MSc Computer Animation and Visual Effects

Masters Project

Implementing Flame using a Mass Spring System

August 2019

Contents

Figures

Tables

Equations

Abstract

Fire simulation is a topic in computer graphics that has multiple different methods of being achieved. The different methods have 3 main categories; texture, particle and physics based **[Ren & Jin & Chen 2009]**. These tend to be used for different purposes, with physics-based systems being used for realism and texture-based systems being used for real time applications. A paper from 2006 proposed a method of fire simulation by using a mass spring method **[Balci & Foroosh 2006]**. This project is to implement flame using the same concept as this paper and to see if this method is effective with modern day machines.

Introduction

A paper from 2006 proposed a method of fire simulation by using a mass spring method **[Balci & Foroosh 2006]**. The mass spring method is a method that is typically used to simulate cloth and the paper from 2006 claims that the idea is based on the old physical arrangement of a Silk Torch, where cloth is used to give the illusion of flames **[Balci & Foroosh 2006]**.

Using mass spring systems to create a cloth simulation is concept that has been in computer graphics for a long time. A paper on the topic was published in 1987 about Elastically Deformable Models using a series of mass, springs and dampers to simulate cloth **[Terzopoulos et al. 1987]**.

The Silk Torch can be traced back to ancient Greece. It is used by magicians to fool spectators into believing that it is a real flame **[Balci & Foroosh 2006]**. It has uses in modern applications such as for theme park rides or just for decorative purposes such as an artificial fireplace **[Balci & Foroosh 2006]**.

This project will be implementing the concept of a Silk Torch using a simpler version of the mass spring method than the one that was used in the original paper from 2006.

*Figure 1: A Silk Torch **[IntaSound P.A 2014]***

Literature Review

Fire simulation has multiple methods that can be used to implement it. The different methods have 3 main categories; texture, particle and physics based **[Ren & Jin & Chen 2009]**.

Possibly the most common method used in real time applications such as computer games is the texture method. The texture method is a simple method where an animated texture of a flame is placed on a plane that is set to always face the camera and is called the billboard method **[Gao & Ma 2010]**.

The method that is used to create the most realistic looking fire is utilising fluid dynamics. The motion of fluids can be calculated by the numerical analysis of the Navier-Stokes equations **[Dobashi 2009]**, however as a by-product of the complexity of the calculation's fluid dynamics are not really used for real time rendering as most machines would not be able to handle it.

The basic principle of a particle system is to use a group of particles to create a fuzzy object **[Qian et al. 2010]**. The particles all have their own properties, such as size and colour and the particles have 3 phases, the production movement and disappearance **[Qian et al. 2010]**. A paper in 2010 talks about combining a particle system with a texture method to try and increase the realism of the particle-based system **[Gao & Ma 2010]**.

For this project the method that will be used will be for cloth simulation, instead of fire simulation. As such it is important to investigate the different types of cloth simulation.

The Mass-Spring model from a paper by Xavier Provot is one that is made up of number of virtual masses and springs, with three different types of spring acting upon it. These springs are called shear springs, structural springs and flexion springs **[Provot 1995]**. The method also uses dampeners for the springs. In figure 2 you can see how the springs are arranged with red lines being the structural springs, the green lines being the flexion springs and the blue lines being the shear springs.

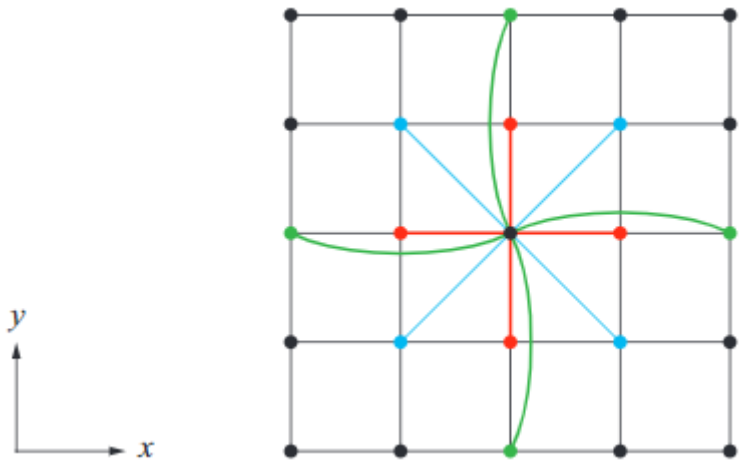

*Figure 2: Springs on a 2D Grid **[Lengyel 2012, pg. 458]***

In order to calculate the mass spring method, then it is important to understand how springs work. The equation that is used to calculate the force on a spring is called Hooke's Law **[Millington 2010, pg.90]**. Hooke's Law works out the value of the spring force by multiplying the negative of the spring constant by the change in length of the spring. This method can be simplified into just using the structural springs along with a damper.

$$f = -k\Delta l$$

*Figure 3: The Equation for Hooke's Law **[Millington 2010, pg.89]***

The paper from 2006 measured the frame rate that the mass spring method to help show the effectiveness of the fire in real time. The frame rate that they managed to get was 80FPS however the specification of the machine they give is 1024 x 768 on a 2.3GHz PC **[Balci & Foroosh 2006]**. As such this project will use this method to help demonstrate if this implementation is an effective one.

Implementation

As the purpose of this project is to test the effectiveness of using a mass spring method for flame simulation the implementation of the code base is quite important. However, as there is more than just cloth simulation required to create this effect, the spring method that has been implemented is a simple mass spring method, rather than an advanced version.

Integration

As part of the mass spring system the mass spring object updates by using integration. This is done by every update loop the internal forces and external forces acting upon the mass points are calculated and the applied. After the forces have been added, the net force is used to update the position of the mass points using a velocity calculated using the mass of the mass points. This is done using **f=ma** which can reordered to get the acceleration as **a=f/m** with the acceleration you can get the velocity using **a*dt**

$$\boldsymbol{force} = \boldsymbol{mass} * \boldsymbol{acceleration}$$

Equation 1: f=ma

$$\boldsymbol{acceleration} = \boldsymbol{force} / \boldsymbol{mass}$$

Equation 2: a=f/m

$$\boldsymbol{acceleration} * \boldsymbol{delta\,time}$$

*Equation 3: a*dt*

Internal Forces

The internal forces of the mass spring object are worked out using the mass spring system. A mass spring object is made up of an std::vector of mass points and a std::vector of springs. As this is a simple version of a mass spring system there are only two types of springs, horizontal and vertical springs.

When the mass spring object is created a grid of mass points are created, the mass points all contain a value for their mass. Then the array of springs is created, every spring has a shared pointer to the mass point A and the mass point B that they are attached to. When they attach themselves to a mass point they send information about the spring back to the mass point so that the mass point has a record of what springs it is attached to.

Every update loop the springs and mass points are updated which is when the spring forces are updated and applied to the mass points. The spring works out it's force by first calculating the current length of the spring by getting the distance between point a and point b attached to the spring. Then the magnitude of the spring force is calculated using Hooke's Law.

$$\boldsymbol{force} = -\boldsymbol{spring\ constant} * \boldsymbol{spring\ length}$$

Equation 4: Hooke's Law [Millington 2010, pg.89]

In this equation the spring length is equal to the current spring length minus the rest length of the spring. The spring direction is then calculated by normalising the resultant vector of the position of point B minus the position of point A. The spring force is then calculated by multiplying the magnitude by the direction.

The spring force on the mass point is based on the if the spring is pointing A to B or B to A, as this effects the direction the spring force is acting upon the mass point.

External Forces

As part of the mass spring system the spring mass object has external forces acting upon it. The external forces that are implemented in this code are wind and the buoyancy of the flame.

When simulating flame using particle simulation, they apply a force of buoyancy **[Qian et al. 2010]** for the up force of the flames. For this code base, instead of having a constant force of gravity acting upon the cloth down, there is instead a force acting up on the cloth that is named buoyancy and with the bottom row of mass points locked the cloth essentially falls upwards giving the impression of the flame rising and lowering. This is applied to the mass spring objects every update loop.

For simulating wind in this code base the wind acts on the mass spring object using an impulse system. This means that the force of the wind is applied to the mass spring object for a certain amount of time, then not applied for a certain amount of time and then applied and so on. This is to give the rolling effect of the wind on the mass spring object to help the flame appear to flicker in the wind. The default time that the wind impulse is on for 1 second and off for 5 seconds.

The force of the wind is a vec3 that applies to the mass points as an external force with its direction and magnitude. The external forces are all added together and stored in the mass points within the mass spring objects.

Textures

In order to create a flame from a cloth simulation the texture is a key part of the process. In the paper that concept of this implementation is based on, they used texture sequencing with variable transparency using a randomly chosen image of a flame **[Balci & Foroosh 2006]**. They found that only a small number of texture images was required, giving an example of using 8 textures **[Balci & Foroosh 2006]**.

However, as this project has a shorter timeframe and has only on person working on it, the method of the texturing for this implementation is a simplified version. The flames are initialised with a random texture of a flame, with 8 possible textures available. While this does not provide as much realism it does give more control to the artists as they can create the flame textures to be used with the implementation. The textures that have been used in this implementation have been generated using Houdini.

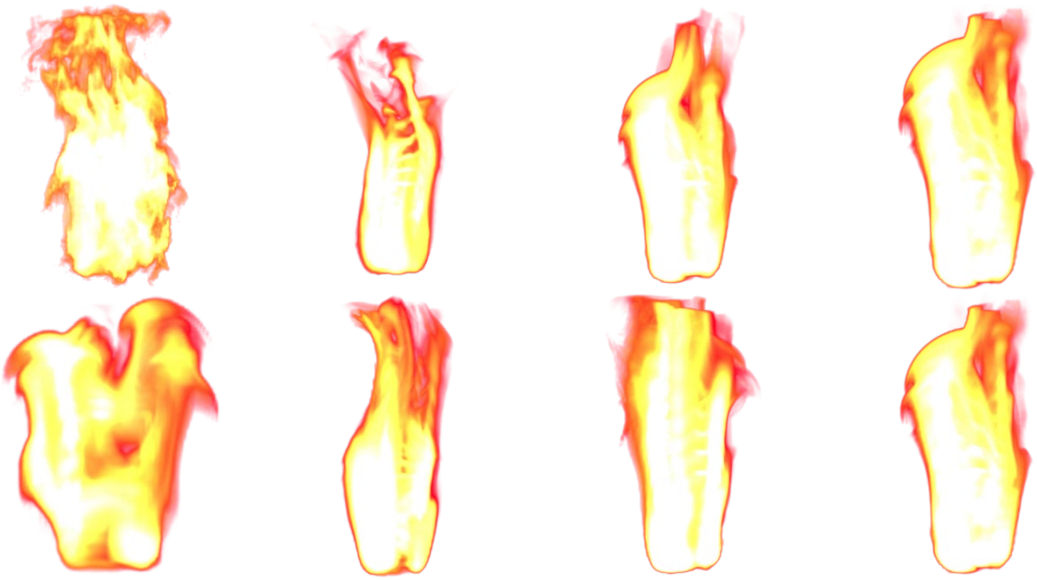

Figure 4: The 8 textures used

In this code base there is also the option of disabling the flame texture to provide a solid cloth texture to show of the cloth simulation underneath.

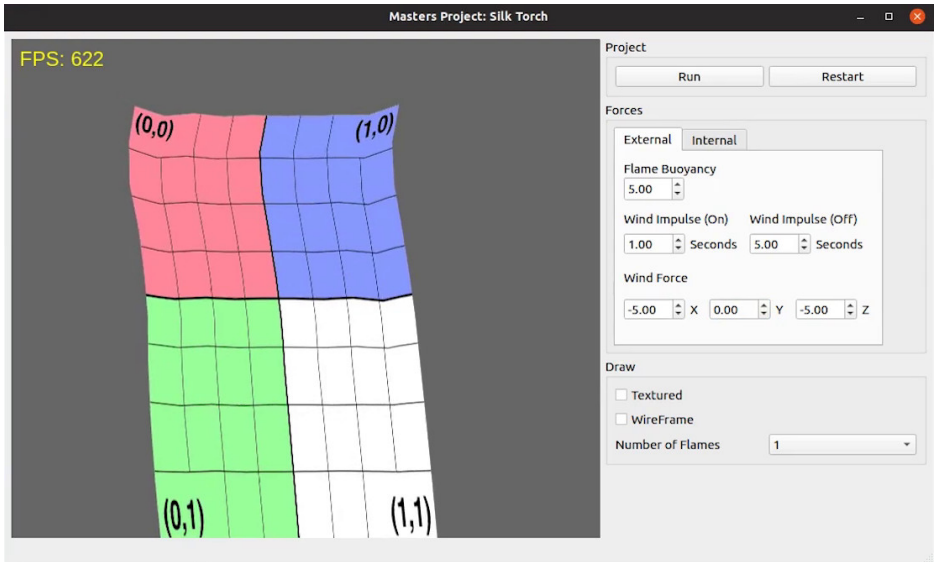

Figure 5: The code running using the cloth texture

User Interface

In order to help with the running the code and testing the method a user interface was required to allow real time control of the flames. The UI is created using NGL working with Qt.

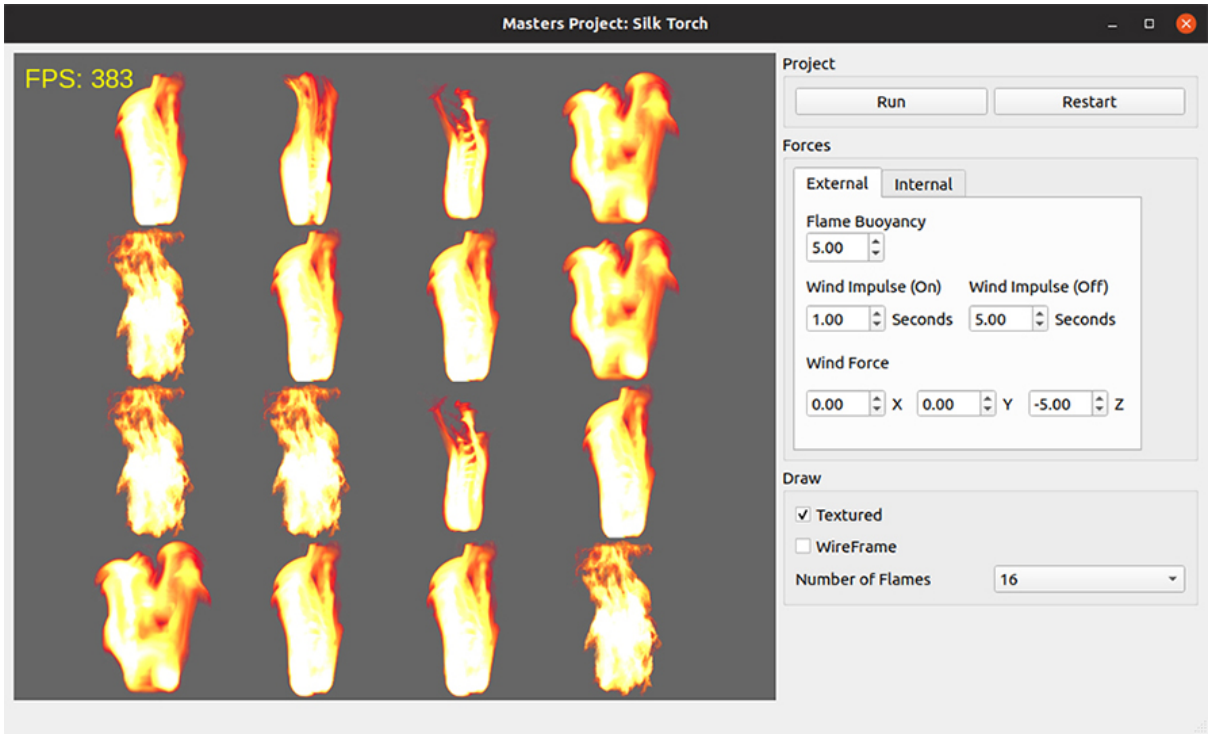

Figure 6: The user interface

The UI has been split into 3 different sections. These sections are Project, Forces and Draw. Inside the Project section there is the button for running the program and a button to reset the currently running simulation.

Figure 7: The Project UI

On the left of the screen there is also a frame rate counter that is for helping collect the performance data of the program. This will display 0 until run has been pressed.

Figure 8: The FPS counter

Under the Forces section is two tabs for the internal and external forces. The internal tab you can change the spring constant, the damping value, the mass and the rest length. The external forces tab contains the flame buoyancy (the force acting up), the time for the wind impulse to be on, the time for the wind impulse to be off and the wind force vector.

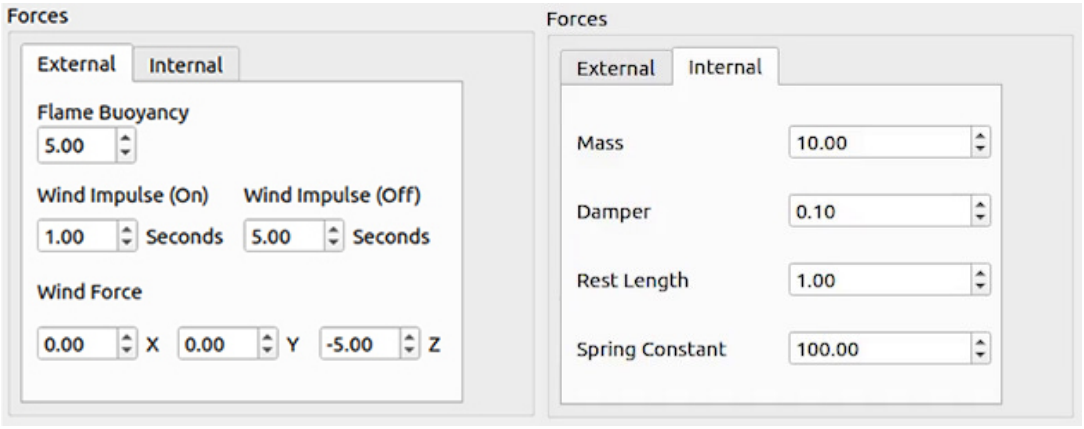

Figure 9: The Forces UI

Under the draw section is the wireframe toggle and a textured toggle that will turn on or off the flame texture. There is also a drop-down box for the number of flames to put in the scene.

Figure 10: The Draw UI

Results

In order to test the effectiveness of using the mass spring method for flame, the code was run on 4 different machines of varying specification. The frame rate that the code ran at was recorded with different numbers of flames on the screen at a time. This was so that it's effectiveness in real time applications could be assessed.

Machine 1 Specifications:
Processor – Intel (R) Xeon (R) CPU E5-1650 v3 @ 3.50GHz
Memory - 32GiB RAM
Graphics - NVIDIA Corporation GM204GL [Quadro M4000]

Number of Flames	FPS 1	FPS 2	FPS 3	FPS Average
1	953	915	934	934
4	907	890	869	888.67
9	670	596	581	615.67
16	347	381	349	359
25	275	236	248	253
36	170	198	187	185
49	156	140	127	141
64	120	110	97	109
81	103	94	99	98.67
100	74	65	67	68.67
121	55	53	57	55
144	45	50	48	47.67

Table 1: The FPS of Machine 1

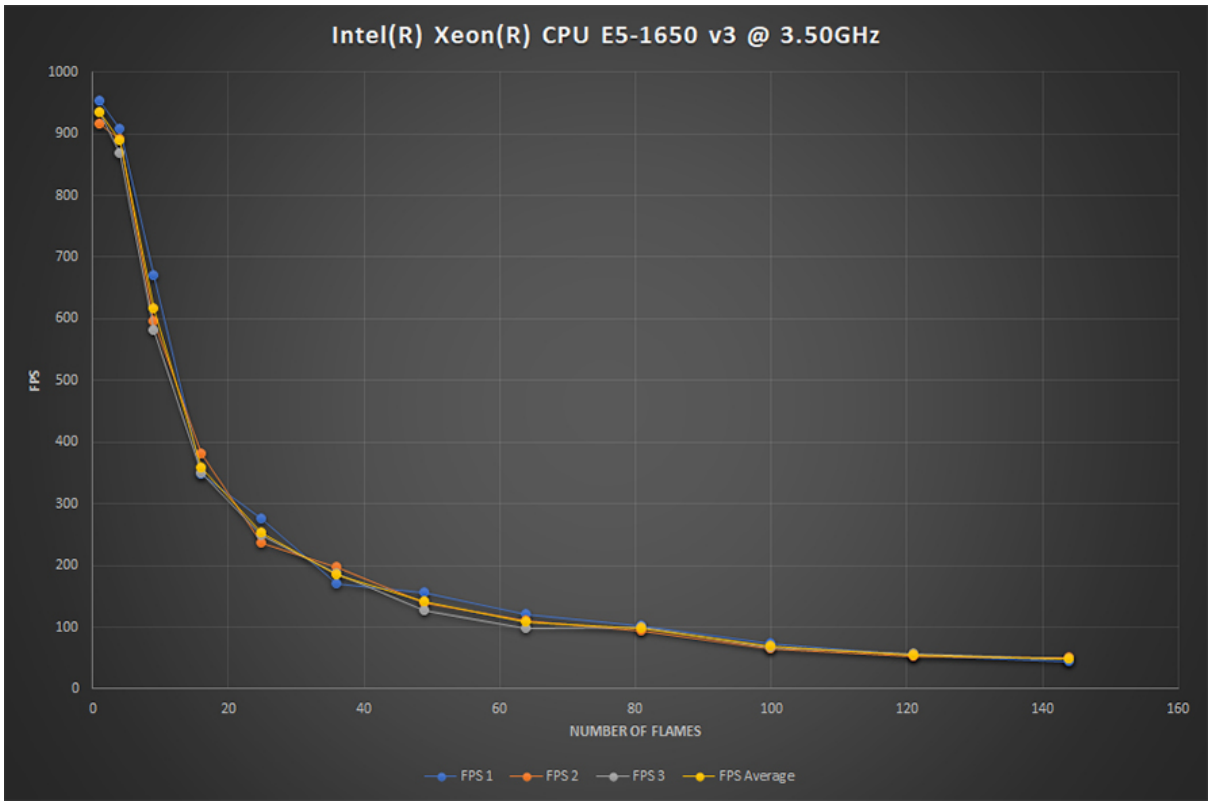

Figure 11: A graph showing the FPS of machine 1

Machine 2 Specifications:
Processor - Intel (R) Core (TM) i7-7700K CPU @ 4.20GHz
Memory - 16GiB RAM
Graphics - NVIDIA GeForce GTX 780

Number of Flames	FPS 1	FPS 2	FPS 3	FPS Average
1	943	940	937	940
4	940	943	934	939
9	903	882	895	893.33
16	559	546	558	554.33
25	377	374	378	376.33
36	270	266	275	270.33
49	203	199	201	201
64	155	154	151	153.33
81	119	125	121	121.67
100	97	100	102	99.67
121	85	84	82	83.67
144	70	71	72	71

Table 2: The FPS of Machine 2

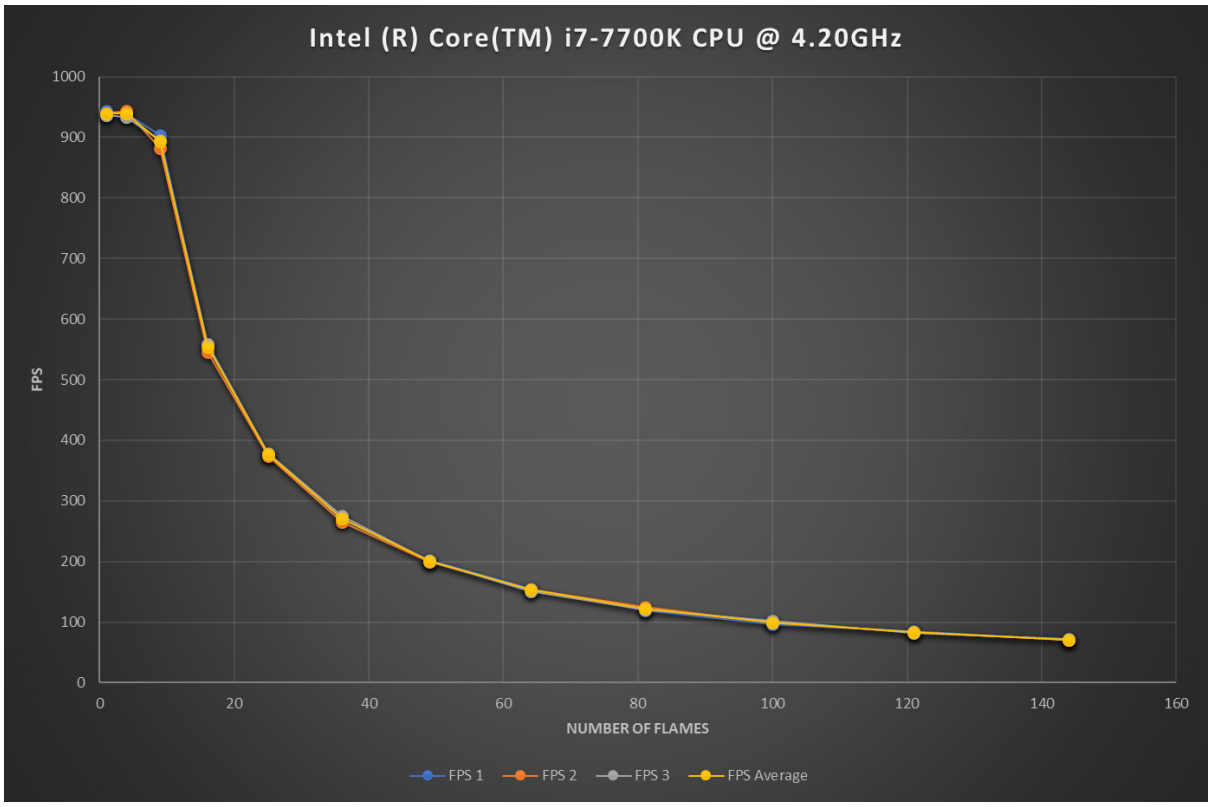

Figure 12: A graph showing the FPS of machine 2

Machine 3 Specifications:
Processor - Intel(R) Core (TM) i7-4700MQ CPU @ 2.40GHz
Memory - 8GiB RAM
Graphics - NVIDIA Corporation GK106M [GeForce GTX 765M]

Number of Flames	FPS 1	FPS 2	FPS 3	FPS Average
1	946	952	939	945.67
4	924	917	933	924.67
9	609	602	591	600.67
16	363	376	379	372.67
25	248	251	241	246.67
36	180	179	181	180
49	135	131	134	133.33
64	104	102	105	103.67
81	83	84	81	82.67
100	68	67	65	66.67
121	57	56	55	56
144	47	48	49	48

Table 3: The FPS of Machine 3

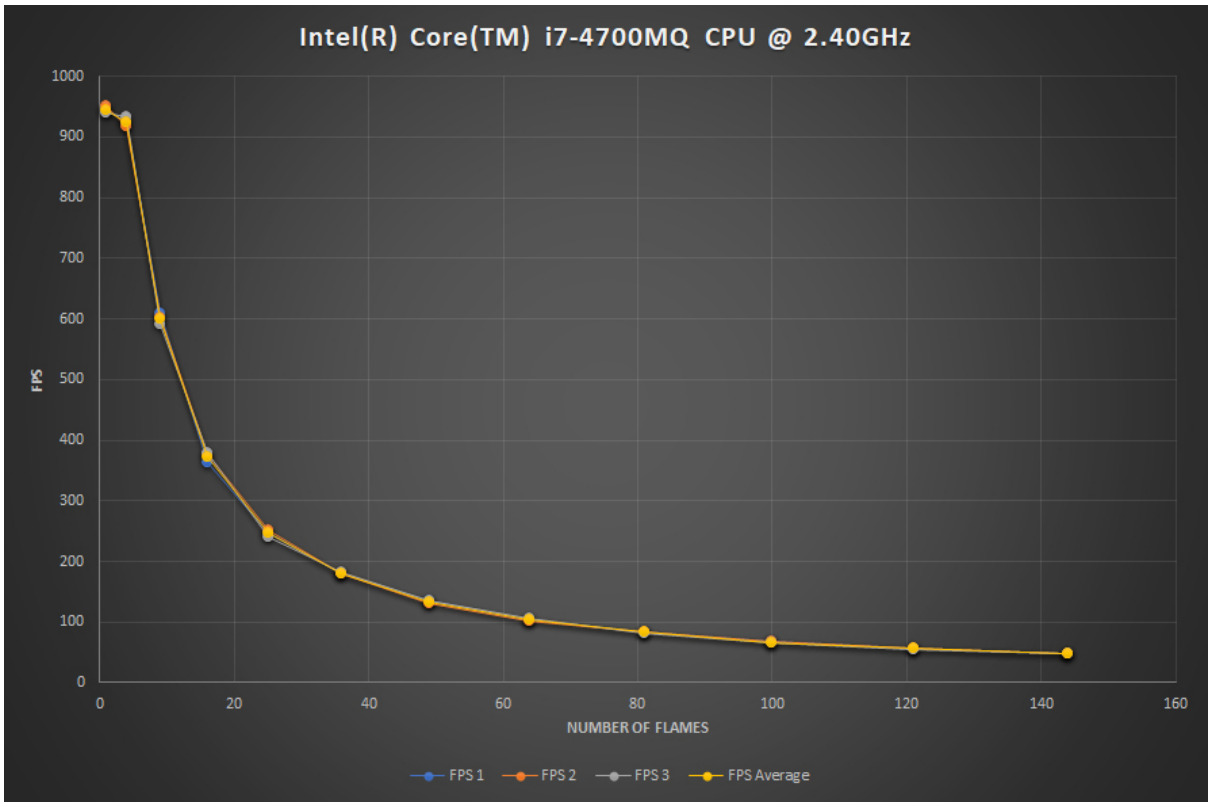

Figure 13: A graph showing the FPS of machine 3

Machine 4 Specifications:
- Intel (R) Core (TM) i3-5010V CPU @ 2.10GHz
- 8GiB RAM
- Intel HD graphics 5500

Number of Flames	FPS 1	FPS 2	FPS 3	FPS Average
1	188	190	193	190.33
4	168	166	163	165.67
9	138	144	141	141
16	114	111	110	111.67
25	87	85	88	86.67
36	76	71	72	73
49	60	61	59	60
64	48	47	49	48
81	39	40	41	40
100	32	33	35	33.33
121	29	28	30	29
144	25	24	26	25

Table 4: The FPS of Machine 4

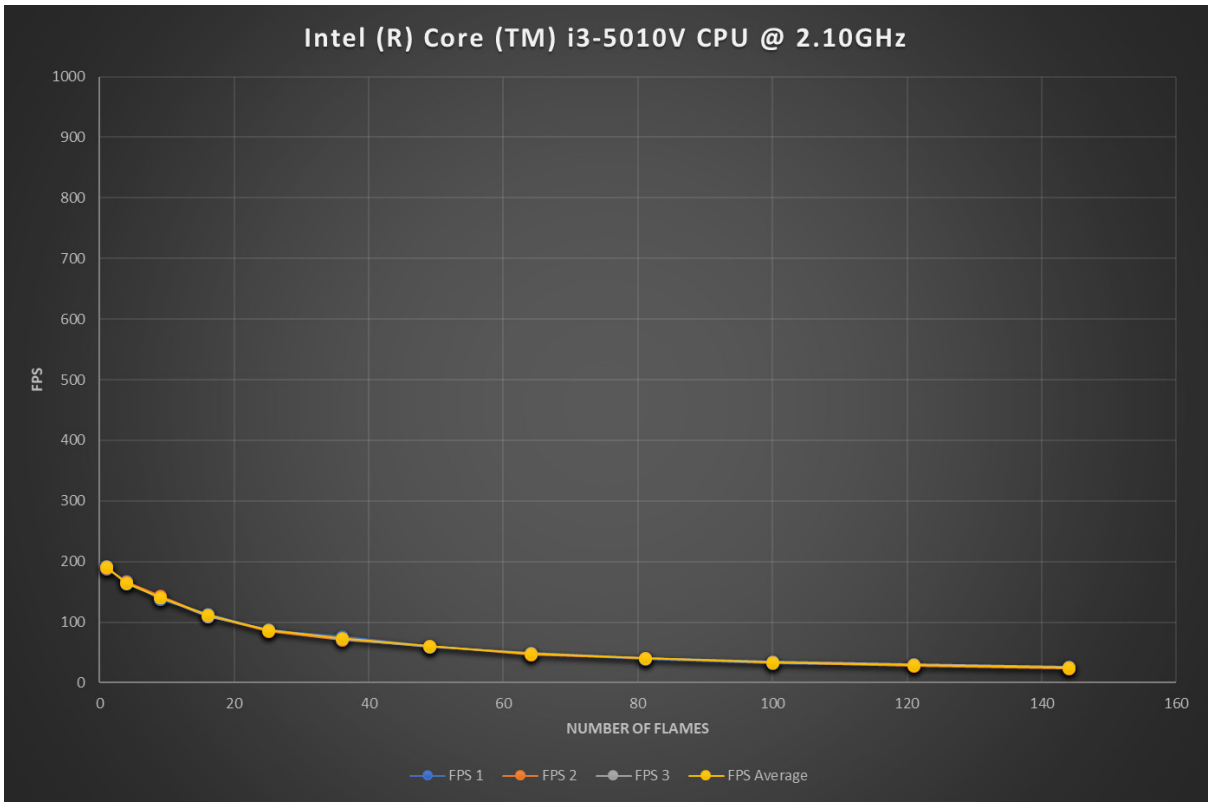

Figure 14: A graph showing the FPS of machine 4

Analysis

After collecting the frame rate data, several observations can be made. One being that the relationship between the number of flames in a scene and the frame rate is not linear.

This means that for every new instance of a mass spring object that is added to the scene the increase in the amount of calculations is not same meaning that the O notation of the algorithm that is being run is not O(n).

Looking at the way that the line in the graphs are shown for all the machines it appears that the relationship between the frame rate and the number of flames has a small curve at first, followed by a dramatic drop that slowly evens back out. This in fact looks like exponential decay which means that it drops dramatically at first but then evens itself out.

As this curve appears to happen on all of the machines it is likely that 144 flames is reaching the limit of the amount of these flames that can be in a scene at a time for use with real time applications as most real time applications run at either 30FPS or 60FPS. At 144 flames all the machines had fallen under 60FPS and the lowest powered machine had fallen bellow 30FPS, however in most situations it would be unlikely that the scene would contain more than 144 flames in it.

However, this application is only testing the movement aspect of the flames, when you add in lighting calculations this could change the results, but for the purpose of the results in this paper this does seem to be an effect method of simulating flames in real time.

From a perspective of being able to technically run as required for implementing flame in real time applications this seems to point towards the mass spring method being a viable one, however there is another factor that has yet to be mentioned and that is the visual one. Even if it was the most efficient implementation possible, if the way that it visually looks is incorrect then it still fails as a flame.

Obviously a large part of it effectiveness will be based on the textures used for the flame and the forces that are chosen to act upon it, however with the wind set to also push in a direction along the x axis it does produce an effect that is quite similar to how a flame would wave in the wind, with the buoyancy of the flame adding to the effect of the flame moving up and down.

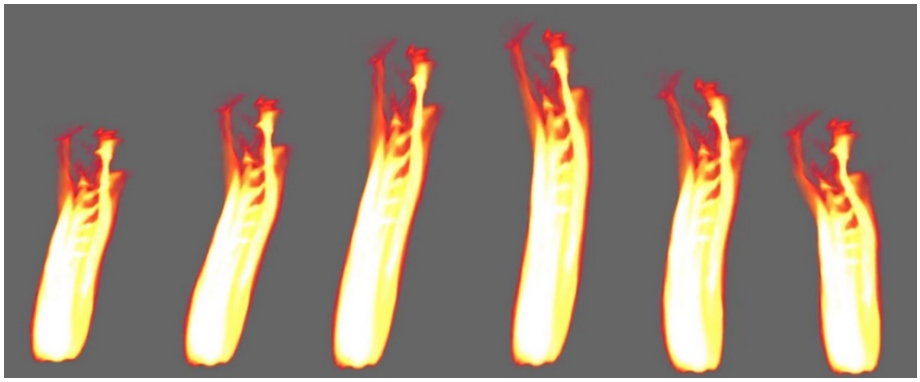

Figure 15: An example of the generated flame movement.

It is not perfect and if you look carefully at the flame you can see it does not look exactly right however it does still give the effect of a moving flame and as long as the flame is intended to be a background feature it would work fine as a flame. It would not work well as a hero feature, however.

Conclusion

There are advantages and disadvantages to this method, the advantages being the interactivity that this method can do in real time. The wind direction can be changed along with multiple other variable to dynamically edit the way that the flames are moving inside the real time application whereas methods such as using an animated texture is a fixed animation that cannot be changed dynamically. However, the downside of this method against methods such as the animated texture is that it does require more processing power.

Full fluid simulation for the fire simulation is not possible with real time graphics now, however it does create more realistic looking fire. Whereas the look of this method of fire simulation does not look realistic when up close however being used as a background flame it would work relatively well.

As shown by the results of this project the mass spring method for fire simulation is reasonably efficient and can be used in real time applications without causing massive frame rate loss.

In conclusion this method has a lot of potential as an alternative method to fire simulation in real time, however for it to be a fully viable option it will need to have some future work done to it. That said this project can be considered a success as it has proven the potential of mass spring systems being used for fire simulation.

Future Work

There is a lot of potential improvements that could be added to this method in the future. As shown in the existing paper on the subject it is possible to create much more realistic textures using variable transparency **[Balci & Foroosh 2006]** and this can be implemented into this codebase to improve its realism.

In the same paper they talk about how they used a more advanced version of the mass spring system that also utilises energy that are based upon heat energy rather than just the mass spring forces on the mass spring object **[Balci & Foroosh 2006]**. This can also be implemented into the code in the future.

If collision detection was added to the mass spring object, then the flames could be interacted with in applications such as games by the player and cause the flame to move out of the way of the player.

Lighting is another function that could be implemented into the codebase in the future to help improve its realism. In idea could be to use a light map to apply to the texture so that when the texture moves and stretches then the light emitted would move with it causing the lighting environment to flick correctly.

Once the extra features are added into the codebase then the codebase could be implemented into another pipeline, for example the flame method could be implemented into Unreal engine for use with real time applications.

References

Ren & Jin & Chen, H.R & Y.J & L.C, 2009. *Realistic rendering of fire scene*. 2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics., Huangshan, China, October 19th-21st 2009. IEEE. Available from: https://ieeexplore.ieee.org/document/5246848 [19th August 2019].

Balci & Foroosh, M.B & H.F, 2006. *Real-time 3D fire simulation using a spring-mass model*. 12th International Multi-Media Modelling Conference, 2006, Beijing, China, Jan 4th-6th 2006. IEEE. Available from: https://ieeexplore.ieee.org/document/1651309 [19th August 2019].

Qian & Liu & Guann, D.Q & X.L & Y.G, 2010. *Research on particle-based simulation for fire*. 2nd International Conference on Future Computer and Communication, 2010, Wuha, China, May 21th-24th 2010. IEEE. Available from: https://ieeexplore.ieee.org/document/5497793 [19th August 2019].

Terzopoulos & Platt & Barr & Fleischer, D.T & J.P & A.B & B.F, 1987. *Elastically deformable models*. SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques, New York, USA, 1987. ACM. Available from: https://dl.acm.org/citation.cfm?id=37427 [19th August 2019].

Provot, X.P, 1995. *Deformation constraints in a mass-spring model to describe rigid cloth behaviour*. Proceedings of Graphics Interface '95, Québec, Canada, 1995. Canadian Human-Computer Communications Society. Available from: http://graphicsinterface.org/proceedings/gi1995/gi1995-17/ [19th August 2019].

Gao & Ma, X.G & X.M, 2010. *Simulation of fire based on improved particle system and texture rendering*. 2010 International Conference on Image Analysis and Signal Processing., Zhejiang, China, October 9th-11th 2010. IEEE. Available from: https://ieeexplore.ieee.org/document/5476049 [19th August 2019].

Millington, I.M, 2010. *Game Physics Engine Development.* Second Edition, Boca Raton: CRC Press

Dobashi, Y.D, 2009. *Simulation of various natural phenomena based on computational fluid dynamics*. 11th IEEE International Conference on Computer-Aided Design and Computer Graphics, 2009, Huangshan, China, Aug 19th-21th 2009. IEEE. Available from: https://ieeexplore.ieee.org/document/5246811 [19th August 2019].

Figures

IntaSound P.A, 2014. New LED Flamelights to hire. IntaSound P.A. Available from: http://www.intasoundpa.co.uk/blog/uncategorized/new-led-flamelights-to-hire/ [19th August 2019].

Lengyel, E., 2012. *Mathematics for 3D Game Programming and Computer Graphics*. Third Edition, Boston: Course Technology

Millington, I., 2010. *Game Physics Engine Development.* Second Edition, Boca Raton: CRC Press