NSight Visual Studio Edition* [software]. Available from: https://developer.nvidia.com/nvidia-nsight-visual-studio-edition. 3.2

*Ouranos!* 1980. [Cassest]. Commodore PET. The Code Works. (document), 1.1, 1, 1.1

Overby, D., Melek, Z., and Keyser, J. 2002. Interactive physically-based cloud simulation. In: *Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on*. IEEE. pp. 469–470. 2.2.3

Puig-Centelles, A., Ripolles, O., and Chover, M. 2009. Creation and control of rain in virtual environments. *VISUAL COMPUTER* 25(11). pp. 1037–1052. 2.4

*rain*. 2013. [Online]. PlayStation 3. Sony Computer Entertainment. (document), 1.1, 2

RasterTek. 2014. *Raster tek d3d11 tutorials*. [Online]. Available from: http://www.rastertek.com/tutindex.html. 3.1

Stam, J. 1999. Stable fluids. ACM Press/Addison-Wesley Publishing Co. pp. 121–128. 2.2.3, 2.2.3

*Super Mario Bros.* 1985. [Cartridge]. Super Nintendo Entertainment System. Nintendo. 1.1

Tariq, S. 2007. Rain. Tech. rep.. NVIDIA. 2.4

*Tomb Raider*. 2013. [Online]. PC. Square Enix. (document), 1.1, 2.1, 3

Wang, N. 2004. *Let there be clouds! fast, realistic cloud-rendering in microsoft flight simulator 2004: A century of flight*. [Online]. Available from: http://www.gamasutra.com/view/feature/130434/let_there_be_clouds_fast_.php. 2.1, 2.2.1