**Module:** UFCFS4-30-3 Creative Technologies Project

**Student Name:** Gundars Pelns

**Student ID:** 15014193

**Project Supervisor:** Tom Bashford-Rogers

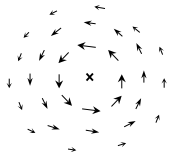
**Tile of the Project:** Real-Time Fire Simulation using the DirectX Toolkit.

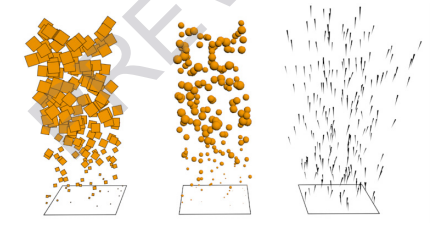
# **Project Summary**

This project will represent how materials slowly burn as the fire spreads across the object deforming it depending on the objects material type. By the final submission, the simulation should allow the user to manipulate the fire as well as the objects that can be set alight.

# **Literature Review**

Throughout the researching process a lot of useful websites, articles and journals about different techniques of rendering fire in real time were found. Some of these articles and journals had very interesting tips about the way a fire could be simulated, as well as what to keep in mind when simulating a fire, as it could require a lot of computational processing power.

One of these articles was particularly interesting because it spoke about the different ways of implementing a fire simulation, however this was mostly aimed for rendering this fire simulation using the ‘Eigenfires’ technique. Which is “a way to represent distinguishable and meaningful features of various fire samples” – [3]. Although it is using this technique, the article by (Kjærnet, 2010) has some interesting ways of creating a fire simulation. Such as using a particle system, which is one of the most popular ways of creating a fire or most other visual effects as such, simply because particle systems are usually quite simple in the way that they are processed. However they require a lot of unique equations and behaviours as they usually don’t look very realistic without these complex behaviours, such as ‘Billboarding’ as this allows you to use simple 2D textures that always face the camera. There are ways to fix this however, by possibly including noise maps, alpha textures or even scrolling textures. These effects will give the particles a more realistic feel to them.



The image on the right shows different types of particles, as you can see the one on the left is using 2D objects that simply point to the camera so that you can’t see that they are 2D. Middle one is using a 3D sphere technique, this can be more useful because of the circular shape and the possibilities of adding interesting shaders and textures on these spheres. The image on the right is a smaller 2D sprite, almost like a line, which are quite useful for effects such as rain. In the article (Nikfetrat, 2013), it mentions a way of creating “a basic 2D particle system using a vortex field” – [4], which sounds very intriguing as vortex fields are used to simulate fire and smoke. The paper describes how the vortex field works, it is simply a field of turbulence that moves along in a specific way. By looking at the image on the right you can see a representation of a 2D vortex field.

The paper also describes a method called ‘Texture Splats’, this technique provides smoother edges on the fire, where splats partially overlap. These splats can be associated with particles by blending them together which then causes the turbulence in the fire to be more visualized. These texture splats have the capability to be procedurally generated by using noise functions such as perlin noise, or can even be rendered from real fire images.

A paper called “Directable, High-Resolution Simulation of Fire on the GPU” – [1] by Christopher Horvath and Willi Geiger (Christopher Horvath, 2009) describes a fire simulation in a lot more depth, such as the exact fluent movement techniques and even more complex ways of rendering fire such as the ‘Coarse Grid Step’. The Coarse Grid Step technique is very useful because it allows for a quick and effective solution for an approximate non-divergent velocity for the particles, while simultaneously adding a selectable amount of viscosity. So this allows the simulation to be altered with on run time, which can be very useful for specific simulations or even simulations that allow for users to interact with the environment. These grids are rather small to allow for more dynamic movement across the simulation, as this will cause the particles to move in different directions but together at the same time.

Fine detail is then added on the “GPU-based refinement stage”-[1] so that the coarse grid step only requires to capture broad, direct motion as the GPU will then do the more complex calculations. The paper also includes important vector attributes that will be needed, such as position and velocity. As well as scalar values like radius, fuel, mass and age. These are the main features that are required for a more complex fire, as it will behave more realistically and have more attributes that can be altered with.

# **Research**

Most of the research that was found was for more complex simulations that had the foundations already set up. This required for more in depth search about the DirectX engine, and how to set one up so that it can be used for developing the fire simulation. There is a wonderful website dedicated to the DirectX engine, there are tutorials for setting up DirectX9 – 11 that are quite confusing but rather helpful. This would be the ‘RasterTek’ – [5] (Unknown, January 3rd, 2016) tutorials, where you can find a whole load of different types of things that you can achieve by using the DirectX engine. Starting from getting a window to render, to having multiple texture arrays that can achieve unique features, such as a glass effect.

It also has a whole tutorial on implementing a simple particle system skeleton. As the first main question was, what type of fire simulation should be used? It was decided to simply use a particle system for now, as that is one of the more straight forward techniques to use. It will also help figure out the behaviour of the fire a little more. Multiple tutorials are useful here for the project, however they all overlap each other in a way, which means that it is quite important to not overwrite code that is already being used for something else. This turned out to be quite challenging as the project was running on a completely fresh engine that used some of the DirectX features (Mostly for rendering the different objects and shaders). Through multiple errors and crashed, it was easier to identify why adding things to the simulation started to break things. As one of the issue was that only one version of the model could be loaded in, however after multiple different attempts of trying to fix this issue. It was discovered that the issue was with the ‘vertex buffer’ and that it was being over written each time something new was being rendered. This issue has been fixed by allowing each of these to render once thus not overwriting each frame. However this should only be a temporary fix and should be fixed by altering the vertex buffer correctly.

Although that has been the main issue with the simulation, some research has been made on what to keep in mind when developing a fire simulation. Most of this information was taken from a “Real-Time Fire Simulation for Video Games” – [2] presentation from the internet (Green, 2015). This presentation represents multiple tips on how to create a ‘realistic’ fire as well as the importance of optimization. It talks about features such as glow, motion blur and how you could use a black and white image to represent the change of temperature across the fire over time.

# **References**

Christopher Horvath, W. G., 2009. Directable, high-resolution simulation of fire on the GPU. *Transactions on Graphics,* 28(3), pp. 2-3.

Green, S., 2015. *Flame On: Real-Time Fire Simulation for Video Games.* s.l., GPU Technology Confrence.

Kjærnet, Ø., 2010. *Framework for real-time forest fire,* s.l.: Norwegian University of Science and Technology.

Nikfetrat, N., 2013. Fire pattern analysis and synthesis using EigenFires and motion transitions. *Computer Animation & Virtual Worlds,* 24(3-4), p. 441.

Unknown, January 3rd, 2016. *RasterTek.* [Online]   
Available at: http://www.rastertek.com/index.html  
[Accessed 1 January 2018].

**Appendixes**

-Progress Log

|  |  |
| --- | --- |
| **September** | Started the implementation of the engine itself, was having troubles so took me the whole month to get a working window on the screen that is able to render a simple object. |
| **October** | Once the window was rendering I was able to start implementing more complex models, such as a cube, and then create a large model with the smaller models. |
| **November** | My plan then was to implement a simple particle system that looked like fire, however I was having trouble with rendering it. |
| **December** | Figured out the issue of why it wasn’t rendering, however I do not know how to fix this issue with the vertex buffer |
| **January** | Manage to get two different models rendering on the same buffer however now the particle system is lost and will need to figure out a better solution for this fix. |