

Research Proposal

Jake Morey

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

Abstract

Contents

Abstract	i
List of Figures	ii
List of Tables	ii
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:	13
3.2 Evaluation Methodology:	18
4 Results:	19
5 Discussion:	20
6 Conclusions:	20

References	21
Appendix	24

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	12
Figure 5	Render Pseudo Code	12
Figure 6	Main CUDA Frame Pseudo Code	15
Figure 7	Render Pseudo Code	16
Figure 8	Render Pseudo Code	16
Figure 9	Render Pseudo Code	16
Figure 10	Render Pseudo Code	17
Figure 11	Render Pseudo Code	17
Figure 12	Render Pseudo Code	17
Figure 13	Render Pseudo Code	18
Figure 14	Render Pseudo Code	18
Figure 15	Render Pseudo Code	18
Figure 16	Render Pseudo Code	19
Figure 17	Dobashi <i>et al.</i> (2000)	19
Figure 18	Harris <i>et al.</i> (2003)	24
Figure 19	Miyazaki <i>et al.</i> (2001)	24
Figure 20	Fedkiw, Stam and Jensen (2001)	25
Figure 21	Harris <i>et al.</i> (2003)	25

List of Tables

Table 1	Table Title	19
---------	-----------------------	----

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

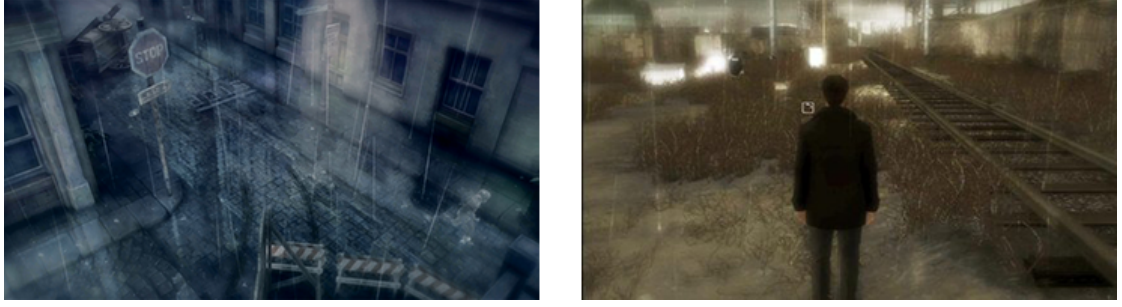


Figure 2: Left: rain (2013), Right: Heavy Rain (2010)

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 discrete 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 ρ is the density, \mathbf{f} represents all external forces and ν 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.

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

$$\nabla \cdot \mathbf{u} = 0 \quad (2)$$

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).

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

$$\nabla \cdot \mathbf{u} = 0 \quad (4)$$

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). ω is the vorticity, defined by $\omega = \nabla \times \mathbf{u}$, and \mathbf{N} is the normalized vorticity vector field and points from areas of lower vorticity to areas of higher vorticity which is defined by equation (6). With h is the grid scale and ϵ is a scale parameter.

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

$$\mathbf{N} = \frac{\nabla |\boldsymbol{\omega}|}{|\nabla |\boldsymbol{\omega}||} \quad (6)$$

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. θ_v is the virtual potential temperature and is defined in equation $\theta_v \approx \theta(1 + 0.61q_v)$. θ_{v0} is the reference potential temperature and is between 290 and 300K as defined by Harris *et al.* (2003).

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

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.

$$\frac{Dq_v}{Dt} = -C, \frac{Dq_c}{Dt} = C \quad (8)$$

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 $2501 Jkg^{-1}$. 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.

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

Π 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.

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

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) states 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.

$$\frac{dq_v}{dt} = -C + E_c + E_r \quad (11)$$

$$\frac{dq_c}{dt} = C - A - K - E_c \quad (12)$$

$$\frac{dq_r}{dt} = A + K + F - E_r \quad (13)$$

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:

- choice of practical and evaluation methods
- ethical considerations
- pseudo code

```
\\After initialising the textures once now advect velocity and temperature
textures
advect velocity();
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 velocity 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

```
//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

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. This application has been built upon a free tutorial framework called Raster Tek, As well as using C++ and the Raster Tek framework Nvidia's CUDA The DirectX 11

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 3.1. Snapshots from Harris *et al.* (2003) and Miyazaki *et al.* (2001) are both shown in the appendix as Figure 18 and Figure 19 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).

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

$$\nabla \cdot \mathbf{u} = 0 \quad (15)$$

$$\mathbf{f} = \mathbf{f}_{vc} + \mathbf{B} \quad (16)$$

Where ρ 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.

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

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

The other equation 18 is the buoyant force equation and uses the virtual potential temperature, θ_{v0} , and θ_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.

$$\frac{dq_v}{dt} = -C + E_c + E_r, \quad (19)$$

$$\frac{dq_c}{dt} = C - A - K - E_c \quad (20)$$

$$\frac{dq_r}{dt} = A + K + F - E_r \quad (21)$$

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 Π is the Exner function.

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

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 6 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 6: 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.

Figure 7 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

```

//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 7: Render Pseudo Code

any rain in the system to be rendered. 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 8 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.

Figure 8: Render Pseudo Code

A pseudo representation of the advection CUDA kernel for the velocity can be seen in Figure 9. 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.

Figure 9: Render Pseudo Code

The Figure 9 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.

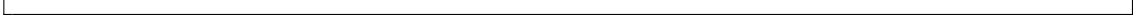


Figure 10: Render Pseudo Code

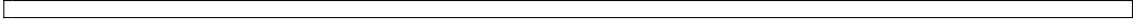


Figure 11: Render Pseudo Code

Figure 10 and 11 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 to 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.



Figure 12: Render Pseudo Code

The next kernel in the process of simulating the clouds is the buoyant force kernel and can be seen in Figure 12. This kernel takes in two textures as inputs for variables and the advected velocity for an output. Figure 13 and Figure 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. 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.

Figure 13: Render Pseudo Code

Figure 14: Render Pseudo Code

The Figure 15 shows the combination of the divergence kernel and the Jacobi kernel. The divergence kernel calculates the divergence of the advected 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.

Figure 15: Render Pseudo Code

The final Figure 16 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.

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 17 shows Dobashi *et al.* (2000)) results in the manner describe here. In the appendix Figure 19 to Figure 21 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.

The other method for testing the application is to test how much resources are

Figure 16: Render Pseudo Code

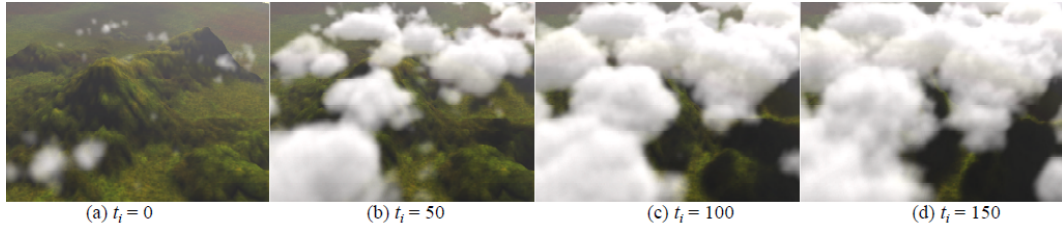


Figure 17: Dobashi *et al.* (2000)

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 NVIDIA (2013) for debugging CUDA and profiling the application.

4 Results:

- Visual Test with screen shots at different time to show cloud and rain progression
- Table showing frame rates, cpu/gpu utilisation, memory usage
- talk about any problem with applications and results

Table 1: Table Title

Grid Dimensions (Width * Height * Depth)	FPS	GPU Utilization	CPU Utilization
64*64*64			
64*64*32			
128*128*128			
128*128*64			
128*128*32			

5 Discussion:

- talk about the results
- how it affected the topic area
- talk about how aims and RQ was answered

6 Conclusions:

- summaries findings
- clarify questions relating with previous work
- how the project could be improved or extended

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
- 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, 17
- 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, 20
- 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, 18, 21
- 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
- Jensen, H. W. 1996. *Global illumination using photon maps*. Springer. pp. 21–30. Rendering Techniques 96. 2.3.3
- 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, 19
- NCAA Football 14*. 2013. [Disk]. PlayStation 3. EA Sports. 1.1
- NVIDIA. 2013. *NSight Visual Studio Edition* [software]. Available from: <https://developer.nvidia.com/nvidia-nsight-visual-studio-edition>. 3.2
- Ouranos!* 1980. [Cassette]. 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
- 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

Appendix:

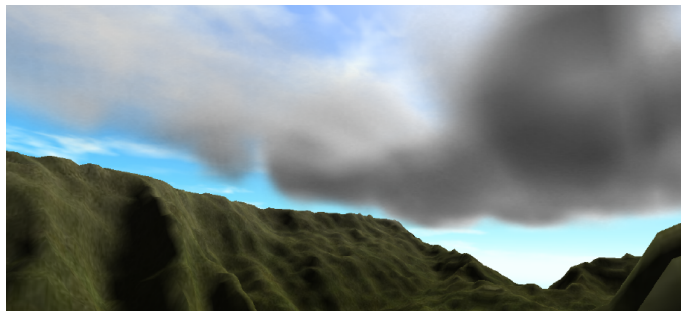


Figure 18: Harris *et al.* (2003)

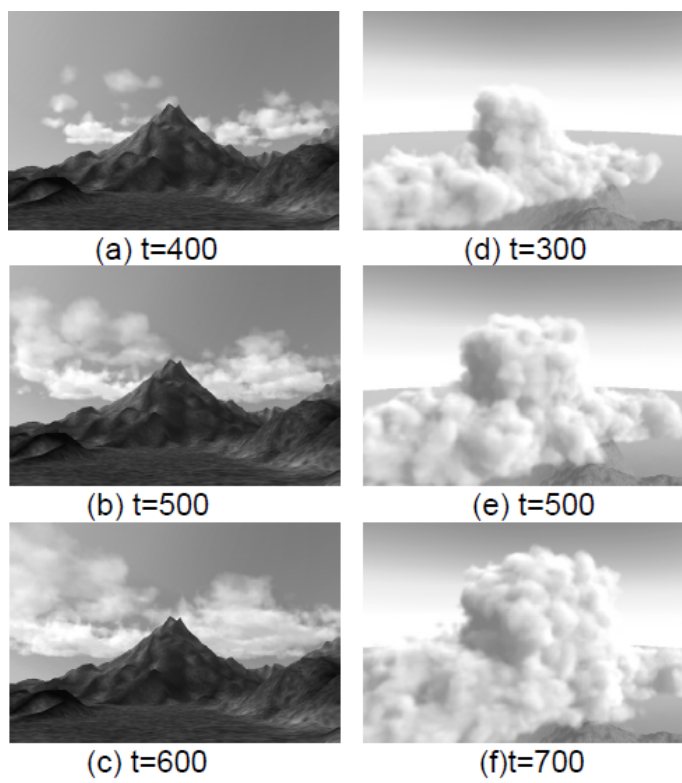


Figure 19: Miyazaki *et al.* (2001)



Figure 20: Fedkiw, Stam and Jensen (2001)



Figure 21: Harris *et al.* (2003)