University of Portsmouth

School of Creative Technologies

Final Year Project undertaken in partial fulfilment of the requirements for the BSc (Honours) in Computer Games Technology.

How can the high performance visualization tool Splotch be adapted to produce cosmological images in two distinct computing environments?

By
Elliott George Ayling
614173

Supervisor: Mel Krokos

Project Unit: CT6CTPRO
March 2016

Project Type: Artefact

Abstract

Two client based projects are presented in this report. Both use the ray-tracer software tool Splotch, which supports effective visualization of cosmological simulation data. The aim was to create applications which focus on providing optimised and accessible functions for use in cosmological and astronomical visualization. The first is an application made with the artistically focused C++ library Cinder, and is designed to offer an easy to use tool to interact with 3D visualizations of cosmological datasets on an accessible platform, and is the first of its type within Cinder. The second shows the adaptation and implementation of the Splotch software on the gSTAR supercomputer for the purposes of the Theoretical Astronomical Observatory. The result of which has a focus on optimisation whilst also adding functionality to automate parts of the Splotch software, with the aim of lowering the level of accessibility to produce the visualization images. The culmination is a report which can act as a guide for any projects which aim to pursue similar goals.

1 CONTENTS

2 Introduction 5
2.1 Project Overview 5
3 Literature Review 6
3.1 Introduction 6
3.2 Simulations 6
3.3 Visualization Tools 7
3.4 Splotch 7
3.4.1 Splotch Previewer 9
3.5 Cinder 9
4 Methodology 11
4.1 Methodological Models 11
4.1.1 Waterfall Model 11
4.1.2 Prototype Model 11
4.1.3 Incremental Build Model 12
4.2 Justification of Chosen Model 12
5 Using Splotch within a Cinder application 13
5.1 Entropy 13
5.2 Requirements 13
5.2.1 Gathering of Requirements 13
5.2.2 Functional Requirements 13
5.2.3 Non-functional Requirements 14
5.3 Initial Research 14
5.4 Design 14
5.4.1 Proposed Solutions 14
5.4.2 Decided Solution 16
5.5 Timeline 17
5.6 Implementation 17
5.6.1 Development Tools 17
5.6.2 Integration with Splotch 18
5.6.3 Cinder Interface 19
5.6.4 Cinder Renderer 20
5.6.5 Shader assets 21
5.7 Prototypes 24
5.7.1 First Prototype 24

5.7.2 Second Prototype ... 24
5.7.3 Third Prototype ... 24
5.7.4 Fourth Prototype ... 25
5.7.5 Fifth Prototype ... 26
5.7.6 Sixth Prototype ... 27
5.7.7 Eight Prototype ... 28
5.7.8 Ninth Prototype ... 29
5.8 Testing ... 30
5.8.1 White and Black-Box Testing ... 30
5.8.2 Module (Unit) Testing ... 31
5.8.3 Testing Implementation ... 31
5.9 Evaluation and Future Work ... 32
5.9.1 Summery ... 32
5.9.2 Performance and Optimisation ... 32
5.9.3 Methodology ... 33
5.9.4 Further Features ... 33
5.9.5 Recommendations ... 34
6 Integrating Splotch for use on the Swinburne gSTAR Supercomputer ... 35
6.1 Introduction ... 35
6.2 Initial Planning ... 36
6.3 Adaptation of the HDF5 Reader ... 37
6.4 Further Work ... 38
6.5 Optimisation and Testing ... 40
6.5.1 Benchmarks ... 40
6.5.2 Further Optimisation ... 42
6.6 Evaluation and Future Work ... 43
6.6.1 Summery ... 43
6.6.2 Performance ... 43
6.6.3 Workflow ... 43
6.6.4 Future Work ... 43
7 Conclusions and Evaluation ... 44
7.1 Summery ... 44
8 Table of Figures ... 45
9 Bibliography ... 46
10 Appendix 1 - Client proposal document ... 52
11 Appendix 2- Splotch Manual First Draft ... 53

12 Appendix 3- Test Cases 57
12.1 First Prototype 57
12.2 Second Prototype 58
12.3 Third Prototype 59
12.4 Fourth Prototype 60
12.5 Fifth Prototype 61
12.6 Sixth Prototype 62
12.7 Seventh Prototype 63
12.8 Eight Prototype 64
12.9 Ninth Prototype 66
12.10 Not Tested 68

2 INTRODUCTION

46 years ago, on July 21 1969, the world watched as Neil Armstrong descended from the Apollo spacecraft and became the first human to step foot on a world that was not our own. The image of Neil Armstrong walking the lunar surface, and the words he spoke when he did have become immortalised in time, and have served to inspire countless generations of children, including a future commander of the International Space Station (Wall, 2012). The achievement of the first lunar walk was a huge accomplishment at the time, but has served to accomplish even more through the actions of those it inspired. But, with the final Apollo mission having concluded 44 years ago, the majority of the exciting scientific discoveries and accomplishments since have been performed with pen and paper, and more recently, computers. So how can science, and the people behind it, attempt to emulate the excitement and fever 46 years ago of seeing a man, and his spectacular small step.

Science outreach programs aim to "simulate interest and to encourage better understanding of the application of science…" (Edwards, 2016). They have a long history, a famous example of which is Michael Faraday's Christmas Lecture series started in 1825 (Sample, 2015), which had the aim of introducing "a young audience to a subject through spectacular demonstrations…" (Royal Institution, n.d). In a time where technology plays such a key role in both personal, and scientific areas, science outreach programs must, too, evolve and adapt to the ever changing landscape.

2.1 PROJECT OVERVIEW

This report will speak about two separate client based projects. The first is a project for The University of Portsmouth's Institute of Cosmology and Gravitation's Katarina Markovič. This project is for the purposes of the art-science collaboration performance, Entropy. Entropy is a public talk about the history of the universe, accompanied by a live audio-visual performance. The primary objective of the project is to create an application which can support cosmological visualizations in a format that is easy and accessible for the public, in the same vein as the goals of Entropy. This takes the form of an application using the C++ library Cinder, which is "for programming with aesthetic intent" (Cinder, 2015a). The application created for this artistic environment will be described in detail in section 5.

The second project is also a client based project, but for Claudio Gheller, the Scientific Community Engagement Group Lead at the Swiss National Supercomputing Centre, and is one of the developers for the focused software in this report, Splotch. Through Claudio the project will involve adapting Splotch for the purposes of the Theoretical Astrophysical Observatory(TAO) on the gSTAR supercomputing system operated by the Centre for Astrophysics and Supercomputing, based out of the Swinburne University of Technology in Melbourne, Australia. The work performed for this project will be discussed in section 6.

Both projects present a unique opportunity to develop applications that focus on easily visualizing cosmological datasets but in two very different environments. The first is for the artistic and atheistically focused Cinder community- who are not necessarily knowledgeable of the technical skills needed to work within a high performance computing environment, and the second is for the astronomical scientific community which TAO serves through a focus on optimised and effective supercomputing systems.

3 LITERATURE REVIEW

3.1 INTRODUCTION

One of the more prevalent questions that modern science aims to understand is how the formation of structures within the universe were created some 13.82 billion years ago following the *Big Bang* (Europeon Space Agency, 2013) in order to gain an understanding of the universe as whole. A huge variety of factors came into play at the creation of the universe which led to the increasingly complex structures that we have today. The results of this has led to adoption of techniques and tools which are designed in a way that attempts to fully describe and explain such a fundamentally complicated system of stars, gases, and galaxies. The data which these techniques produce can range from megabytes in size (Goldbaum, 2011), to petabytes and more (Lerner, 2015), and so a need is created in order to explore this data in a visual and intuitive way.

Visualization is a way to look at these huge data sets in a way that allows us to understand the data in a communicative way. Kosara (2007) defines data visualization as being:

1. **Based on (non-visual) data**- the data must come from outside the program and the program must be able to work on different datasets.
2. **Able to produce an image**- The goal of the visualization must be producing one or more images as its means of primary communication of the data. The visualization must be able to stand on its own.
3. **Able to produce a readable and recognisable result**- The result must be understood by the viewer, even if this requires training or practice. The use of additional elements is possible, but must not take precedence over communication goals of the visualization.

Therefore, a visualization should create images that are both intriguing and informative whilst also serving the scientific purpose of allowing us to represent numerical values in a meaningful system.

3.2 SIMULATIONS

In order to visualize this data, they must first be produced, and in order to realistically do so, there exists a number of methods which produce numerical simulations used to simulate these various structures of the universe (Dolag, Borgani, Schindler, Diaferio, & Bykov, 2008).

One such example of these simulations is the Millennium Simulation (Springel, et al., 2005), which uses a modified version of the *GADGET2* code (Springel, Yoshida, & White, 2001) to simulate $2{,}160^3$ particles. The Millennium Simulation used 512 processes of a parallel computer at the Computing Centre of the Max-Planch Society, almost 1TB of memory, and required around 350,000 core hours, or 28 days of wall time (Springel, et al., 2005). This was considered the best large cosmological simulation for a number of years, along with the Millennium-II simulation (Boylan-Kolchin, Springel, White, Jenkins, & Lemson, 2009) but has now been to shown to have used obsolete parameters which are considered to be inaccurate (Stephens, 2011). The higher resolution and more accurate Bolshoi Simulation was introduced in 2011 (Klypin, Trujillo-Gomez, & Primack), which boasted “nearly an order of magnitude better mass and force resolution than the Millennium Run”.

Further simulations include those introduced by the Illustric Project (Vogelsberger, et al., 2014) which produced 230TB of cumulative data volumes. The largest of these simulations took 19 million core hours to produce (The Illustris Collaboration, 2015).

More recent visualizations include one of the largest named the Q Continuum simulation (Heitmann, et al., 2015), which is a simulation carried out on a GPU-accelerated supercomputer, and involves more than half a trillion particles. The raw output of the Q Continuum simulation is approximately 2PB (petabytes), a factor of 100 increase compared to the Millennium simulation (Heitmann, et al., 2015, p. 5) and it used almost 90% of the 18,688 computer nodes of the Titan supercomputer that was used for the simulation.

Further simulations include The v^2GC simulations (Ishiyama, et al., 2015)- the largest of which contains 550 billion dark matter particles taking 11 million CPU hours and using 50TB of memory (Ishiyama, et al., 2015, p. 3), and the Bolshoi-Planch and MultiDark-Planck simulations (Klypin, Yepes, Gottlober, Prada, & Hess, 2016).

One of the attributes that all of these simulations have in common is that they exhibit very large datasets and have all required extreme computational efforts for them to come to fruition.

3.3 Visualization Tools

In order to correctly visualize data there exists a number of current tools available. These tools must be able to process massive volumes of data with accurately within a reasonable time. Due to the size of such data, visualization tools need to be able to be run on computers with intense graphical and computing power.

A number of open source software tools have attempted to take these large simulations and create visualizations. One of these tools is ParaView (Song, Zheng, & Shen, 2006). ParaView is an application built upon VTK (Schroeder, Martin, & Lorensen, 1996) which supports multiple platforms for its visualizations, which can be performed interactively in 3D. It has support for parallel processing (Kitware, n.d) and has been used to visualize cosmological simulations (Woodring, et al., 2011). ParaView can be ran from a desktop computer, as well as within a HPC (high performance computing) environment, however it is aimed towards the scientific community and has a level of access which accommodates this.

The same can be said for similar tools, VisIt (Lawrence Livermore National Laboratory, n.d) and VisIVO (Becciani, et al., 2010), both of which are also based on VTK. VisIVO contains a web interface which allows users to "upload and manage their datasets" (Becciani, et al., 2010, p. 18) however, this is specified as for use to "the scientific community". It has also been released as a science gateway which allows standard users the ability to upload and manage their datasets whilst hiding the underlying technical aspects (Sciacca, et al., 2013).

3.4 Splotch

The two project described in this report concentrate on Splotch, which is a "public ray-tracing software…specifically designed to render in a fast and effective way the different families of point-like data" (Dolag, Reinecke, Gheller, & Imboden, 2008). It is a high performance algorithm used for visualizing large particle-based simulations written entirely in C++. Splotch takes time outputs from a dataset which can be in the tens of terabytes(TB) to produce accurate images of the particles showing their position and velocities, as well as visualizing density, smoothing length, and other parameters. Splotch is optimized to run on standard HPC architectures using MPI based approach (Jin, et al., 2010) with OpenMP. It has also been modified to work with a CUDA programming paradigm (Rivi, Dykes, Krokos, & Dolag, 2014) to exploit modern HPC populated with GPUs.

Figure 3.4-a Sample renderings in Splotch of small (left), medium (middle) and large (right) data sets. From (Jin, et al., 2010).

The main interface to use Splotch is via its parameter files. These are text files which define parameters for the software, like the dataset location on the system, brightness values, particle type, and camera position. The Splotch software is self-contained with no dependencies other than those needed for parallelism, CUDA, and for specific file formats such as HDF5. This makes it highly portable.

Splotch uses a volume ray casting approach which calculates individual contributions of particles to the final rendered image by using the radiative transfer equation (Shu, 1991). It supports this with a parallel implementation which distributes parts of the particles to separate processors, each producing a partial rendering, which is then composed into the final image. The CUDA implementation of rendering shows large gains in performance over the sequential processing version (Rivi, Dykes, Krokos, & Dolag, 2014, p. 17).

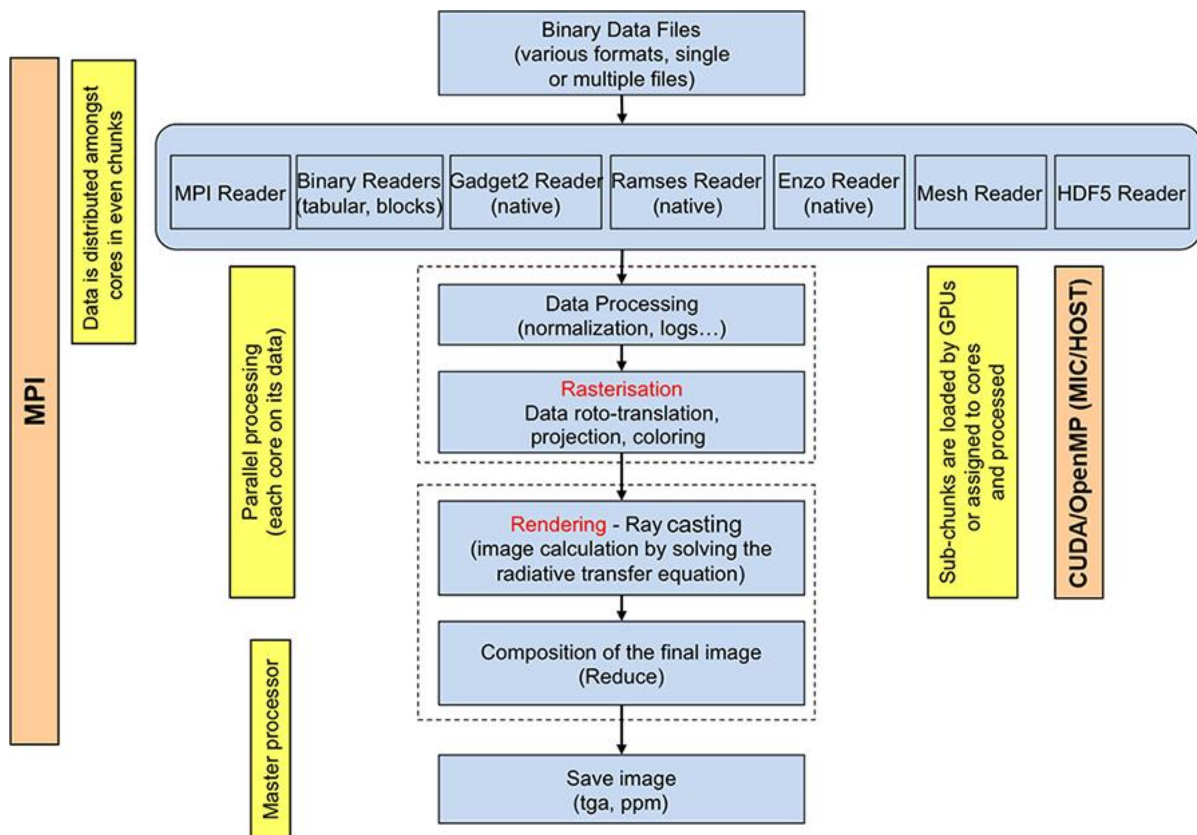


Figure 3.4-b Execution model of the Splotch code. From (Dykes, 2014).

3.4.1 Splotch Previewer

The Previewer is an additional module for Splotch which can be used by compiling the Splotch software with the previewer option in its makefile, and then using the '`-pv`' command when running Splotch.

The Previewer offers an interactive 3D visualization of a dataset within an OpenGL renderer. A user can control the camera using the keyboard and mouse as well as edit parameters via an on screen command line. The Previewer can be used anywhere Splotch can on provision that the relevant libraries are installed. It also can be used to create animations and write to the parameter file.

Whilst the Previewer offers an extra level of interactivity that Splotch does not, it still suffers from a high level of accessibility for anybody not knowledgeable with Splotch or the environments that Splotch takes advantage of. There is seemingly a lack of a tool which could be utilised by a non-scientific user to visualize and interact with datasets.

3.5 CINDER

Cinder is a C++ library with official support for OS X, Windows, iOS, and WinRT (Cinder, 2015a). It offers support for OpenGL and DirectX. A main function of Cinder is to make coding as simple as possible with its inclusion of functions that simplify aspects of coding such as primitive drawing (e.g., `drawCube`).

Cinder refers to itself as being "for programming with aesthetic intent- the sort of development often called creative coding" (Cinder, 2015a). According to Rijnieks (2013, pp. 7-8) creative coding is "a field that combines coding and design".

Processing (Processing Foundation, n.da) is another popular creative coding tool which also supports various operating systems and OpenGL. However, whereas Cinder is a library that uses C++, Processing includes its own IDE with its own language (also named Processing) which builds from the Java language (Reas, 2007). As both Cinder projects (Cinder, 2015b) and Processing projects (Processing Foundation, n.db) can create very similar results, Cinder seems like the preferred choice if a user is familiar with C, C# or C++, and Processing the preferred choice if a user is familiar with the Java language. For the purposes of this project, the decision to use Cinder was a client choice, and from a technical point of view, as both Splotch and Cinder share a common language, Cinder is better suited.

As both Cinder and Processing share the common function of creative coding, implementing Splotch within a Processing application would most likely be able to produce similar results as are described in this report. However, as Processing uses its own Java like language, the entirely of the Splotch code would have to be rewritten to support this language. Therefore, Cinder, again, is the preferred library.

Cinder has been used in an abundance of public projects. One example is a multimedia wall in the Deutsche Bank, in Hong Kong (Akten, Bereza, Buni, McNamee, & Dörfelt), which creates patterns and images that are generated in real time. Another is an installation of an interactive LED floor used to showcase the car manufacture Audi's A2 concept car (kollision, 2011). Cinder has also been used for public outreach projects with the aim of educating a user, such as this installation in New Zealand which simulates an ocean feeding frenzy (Hodgin, Boil Up: Realtime Feeding Frenzy, 2013), and an installation in the Canadian Museum for Human Rights (Upswell, n.d) which created an interactive exhibition aiming to convey a sensitive subject matter in an informative way. Cinder also has been used to create particle systems such as in the Aether project (Lengeling & Castro, 2014).

The strength of Cinder comes from its strength of providing information through its aesthetically interesting interfaces, along with its ease of use from a developer point of view. Combining this in a way that take advantage of the Splotch software's ability to process point like data would produce an application which has a lower level of accessibility than using Splotch directly. As the Windows operating system has the majority market, and Mac OS and Linux share less than a quarter (Net Applications, 2016), producing this application on a Windows system would allow it to reach the largest portion of users.

4 METHODOLOGY

This section will discuss the various methodological approaches available, including their advantages and disadvantages, and discuss the chosen model with justification for choosing it.

4.1 METHODOLOGICAL MODELS

4.1.1 Waterfall Model

The waterfall model is a simple methodology where the development 'flows' downwards towards the final release in a sequential fashion- only progressing onto the next step when the current step is fully complete. The waterfall model is advantageous when the development cycle is known and understood, as once one has moved onto the next step, the previous steps are not revisited (Weinstein & Jaques, 2010). This means that the design is not flexible or able to be modified. It is a very strict model and does not account for uncertainties within the software development process. The result of its rigidity means it is also not well suited for client feedback during the feedback.

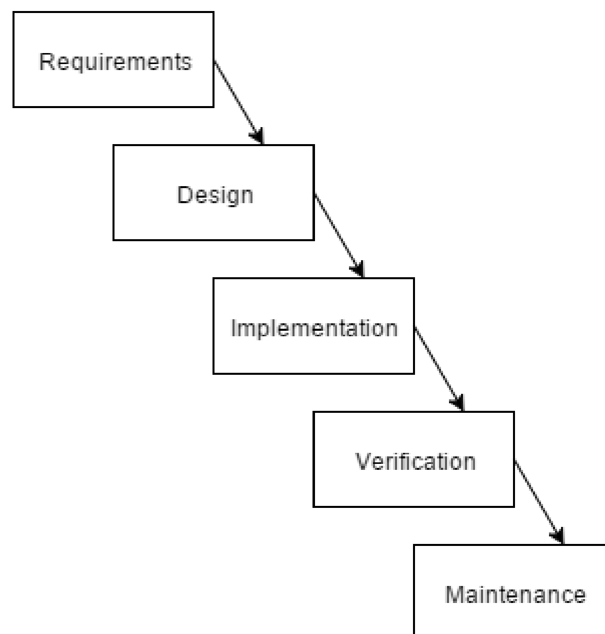


Figure 4.1.1-a Waterfall model flowchart.

A potential supporting argument for the waterfall method is that more time will have to be spent in the planning stages of the project, which could potentially identify problem areas, or help when planning the timeline of the project.

4.1.2 Prototype Model

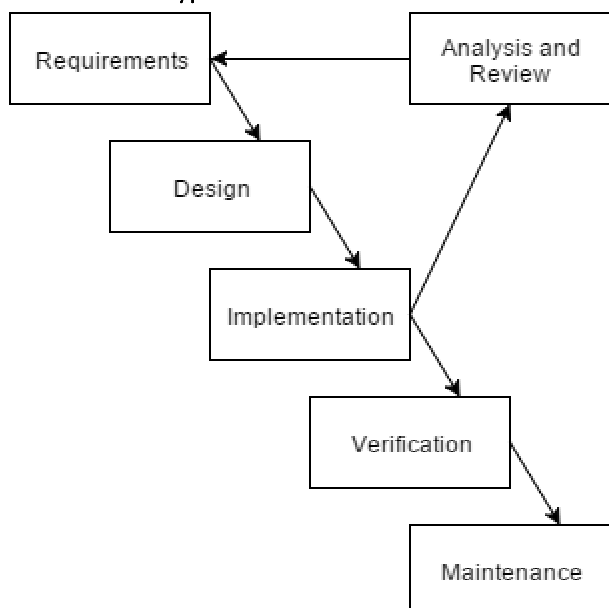


Figure 4.1.2-a Prototype model flowchart.

A prototype model takes the basic structure of a waterfall model, but allows the development cycle to effectively go through that structure multiple times over the course of development (Bersoff & Davis, 1992). A prototype is designed and planned, and then produced. This prototype is then tested and shown to the client for feedback, which allows it to be built upon, or a new prototype created. The advantage of a prototyping model is the flexibility it gives the developer to revise the design and implementation following potential overlooked or unknown problems that may come up during development. It also allows the client to be more involved in the process of development, and hopefully produces a final

deliverable which is more in line with their requirements. This relationship with the client can reduce the risk of wasted time as prototypes are developed over weeks or months, as opposed to the waterfall model which is a fixed timeline from design to deliverable.

There are various variations to the prototype model, such as throwaway prototyping and incremental prototyping, however the most appropriate for our project is the evolutionary prototyping model which is the process of developing an initial prototype which is then refined following feedback and tests as the project goes on. This results in a very flexible model.

4.1.3 Incremental Build Model

The incremental build model "combines elements of the waterfall model applied in an interactive fashion" (Pressman, 2005, p. 48). The first increment of this model produces a core product which fulfils the basic requirements of the software, and further increments contain the supplementary features. This increment is then reviewed and tested, by the developer and client, which allows a plan to be created for the next increment which details further features and functionality.

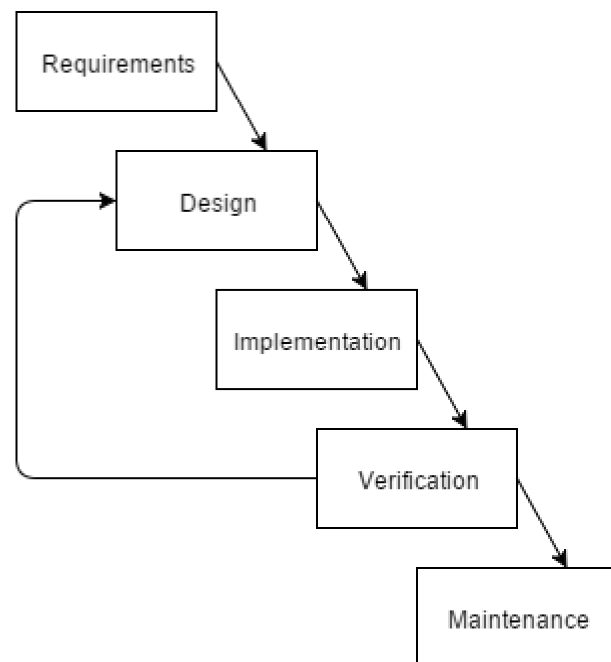


Figure 4.1.3-a Incremental build model flowchart.

Unlike the prototype model, which focuses on implementing and testing single features for each prototype the incremental build model focuses on creating an operational but 'stripped-down' version of the product initially and then building upon this in smaller increments. It has similar advantages to the prototype model, but it allows for a product to have a clearly planned and linear path of development to the initial increment, whilst allowing the flexibility of further features to be reviewed and discussed.

4.2 Justification of Chosen Model

Due to the uncertain nature of the validity and potential unseen setbacks of the project, it was decided to use the incremental build model. The main reason for choosing this model is that the main bulk of the work involved in the project will be the initial increment of developing the Splotch software to work within a Cinder environment. This can be considered the 'core' feature of the project and an abundance of planning, understanding, and development will be needed for this initial increment; which lends itself to the linear approach that the first increment allows. Following this, the software will need to be assessed and reviewed to evaluate performance and feasibility of the software as a whole. After this assessment, subsequent features can be discussed and implemented with the knowledge of how the programs core functionality is performing. An example situation which lends itself to this model could be that, following completion of the core functionality, the performance of the software is subpar, this would allow the next increment to focus on performance following a review of the software. This is a problem which could not be accurately predicated until the program has had this initial increment, which, when using this model, allows the development more flexibility to react to potential problems.

5 USING SPLOTCH WITHIN A CINDER APPLICATION

This chapter will detail the process of creating a Cinder application that utilises Splotch to create a visualization tool.

5.1 ENTROPY

'Entropy' is an "art- science collaboration" that combines public talks on astronomy with a live audio-visual performance (Markovic, 2016). The show is being aimed to be shown at various science, music, and art festivals around the UK and internationally. The show will be visualizing astronomical data from various real world experiments, as well as computational simulations supernova explosions. These visualizations will be accompanied by educational talks and specially produced musical tracks. The expected audience is expected to be several thousand people within the first year of the live performances (Markovic, 2016).

5.2 REQUIREMENTS

5.2.1 Gathering of Requirements

Requirements for this project were gathered from one to one meetings with the client from the Entropy project. Meetings with the client were initially relatively regular in order to get a general idea of the project, already existing materials, and outline of requirements and work. As there were multiple chapters to the Entropy project, there were many options for how the work could proceed. Following these meetings, it was decided that the work on these project should focus on how to integrate loading large astronomical datasets within a Cinder application. These meetings allowed the project requirements to be defined precisely.

5.2.2 Functional Requirements

1. **Visualize astronomical datasets within a Cinder application-**
   Existing work for the Entropy project has been built using the Cinder library. As such the client required the Splotch integration to also be within Cinder. The application must be able to use Cinder's libraries whilst accuracy visualizing datasets.

2. **Ability to create images that are alike to Splotch produce images but within a non-HPC environment-**
   The Splotch software is written with a HPC environment in mind, but the Entropy project is aiming to be an educational public project. Therefore, it is important the software has a lower level of accessibility for a user. This meant that the software should be able to be ran on a variety of systems, not just Unix environments, for any person who wished to do so.

3. **Read and edit Splotch parameter files-**
   Having the ability for the program to edit the parameter file within the application streamlines the experience for the user, as they will be able to edit and load as they require, without having to exit the program or use an external editing tool. This also supports the ability for the user to move onto using the full Splotch software, as they can use the same parameter file they are editing here.

4. **Real time interactivity-**
   The user should be able to have some kind of control over the visualization once it has been loaded. This will at least include camera controls, but also the user should have some form of control over the aesthetics of the visualization in real time. Controls will also aim to be as intuitive and effortless as possible.

5.2.3 Non-functional Requirements

The software will act as an extension of the existing Splotch software, using its source files as necessary, in order to create as close an image as possible within Cinder. The accuracy of this image will be determined by its similarity to the image that the Splotch software would produce of the same dataset. It will aim to be as performance conscious as possible, in order to minimise the computational overhead so as to allow for as large a dataset to be visualized, dependant on a user's hardware.

The application will, as much as possible, be as accessible to non-scientific and non-academic users so as to promote the public outreach aspect of the Entropy project.

The final application will be released as an open-source program.

5.3 Initial Research

Following and during the gathering of the requirements, it was important to fully understand the Splotch software so as to understand how best to approach integrating it into a Cinder application. As there is no precedent for astronomical visualization software within Cinder, and very little visualization tools for the Windows platform at all (most use a Linux emulation), there was a large period of research at the start of the project. At the time of the project, the documentation for the newest version of Cinder (0.9.0) was not substantial and the majority of the learning was performed by studying a subset of samples provided to observe how the library worked.

However, the largest portion of time within this period was spent studying the Splotch software. This required the installation of a Linux distribution, as well as the sourcing of datasets to test the software with. The documentation for Splotch was almost non-existent which forced the studying of the software to rely largely on testing, and studying of the code. A substantial amount of help via conversations with developers who have worked on the Splotch software allowed a better understanding of its use. Example files were also provided from these developers. At the start of development, developer experience with the Linux operating system was sparse, and so time was also spent on becoming familiar with the use of the command line.

5.4 Design

Due to the nature of the application, a large amount of the inner workings of the application would only be discovered via exploration and testing of the software. It therefore seemed unrealistic to be able to create an in-depth design document of the code structure, and instead a general overview of how the software should performed was to be generated.

5.4.1 Proposed Solutions

Following the initial planning stages, a document was produced to propose possible solutions to the task presented (see appendices). There were two solutions proposed.

5.4.1.1 Solution 1- Remotely connect to a Splotch machine, via a Cinder application.

This solution would create a Cinder application which would act as an interface in which the user could load and edit a parameter file. The Cinder application would act as the front end, providing the GUI (Graphical user interface) to the user. The application would then send the resulting parameters to the Splotch software, which would have to ran on a Linux machine, to render the image according to the set parameters. Splotch would then send the data back to the Cinder application, which would either, output the image, or use the resulting data to create a 3D visualization.

This solution would allow a user to produce the exact same image as the Splotch software would, except from a Windows or OSX system, whilst also being easy for them to use.

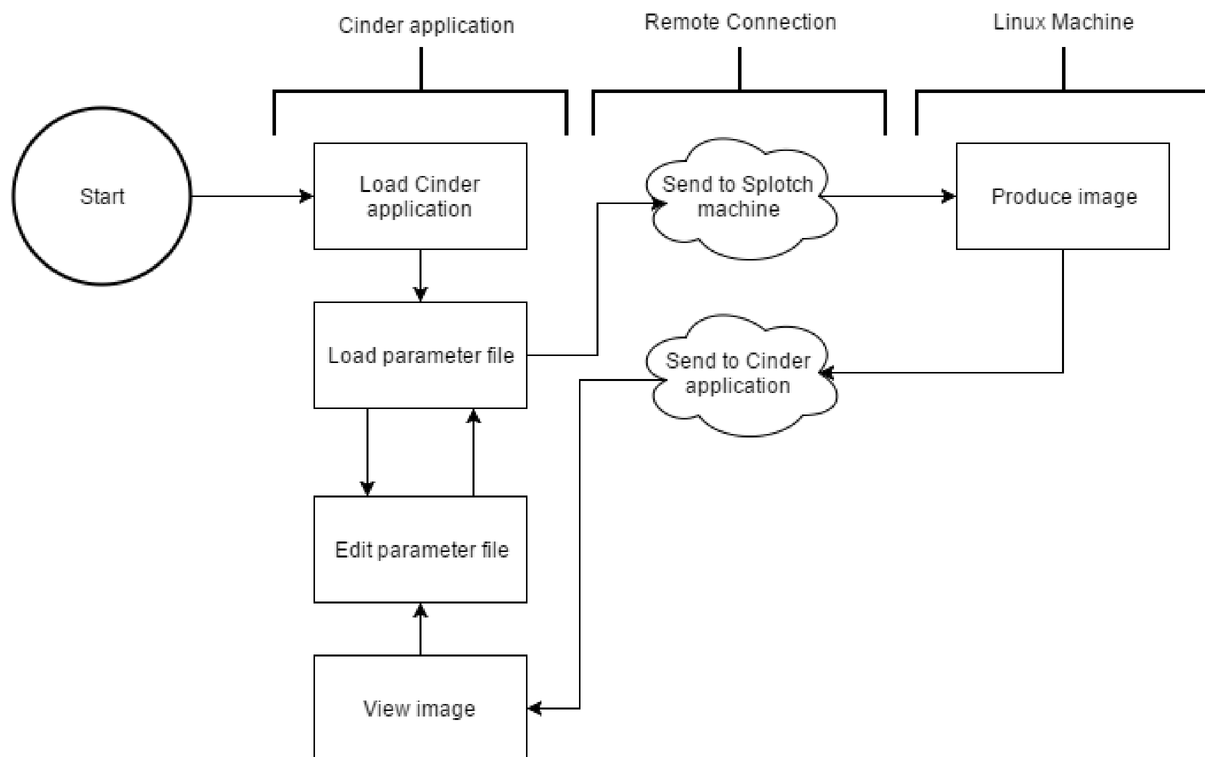


Figure 5.4.1-a Application flow of remote connection solution.

The drawback to such a system would be that the user will need to have access to a HPC or generic Linux environment. As the project has an emphasises on being as accessible as it can, and as most data seems to place Linux market share as being below 2% of users (Protalinski, 2015) (Net Applications, 2016) this does not support this ideal. Also due to the possibility of having extremely large datasets as discuss in section 3.2, the data would have to also be stored in the Linux system to avoid the large amount of time it would take to transfer these to the Splotch software. This could potentially still be useful for the scientific community, but it also seems to add an unnecessary step when the user, which would need access to the Linux machine with both Splotch and the data stored on it regardless, would choose to use the Cinder application on a separate machine, instead of running the software directly.

5.4.1.2 Solution 2- Creating a Splotch-like Plugin for Cinder.

A second proposed solution was to use Splotch for the purpose of its readers and other functions to process the data into a form that a Cinder application could then access and use to create a 3D visualization on the same machine. The application would function in a similar way to the already existing Previewer for Splotch, but would be able to run on a Windows machine whilst also providing a GUI for the user. The image would be calculated via Splotch, and rendered via the Cinder

application, which would also be providing the interactivity and offer the opportunity to provide additional information for the purposes of the Entropy project.

The Splotch software would be compiled within the program and be running on the users' machine without the need for them to have access to a Linux environment. This would allow the application to produce an image that replicates a Splotch image within Cinder. As the application would be using the same parameter files that the Splotch software uses, this would also offer the opportunity for a user to take that same file and run it via the complete Splotch program within a Unix environment. This solution would satisfy all functional requirements with the only drawback being that the images produced would not be the same images that Splotch itself would produce, but rather as close as can be achieved on the user's machine.

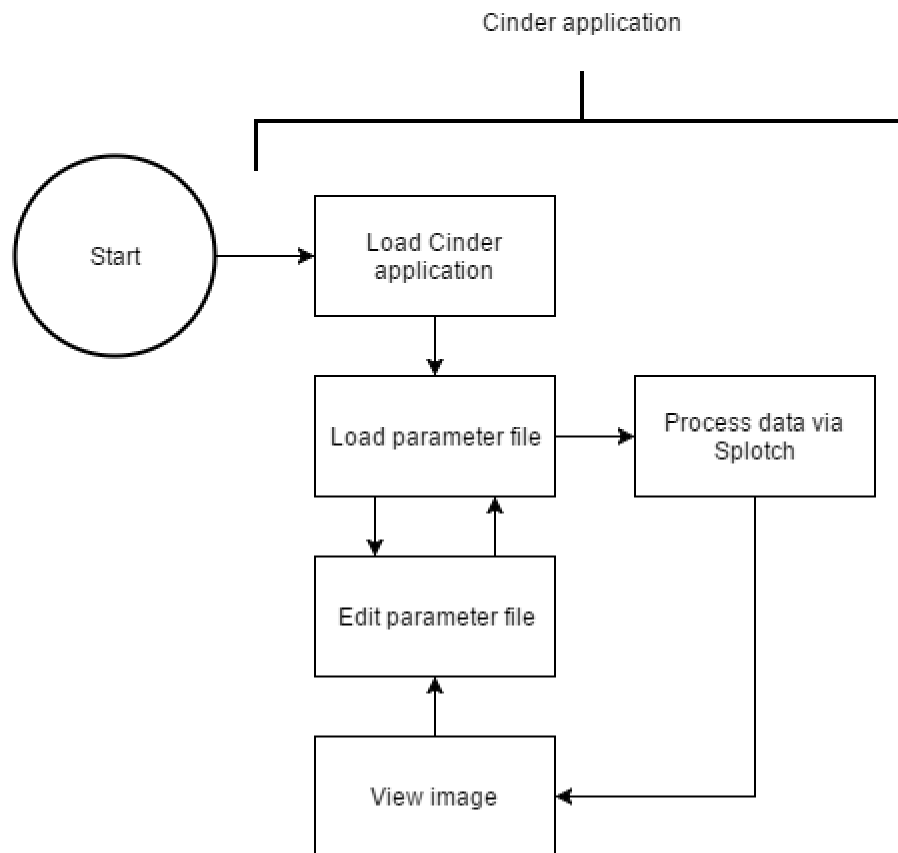


Figure 5.4.1-b Application flow of solution.

5.4.2 Decided Solution

Following positive discussion with the client regarding the solutions, solution 2 was deemed to be most suitable for the purposes of the Entropy project.

5.5 TIMELINE

A timeline for the project was designed at this point in time, which can be seen below. Modifications were made to the timeline during the project due to incorrect estimations. These modifications are discussed at their relevant time in section 5.7. Further modifications were also made to the timeline due to the TAO project (discussed in section 6).

Task Name	Duration	Start	Finish
Research	**48 days**	**Thu 08/10/15**	**Sun 13/12/15**
Initial meetings and outline of project	13 days	Thu 08/10/15	Mon 26/10/15
Understanding Cinder	11 days	Tue 27/10/15	Tue 10/11/15
Understanding Splotch	24 days	Mon 09/11/15	Thu 10/12/15
Deliverable 1 Preparation	2 days	Tue 01/12/15	Wed 02/12/15
Presentation	0 days	Thu 03/12/15	Thu 03/12/15
Implementation	**46 days**	**Mon 14/12/15**	**Sun 14/02/16**
Planning Implementation	7 days	Thu 03/12/15	Fri 11/12/15
Coding	40 days	Mon 14/12/15	Fri 05/02/16
Testing	46 days	Mon 14/12/15	Sun 14/02/16
Write Up	27 days	Mon 15/02/16	Tue 22/03/16
Deliverable 2 deadline	0 days	Thu 24/03/16	Thu 24/03/16

Table 1 The original timeline for the project.

5.6 IMPLEMENTATION

The implementation of the application took the form of creating incremental prototypes, where the initial prototypes were the bulk of the work, and each sequential prototype added features onto the application in smaller increments. Each prototype would then be tested thoroughly before moving on. The focus was on getting the application to have its core mechanics working before moving onto code optimisation. Additional features could then be added at a later date.

This section will describe an overview of how the application is working, and then provide a look at its prototype iterations.

5.6.1 Development Tools

As briefly discussed under in section 5.3, the latest version of Cinder (0.9.0) did not have a substantial amount of documentation available, with the majority of information available being for the previous version (0.8.6). This meant that there would be a longer period of learning for the newest version of Cinder, as a more thorough studying of the provided samples would be necessary. However, as the 0.9.0 version of Cinder has a completely rewritten OpenGL API (Cider, 2015c) it was decided that the application will use the most up-to-date version available in order to be as compatible as possible moving forward.

The IDE (integrated development environment) that was used was largely decided by its compatibility with Cinder. Both Microsoft Visual Studio 2013 and Microsoft Visual Studio 2015 are fully supported by Cinder 0.9.0, however only the former is supported using Cinder's Tinderbox application- an application which can correctly set up a Visual Studio project with Cinder for the user. It was therefore decided to use Microsoft Visual Studio 2013 as an IDE, as the project can always be easily ported to the 2015 at a later date if necessary using Visual Studios in built tools.

5.6.2 Integration with Splotch

The main resource for the application was the Splotch software. This meant that the application will attempt to use as much as the original source code as possible in order to reduce its physical size on disk and also reduce the need for code to rewritten. It also serves the dual purpose of processing the data in the exact way that the Splotch software would. The main reference for how this application would integrate the Splotch software was the already existing Previewer application within the Splotch source files. The Previewer code was used as a basis for an interface between the Cinder and Splotch code.

It was decided that the Cinder application would have its own edited version of the Previewer source files instead of accessing the original files as it does with the core Splotch functions. The reasoning for this was so the Splotch Previewer, and the Cinder application could function as two branches out from the core Splotch without interacting with one another. It would allow each program to be edited and produced separately. As they are both intended for different platforms, this would eliminate any potential conflicts as well as allowing each to play to the strengths of its respective platform.

The main changes to the Previewer files were:

- **Removal of redundant code**- the rendering and user interactivity for the application will be done via functions provided by Cinder. A large amount of the functionality in the Splotch Previewer was therefore unnecessary and removed. This also increasing code readability.

- **Porting of code for use on Windows-** this included using the appropriate equivalent Windows libraries. It also included changes in the way the application reads the file path due to the difference in file systems between the two operating systems. E.g., Windows using backslashes in place of forward slashes.

- **Editing functions to work within the Cinder application-** certain functions had to be edited in a way that the Cinder application could use them in a logical and efficient way.

The core Splotch code also had to be edited minimally in order to be compatible with the Windows operating system. These edits were done under a “`WINDOWSCINDER`” macro definition so as to interfere as little as possible with the original code. The main edits made to the core Splotch code were made in order to allow it to compile on the Windows system and include changes to included libraries, and an added function. The Cinder application does not have its own copies of the Splotch code, and instead its functions are called from within the edited Previewer files.

5.6.3 Cinder Interface

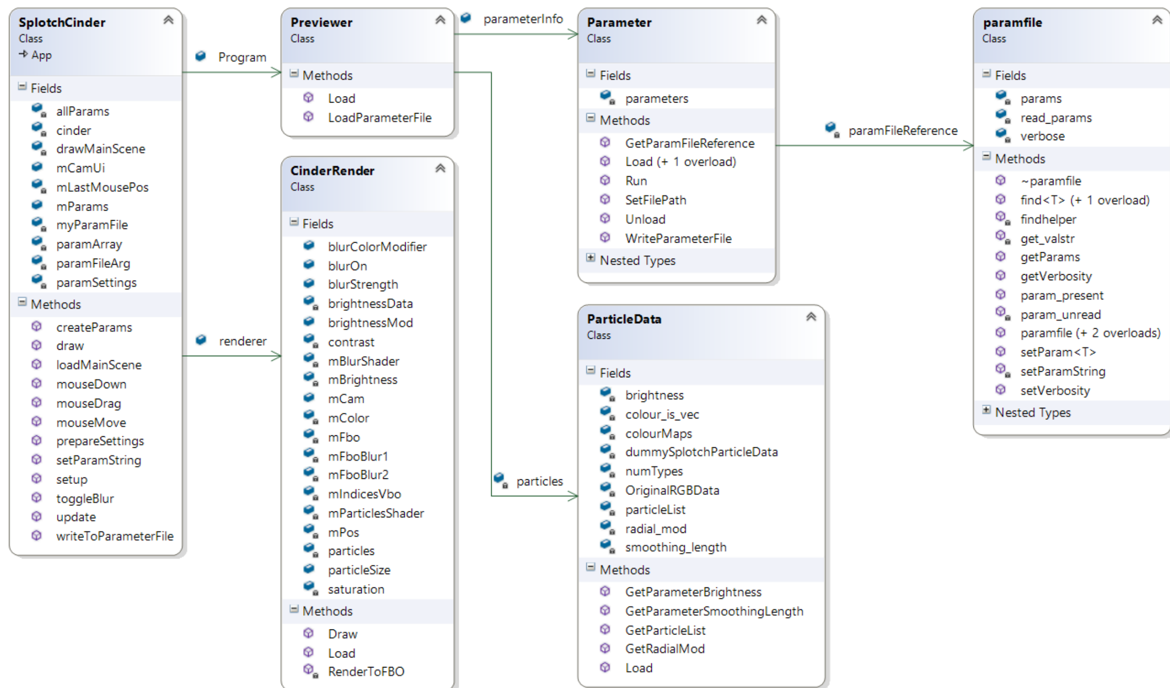


Figure 5.6.2-a Application class diagram.

The `SplotchCinder` class holds an instance of the `Previewer` class and an instance of the `CinderRender` class as can be seen in the above class diagram. This is the top level class responsible for managing the entire application. The main loop of the application is handled by the '`CINDER_APP`' macro which is provided by the Cinder library.

When the application is loaded, the user will be presented with a Windows explorer window which will allow the user to find and select the parameter file that they wish to use for the visualization. The explorer window will only allow the user to select a file with the '`.par`' extension. This acts as a failsafe for the system in the case a user tries to select something other than a parameter file.

The application will then pass the path to the chosen parameter file into the Previewer's '`LoadParameterFile`' function which will use the functions within Splotch to process the file and return a reference to the class which the parameter information is stored in. The creation of the GUI is then performed using the fields within the parameter file. This process is performed dynamically using a combination of Splotch and Cinder functions, and a dynamic multidimensional array. Writing the function this way allows the application to be able to load any number of parameter file variables and their fields. This is important in order for the application to keep up to date with any updates to the core Splotch functionality which may require new fields within the parameter file.

Once the parameters from the parameter file have been loaded into the application and displayed in a parameter menu, the user can freely edit these values and save back to the file. Whilst all of the parameters are editable, not every parameter effects the image produced in this application. It was however decided that it was important to include the entirety of the fields regardless of their effect on the final visualization so as to allow the application to act as a complete entry to exit point for a

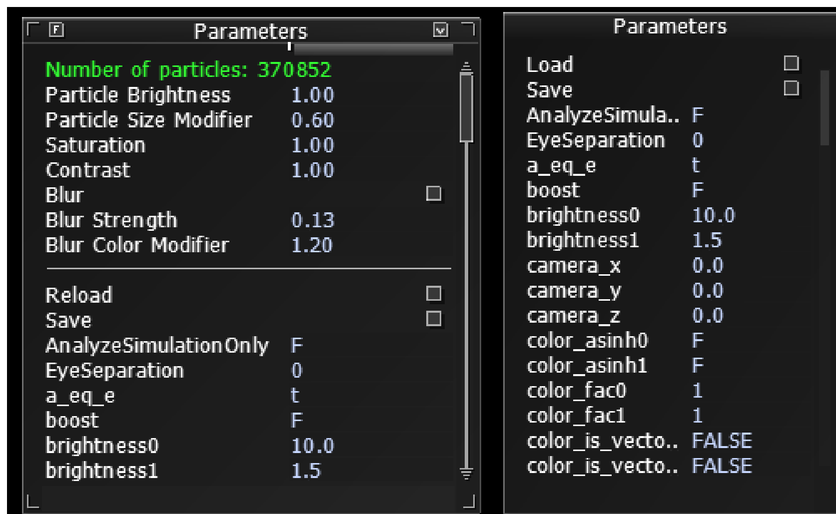


Figure 5.6.3-b Examples of loaded parameter files. The leftmost part of the image shows the parameter menu after having loaded the visualization- The upper most values above the separator are real time parameters that affect the renderer.

user to load, edit, and visualize data via Splotch on Windows. It also allows a user to take a parameter file edited within this application to use on the full Splotch software.

It was decided to use Cinders built in interface functions as it allows the application to quickly add functionality which includes the ability to bind a function to a button on the menu, such as the 'Load' button. It also serves to keep the applications size to a minimum by using functions already included in its libraries.

```
void SplotchCinder::createParams(bool mainScene, bool paramLoad)
{
    if (mainScene)
    {
        mParams->addParam("Particle Brightness", &renderer.brightnessMod).min(0.1f).max(10.0f).precision(2).step(0.02f);
        mParams->addParam("Particle Size Modifier", &renderer.particleSize).min(0.0f).max(1.0f).precision(2).step(0.02f);
        mParams->addButton("Blur", bind(&SplotchCinder::toggleBlur, this));
        mParams->addParam("Blur Strength", &renderer.blurStrength).min(0.0f).max(1.0f).precision(2).step(0.01f);
        mParams->addParam("Blur Color Modifier", &renderer.blurColorModifier).min(0.0f).max(2.0f).precision(2).step(0.01f);
        mParams->addSeparator();
    }
```

Figure 5.6.3-c Code snippet of loading GUI parameters.

5.6.4 Cinder Renderer

The renderer within the application is a very crucial part of the entire visualization application, as it produces how the application will visualize the data. As the application has the ability to deal with very large data sets with potentially millions of particles, it was also important to implement it in a way to attempt to keep as high a framerate as possible, with further thought to keeping the framerate at a stable point.

The render starts by loading the applications shader assets into memory, and setting the cameras clipping plane. The clipping plane is set to an appropriately high number so it can include all particles from the dataset which can potentially span distances outside the range of the default values within the virtual space.

The list of particles is then assigned to a new variable by calling a function from the `ParticleData` class which was passed into the renderer. Particle positions and respective colours are loaded into the shader via two SSBOs (Shader Storage Buffer Object) and drawn using an index buffer. SSBOs were chosen over UBOs (Uniform Buffer Objects) as they have the capacity to be as large as the graphics card memory limit (JeGX, 2014) which is helpful towards being able to render as many

particles as possible. The position SSBO is required to be a `vec4` for it to be able to be multiplied with the model matrix within the vertex shader. The colour alpha is always set to 1.

```
ci::vec4 *pos = reinterpret_cast<ci::vec4*>(mPos->map(GL_WRITE_ONLY));
ci::vec4 *color = reinterpret_cast<ci::vec4*>(mColor->map(GL_WRITE_ONLY));
for (size_t i = 0; i < particles.size(); ++i) {
    pos[i] = ci::vec4(particles[i].x, particles[i].y, particles[i].z, 1.0f);
    color[i] = ci::vec4(particles[i].e.r, particles[i].e.g, particles[i].e.b, 1.0f);
}
mPos->unmap();
mColor->unmap();
```

Figure 5.6.4-a Code snippet of mapping particle position and colour into the SSBOs.

Particles are drawn after invoking the `glBlendFunc` function in order to blend particle colours together where they overlap and simulate transparency. This results in an image which has a more realistic simulation of colours.

A Gaussian blur is also created on the image by rendering the particles to a FBO (Frame Buffer Object), then passing this FBO through a Gaussian blur shader twice to create horizontal and vertical blur FBOs which is then drawn over the first FBO via additive blending.

The index buffer is arranged in a regular way to create two triangles into a quad. Which is then computed into a circular sprite in the shaders and allows the application to present particles as 2D elements within the 3D space, known as billboards. This is for the purposes of performance within the application and the reason for using this method will be discussed in a further section.

5.6.5 Shader assets

The application currently computes the particle screen positions and colours using shaders. Shaders are computed on the GPU which means they are generally much faster in comparison to CPU computations (Rayne, 2014), in some cases this can up to 25 times as fast (Christensen, 2011). Using shaders with Cinder is relatively simple as it uses the GLSL shader language and even contains a class for easily loading a shader program. Computations were performed on the vertex shader when possible as a calculation on a fragment shader is much more expensive than a calculation on a vertex shader due to the fragment shader being executed once per pixel and pixel quantity in a scene generally outnumbers vertex quantity (Microsoft, 2016a).

```
void main()
{
    int particleID = gl_VertexID >> 2; // 4 vertices per particle

    vec4 particlePos = pos[particleID];
    vec4 colorV = color[particleID];

    vec3 colorV3 = vec3(colorV.x, colorV.y, colorV.z);  //Convert into a vec3
    colorV3 = ContrastSaturationBrightness(colorV3, brightnessMod, saturation, contrast);   //Read into function
    colorV = vec4(colorV3.x, colorV3.y, colorV3.z, colorV.w);   //Convert back into vec4
    Out.color = colorV;

    //map vertex ID to quad vertex
    vec2 quadPos = vec2( ( ( gl_VertexID - 1 ) & 2 ) >> 1, ( gl_VertexID & 2 ) >> 1 );

    vec4 particlePosEye = ciModelView * particlePos;
    vec4 vertexPosEye = particlePosEye + vec4( ( quadPos * 2.0 - 1.0 ) * iparticleSize, 0, 0 );

    Out.texCoord = quadPos;
    gl_Position = ciProjectionMatrix * vertexPosEye;
}
```

Figure 5.6.5-a Code snippet of the vertex shader.

As previously discussed, the vertex shader for the application takes the position and colour SSBOs and uses them to compute a circular billboard sprite. It takes four vertices per particle maps those to a quad vertex. Using these various variables, along with the projection and model matrices, it computes the position so that the sprite will always be facing the camera within the application.

When this information is then passed into the fragment shader, the sprite is still quad shaped. The fragment shader computes the circle sprite by finding a specified radius of a circle from the centre of the quad. If any fragments are beyond that, they are discarded.

Within the vertex shader the final colour for the particle is computed using its existing colour and any modifiers that are applied to it via the parameter menu. These can include brightness, contrast, and saturation.

A Gaussian blur can also be toggled on and off for the final visualization. This is done via a fragment shader using an offset specified by the user. If a user toggles the blur on, then, during the rendering stage of the pipeline, the FBO that the particles are drawn into are drawn into a second FBO via the Gaussian blur shaders. This is performed twice- once for the horizontal blur, and a second time for the vertical blur. The results from the second FBO are then drawn over the initial FBO via additive blending. This creates the final blur effect on the visualization.

A Gaussian blur is also used in the full Splotch software, and so this feature supports the requirement for creating images that replicate Splotch produced images. The result creates a much nicer final image, with particles that appear to be brighter when in close proximity to one another.

```
#version 330

uniform sampler2D   tex0;
uniform vec2        sampleOffset;
uniform float       colorModifier;

in vec2     vTexCoord;

layout (location = 0) out vec4 oFragColor;

void main()
{
    vec3 sum = vec3( 0.0, 0.0, 0.0 );
    sum += texture( tex0, vTexCoord + -10.0 * sampleOffset ).rgb * 0.009167927656011385;
    sum += texture( tex0, vTexCoord +  -9.0 * sampleOffset ).rgb * 0.014053461291849008;
    sum += texture( tex0, vTexCoord +  -8.0 * sampleOffset ).rgb * 0.020595286319257878;
    sum += texture( tex0, vTexCoord +  -7.0 * sampleOffset ).rgb * 0.028855245532226279;
    sum += texture( tex0, vTexCoord +  -6.0 * sampleOffset ).rgb * 0.038650411513543079;
    sum += texture( tex0, vTexCoord +  -5.0 * sampleOffset ).rgb * 0.049494378859311142;
    sum += texture( tex0, vTexCoord +  -4.0 * sampleOffset ).rgb * 0.060594058578763078;
    sum += texture( tex0, vTexCoord +  -3.0 * sampleOffset ).rgb * 0.070921288047096992;
    sum += texture( tex0, vTexCoord +  -2.0 * sampleOffset ).rgb * 0.079358891804948081;
    sum += texture( tex0, vTexCoord +  -1.0 * sampleOffset ).rgb * 0.084895951965930902;
    sum += texture( tex0, vTexCoord +   0.0 * sampleOffset ).rgb * 0.086826196862124602;
    sum += texture( tex0, vTexCoord +  +1.0 * sampleOffset ).rgb * 0.084895951965930902;
    sum += texture( tex0, vTexCoord +  +2.0 * sampleOffset ).rgb * 0.079358891804948081;
    sum += texture( tex0, vTexCoord +  +3.0 * sampleOffset ).rgb * 0.070921288047096992;
    sum += texture( tex0, vTexCoord +  +4.0 * sampleOffset ).rgb * 0.060594058578763078;
    sum += texture( tex0, vTexCoord +  +5.0 * sampleOffset ).rgb * 0.049494378859311142;
    sum += texture( tex0, vTexCoord +  +6.0 * sampleOffset ).rgb * 0.038650411513543079;
    sum += texture( tex0, vTexCoord +  +7.0 * sampleOffset ).rgb * 0.028855245532226279;
    sum += texture( tex0, vTexCoord +  +8.0 * sampleOffset ).rgb * 0.020595286319257878;
    sum += texture( tex0, vTexCoord +  +9.0 * sampleOffset ).rgb * 0.014053461291849008;
    sum += texture( tex0, vTexCoord + +10.0 * sampleOffset ).rgb * 0.009167927656011385;

    oFragColor.rgb = sum * colorModifier;
    oFragColor.a = 1.0;
}
```

Figure 5.6.5-b The Gaussian blur fragment shader.

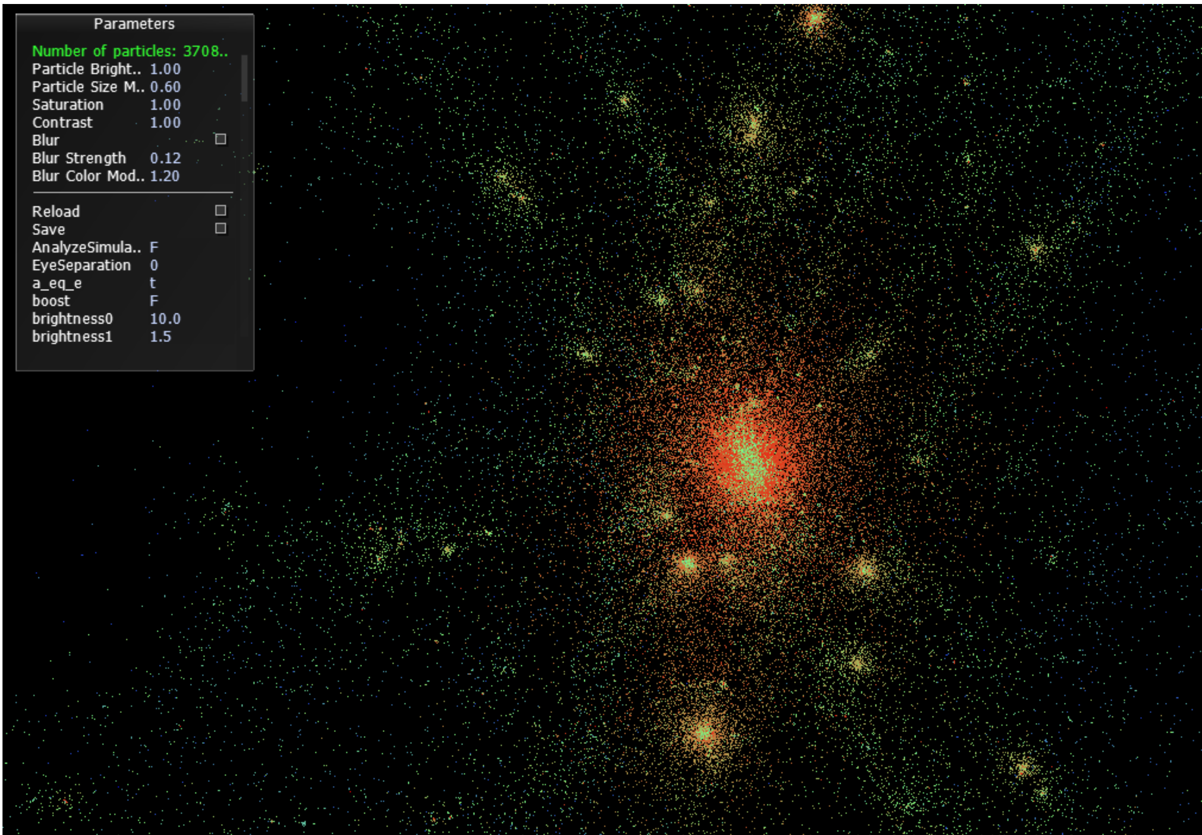


Figure 5.6.5-d A cropped image of the blur off.

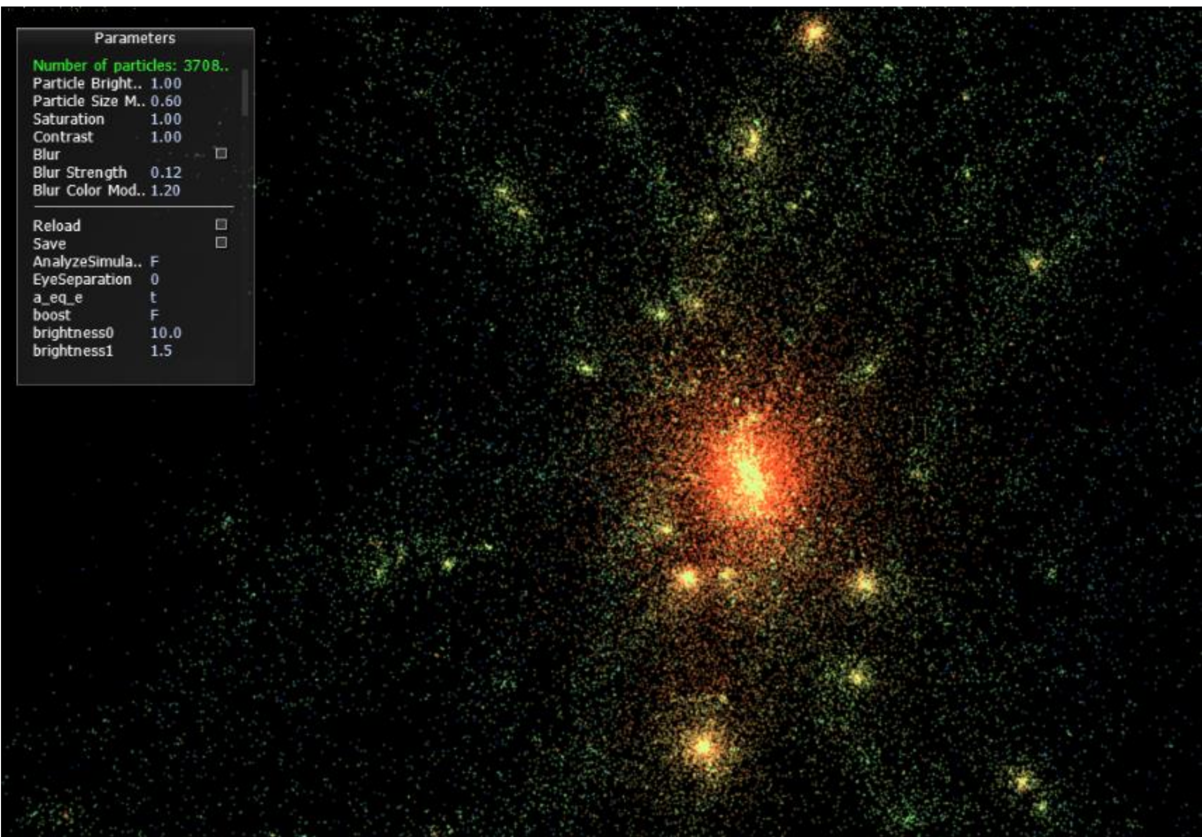


Figure 5.6.5-c The same image with the blur on. Notice how areas with a higher density of particles appear brighter.

5.7 PROTOTYPES

5.7.1 First Prototype

The initial prototype for the application came in the form of using Cinder to read in a text file filled with random x, y, z coordinates. It would then render a sphere in each position. This was useful in order to identify the best way to render shapes within Cinder. Whilst the method that was used in this prototype was easy to implement and use, it ultimately suffered bad slowdowns from using just 10,000 points. This was far away from the amount of points the application could potentially be drawing, and so a new method would have to be conceived going forward.

This prototype was completed under the period of time of learning Cinder in preparation for the rest of the project.

5.7.2 Second Prototype

The next few prototypes focused on the best way to integrate the Splotch files into the application and allow it to be compiled. The method for achieving this, was to identify which function in which file was needed to called from Cinder in order to retrieve the list of particles. At this point in time, the plan for the application was to use only the reader files from Splotch (and any relevant includes needed), and take the relevant particle list after using these- without the need to use the Previewer files. However, it seemed counter intuitive to disregard how the Previewer was working with the Splotch readers when it was performing a similar function to how the Cinder applications integration with Splotch was being planned. Therefore, it was decided for the pipeline to go through the Previewer files as they offered a framework which was already integrated with the core Splotch files.

After this point, the Previewer files were slowly added to the project one by one, meticulously resolving errors as they came up- the majority being from the differences in platform the software was ran on. The rendering portion of the files were all commented out, as the focus was on simply allowing the application to include and compile with the relevant Splotch and Previewer files. Any and all files were included if they were used by Splotch, regardless of whether the Cinder application would need them in the final application. This was to reduce time spent on porting files and classes which would not end up being used in the final application. Eventually the application was compiling, however no functions from Splotch were yet to be called. The application in itself, was just creating a window.

5.7.3 Third Prototype

The next prototype was focusing on allowing the application to specify a parameter file and being able to pass this into Splotch. When running the original Splotch, the path to the parameter file is specified in the command line. As the application would not be running from the command line, the path to the file was hard coded for the purposes of testing. The main problem in this prototype was adapting the way the Previewer manipulated the parameter file path due to the difference in file systems on a Windows and Linux system. One example of this was modifying the way the application finds the path its exe file. Which can be seen in the figures below. The variable containing the path to the exe does not end up being utilised in the final application, but at the time the prototype was attempting to emulate the same functionality, but on the Windows platform. The final state of this prototype had the application finding and utilising the parameter file, feeding the data into its appropriate reader, and attempting to read that data. It was at this point that errors were occurring due to out of range memory copies.

```
Parameter Previewer::parameterInfo;

void Previewer::Load(std::string paramFilePath)
{
    // Get path of executable
    int ret;
    pid_t pid;
    std::string exepath;
    char pathbuf[PROC_PIDPATHINFO_MAXSIZE];
    pid = getpid();
    ret = proc_pidpath (pid, pathbuf, sizeof(pathbuf));
    if ( ret <= 0 )
    {
        std::cout << "Could not get path of exe.\n";
        std::cout << "PID: " << pid << std::endl;
        exit(0);
    }
    else
    {
        // Remove executable name from path
        int len = 0;
        exepath = std::string(pathbuf);
        for(unsigned i = exepath.length()-1; i > 0; i--)
            if(exepath[i] == '/')
            {
                len = i;
                break;
            }
        exepath = exepath.substr(0,len+1);
    }

    #endif

    DebugPrint("Previewer path: %s\n", exepath.c_str());
```

Figure 5.7.3-b Code snippet of part of the original load function for finding the path of the exe of the application.

```
previewer::ParticleData Previewer::Load()
{

    char pathbuf[MAX_PATH];
    GetModuleFileNameA(NULL, pathbuf, MAX_PATH);
    // Remove executable name from path
    int len = 0;
    std::string exepath(pathbuf);

    for (unsigned i = exepath.length() - 1; i > 0; i--)
    {
        if (exepath.at(i) == '\\')
        {
            exepath.erase(exepath.begin() + i + 1, exepath.end());
            break;
        }
    }
```

Figure 5.7.3-a Code snippet of the modified function to work on Windows.

Time taken unsuccessfully attempting to fix these errors meant that the approaching deadline for the end of the coding period in the original plan had to be adjusted, as it would no longer be an accurate estimation.

5.7.4 Fourth Prototype

Following the successful implementation of the application finding and attempting to read the data, the next prototype focuses on solving the memory errors, as well as how we could visualize the particles within Cinder. Since the previous prototypes had been using a 9.24GB sized dataset, the application was tested on a smaller sized dataset to identify if this was the issue. Following tests on a 46MB sized dataset the application was able to successfully compile and load the data. This data was

not being rendered on screen at this point however. The particle list that was produced by Splotch was haphazardly fed back up through the pipeline of the application. The original renderer from the first prototype was then edited to work with this particle data. The application was able to render spheres in the positions that were read from the data.

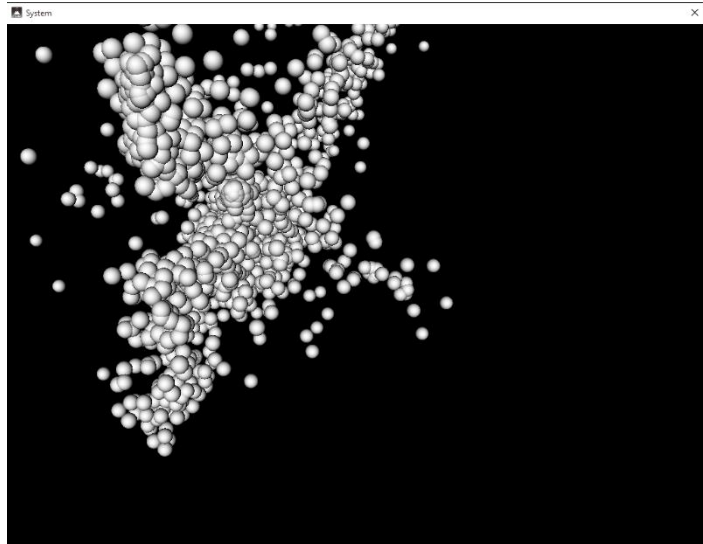


Figure 5.7.4-a Screenshot of the dataset being displayed as spheres.

Due to the heavy performance from trying to render around 370,000 spheres in this dataset, the application had to be set to only render a hundredth of the particles. As the strain from even rendering a hundredth of the particles was so high, the camera controls were not working correctly. This lead to the particles positions being scaled down so they could be seen by the camera more easily.

This prototype was the proof of concept that Cinder was able to correctly communicate with Splotch in order to render datasets.

5.7.5 Fifth Prototype

It was obvious that the renderer was not going to be able to render the particles in their correct positions as spheres, as this would strain the system massively. Therefore, this prototype was focused on creating a renderer which would be able to visualize the data in a meaningful and accurate way.

Initially it was planned to use the renderers that were included in the Previewer, so as to produce the same visualization, but via Cinder. A large amount of time was spent attempting to get these renderers to implement correctly with Cinder, however there were many compatibility problems which prevented this. These include OpenGL version discrepancies as well as platform differences. It was decided therefore to dismiss this plan, and write a dedicated renderer using Cinder functions. Ultimately, this seemed like the option that better suited the functionality requirements listed. As the application was using Cinder regardless, it should therefore attempt to use it in its fullest extent in order to exploit all advances that it offers.

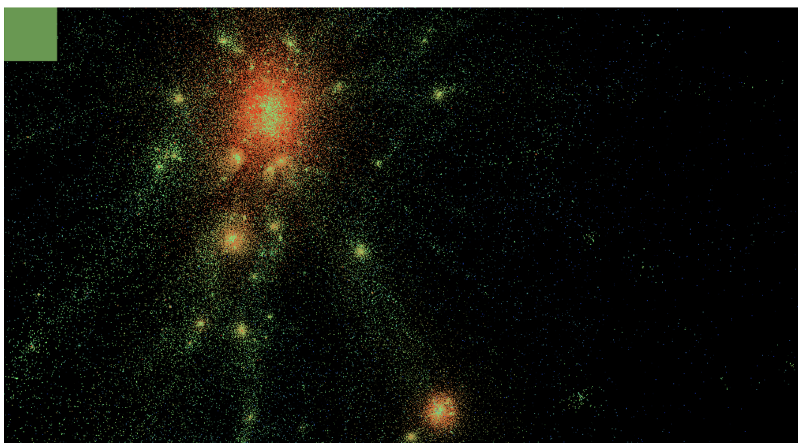


Figure 5.7.5-a Screenshot of the data being rendered via the initial Cinder renderer.

A dedicated Cinder renderer was therefore created as has been described in section 5.6.4. This prototype's renderer version is the basis for the final version which was described. The final version is however, much more streamlined and effective.

The unexpected added time that the creation of the dedicated Cinder renderer meant that the coding deadline had to again be pushed back.

5.7.6 Sixth Prototype

Now that the data was being rendered in a meaningful way, the focus for this prototype was implement a Gaussian blur. This was so the particles could appear to 'glow' where they are more concentrated and colourful. This is a relatively computationally cheap way to simulate the particles producing light.

The two pass Gaussian blur implantation is not the fastest way to calculate a blur as the application has to calculate each pixel through a n sized convolution kernel twice. The Cinder application has a kernel width of 21 as, after testing, this provided the best trade-off between image quality and performance.

At this point in the development cycle, the discussions and preparations for the TAO project were underway. This was an unexpected part of the project as a whole, and as such, in order to still be able to fulfil the requirements for this project, as well as being able to complete the TAO project, the testing cycle for this project was cut down.

5.7.6.1 Seventh Prototype

By this point, the application had successfully integrated Splotch in a way that would allow it to visualize the particles in a manner that is similar to the Previewer. The proposal for this prototype was to test and allow the application to be compatible with larger datasets.

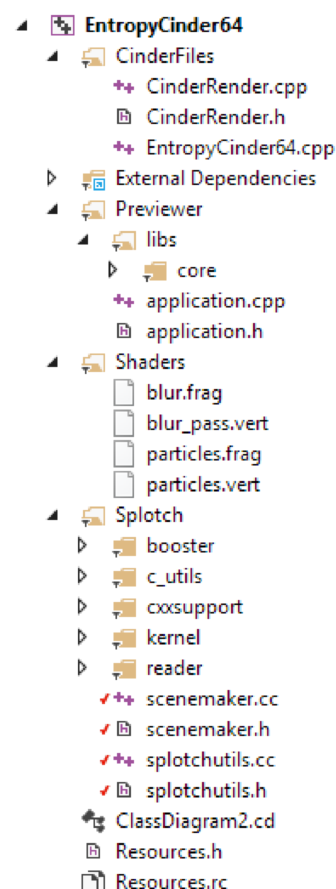


Figure 5.7.6-a Screenshot of the Visual Studio structure.

The only larger dataset available to the project at the time was a 9.4GB dataset in RAMSES code (Teyssier, 2002) and so tests of the application were based on this. The Visual Studio project for the application was created using Cinder's Tinderbox application. The default values for which create a 32bit application. Following investigation of the `memcopy` error that the application was producing when attempting to load this dataset, it was theorised that since the project was producing a 32bit application, it was unable to load any of the dataset past a 3GB limit as opposed on 32bit applications (Rutter, 2012). This was a design oversight when the project was initially planned. This meant that if the application was to load any data that was larger than 3GB it would have to be ported to be a 64bit application.

This porting was done by creating an entirely new 64bit project so the library linking could be correct and compliable before moving the existing code. This was a slow process, but it also offered an opportunity to clean up the project by removing unnecessary files that were still present, as well as organising the necessary files into an organised file structure. This made future work more streamlined.

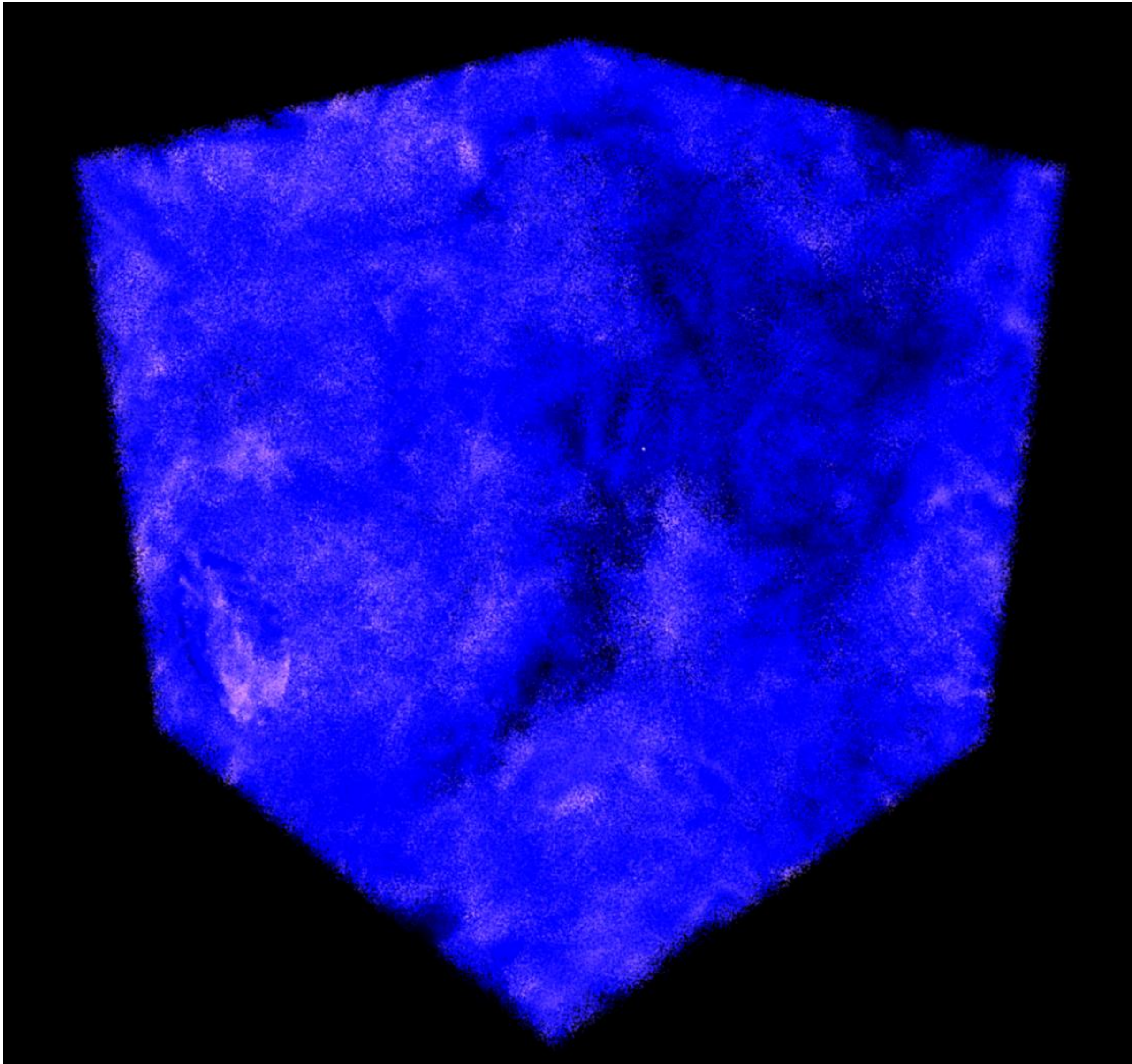


Figure 5.7.6-b Screenshot of the large 9.2GB RAMSES dataset being visualized within the Cinder application.

Following the porting of the application to a 64bit application, it was able to load and visualize the larger dataset. This does restrict the application to 64bit operating system, which lowers the accessibility level, but with 92.8% of new computers being sold worldwide, which have the Windows operating system, running a 64bit architecture (Popa, 2015) this trade-off is acceptable.

This unexpected prototype stage, which was the fault of an oversight during the planning period meant the estimation for completion of the project was again pushed back. The repercussions for this was that there was going to be less time for testing later.

5.7.7 Eight Prototype

The eighth prototype focused on adding in the ability for a user to change attributes for the visualization render in real time. Initially this took the form of the ability to toggle the blur and modify the blur strength. This eventually expanded into the ability to modify the individual particle size, as well as variables which effected its saturation, contrast, and brightness. It was decided to allow a user to modify these parameters to increase interactivity within the application. Whilst a default value is provided for each of the parameters, they may not be appropriate for every set of data, and so are modifiable within a specified range.

5.7.8 Ninth Prototype

The final prototype for the application adds the ability for the user to load and edit a parameter file before loading the relevant data. The requirements for this feature were:

- A user can point to a parameter file using an explorer window.
- A user can load and view all the fields from this parameter file.
- A user can edit these values.
- A user can save edited values back to the parameter file.

With these requirements in mind, a function was constructed to interact with the parameter file. It uses a Splotch function to get a reference to the loaded parameter file. It will then dynamically create an array based on the size of the parameter file. Following this, the “Load” and “Save” buttons are bound with their respective functions using a Cinder function.

The parameters are loaded into the dynamic array which is then used to create the parameter menu within the same loop. A separate array was used so as to allow the application to have its own copy of the parameters, which it could edit and save freely and independently of the original values. This gives the potential for the application to know which values have been modified and their difference against the original values. This could be used to create a feature at a later date which interpolate between different colour palettes for example. The function to write back to the parameter file uses the standard C++ `ofstream` class. The function writes the edited parameters back to the file line by line, before deleting the dynamic array, and rereading the parameter file with the new values.

```
    if (paramLoad)
    {
        allParams = myParamFile->getParams();
        string first, second;
        int count = 0;
        paramArray = new string*[allParams.size()];

        for (int i = 0; i < allParams.size(); ++i)
        {
            paramArray[i] = new string[2];
        }

        if (mainScene) mParams->addButton("Reload", bind(&SplotchCinder::loadMainScene, this));
        else mParams->addButton("Load", bind(&SplotchCinder::loadMainScene, this));

        mParams->addButton("Save", bind(&SplotchCinder::writeToParameterFile, this));

        for (params_type::iterator it = allParams.begin(); it != allParams.end(); it++)
        {
            paramArray[count][0] = it->first;
            paramArray[count][1] = it->second;
            mParams->addParam(paramArray[count][0], &paramArray[count][1]);
            count++;
        }
    }

    mCamUi = CameraUi(&renderer.mCam, getWindow());
}
```

Figure 5.7.8-a Code snippet of dynamically loading the parameters.

At this point in time, the project was past its coding period deadline and into the testing stage of the project. Although tests were carried out at the end of each prototype, a thorough dedicated testing period that was planned to take place at this point in time, was unable to be performed due to the longer than expected coding implementation, and also the commitment to the TAO project.

5.8 TESTING

In order to ensure that the application is fulfilling its requirements it needs to be tested thoroughly in an orderly framework. Myers, Sandler, & Badgett (2011) talk about the complexity and necessity for extensive testing before going on to define software testing as:

> …a process, or a series of processes, designed to make sure computer code does what it was designed to do and, conversely, that it does not do anything unintended. (Myers et al. 2011, p. 2)

The importance of testing is stressed by Samaroo, Thompson, & Hambling (2015, pp. 11-12), who explain that software failures can potentially lead to loss of money, time, and reputation, and in extreme cases such as software for hospital or aircrafts- injury or death. Although any errors in the application for this project are not likely to cause any dramatic problems such as these, it serves as an example of how it is still critical to test the application to ensure it runs in its intended manner.

Jorgensen (2013) states that "the essence of software testing is to determine a set of test cases for the item to be tested". He goes on to talk about "Reference Testing", where a system is judged by "expert users" who can tell if the application is producing the correct results. In the case of the application in this project, this kind of reference testing is being performed, not by "expert users" but by its comparison to the Splotch output image of the same datasets. The Splotch image can be considered to be the "expert produced result" and determining if the applications visualization is producing a similar enough result can be considered to be the "reference test". Of course this comparison does not produce any tangible or concrete test results unless we clearly specify what the criteria for "similar enough" is.

5.8.1 White and Black-Box Testing

Black-box testing (also known as input/output driven testing, or functional testing) is a type of testing where the tester is only concerned with the output of the program given the inputs, and the inner workings is not of concern. Test cases for black-box testing have two advantages according to Jorgensen (2013, pp. 7-8):

1) They are independent from the software implementation, meaning that the test cases can still be useful even if the implementation changes.
2) Test case development can occur in parallel with the software development, reducing overall development time.

Meyers et al. (2011, pp. 9-10) discusses that it is not possible to exhaustively test an application in this manner, as a tester would have to make use of every possible input condition as a test case, and this is simply not possible in any non-trivial program.

A contrasting testing approach is known as white-box testing (also known as logic-driven testing, or structural testing), where the tester is permitted to examine the internal structure of the program. Meyers et al. (2011, p. 10) explains that the most thorough application of this method is to execute all possible paths of control flow through the program. However, the number of paths could be unfeasibly large, and this does not provide a realistic approach. It seems then, that a mixture of both testing methods should be applied in order to be able to provide, as closely as possible, a complete but realistic approach to testing the application.

5.8.2 Module (Unit) Testing

Module, or Unit, testing is a method of testing that focuses on breaking down the application into a series of modules which are tested individually. This allows the testing method approach to be more manageable, especially when dealing with large applications. As a module is isolated and tested away from other code, this eases debugging tasks, as an error will be known to have occurred within the tested module, as Meyers et al. (2011, p. 85) mentions.

Defining test cases for module tests consists of defining the input and output for the module and then analysing the source code and logic flow using a white-box method.

5.8.3 Testing Implementation

For the purposes of the artefact, a combination of module testing, and black-box testing was chosen. It was decided to test the Splotch function using the black-box method, whilst module testing would be performed on the core application.

As the Splotch software is an existing piece of software with its own independent development, it is assumed that the software has been, and is, the subject of its own testing completely separate from this project. Therefore, as the source code for Splotch is subject to change without regard for this projects application, it seemed that the testing for the project should only be concerned with the inputs and outputs for these functions. It was also unrealistic, given the timeframe of the project, to be able to thoroughly test the Splotch software and the projects artefact in unison.

The testing for the Cinder reliant parts of the artefact would be performed using the unit testing method, as this is a much more manageable scale to perform this method on. Functions were tested individually as per their requirements of use at the end of each prototype cycle- these can be found in the appendices.

As there was a limited availability of appropriate datasets available to act as an input for the project, testing for as many inputs that were desired was not possible. This severely limited the effectiveness of testing. Furthermore, due to the unexpected opportunity for the project described in section 6, the testing for the project described under this section was not as thorough as originally planned, as time was diverted onto the new project. Because of these reasons this means that there are likely a number of unknown bugs still in the application.

5.9 Evaluation and Future Work

5.9.1 Summery

Looking at the complete artefact of the project, it can be argued that is has largely been a success. The application has fulfilled all functional requirements that were gathered at the start of the project. The application has been designed in a way that allows it to be scalable and reliable, using justifiable decisions. The application is successfully using Cinder functionality in order to work. Although this functionality can be considered to only be shallow in terms of what Cinder can offer, the application is built in a way that allows it to easily integrate further with Cinder. This being said, the application serves as a proof of concept that Splotch can be used in a more accessible environment, and also serve as a framework for loading and sorting astronomical datasets into a form that is easily exploited.

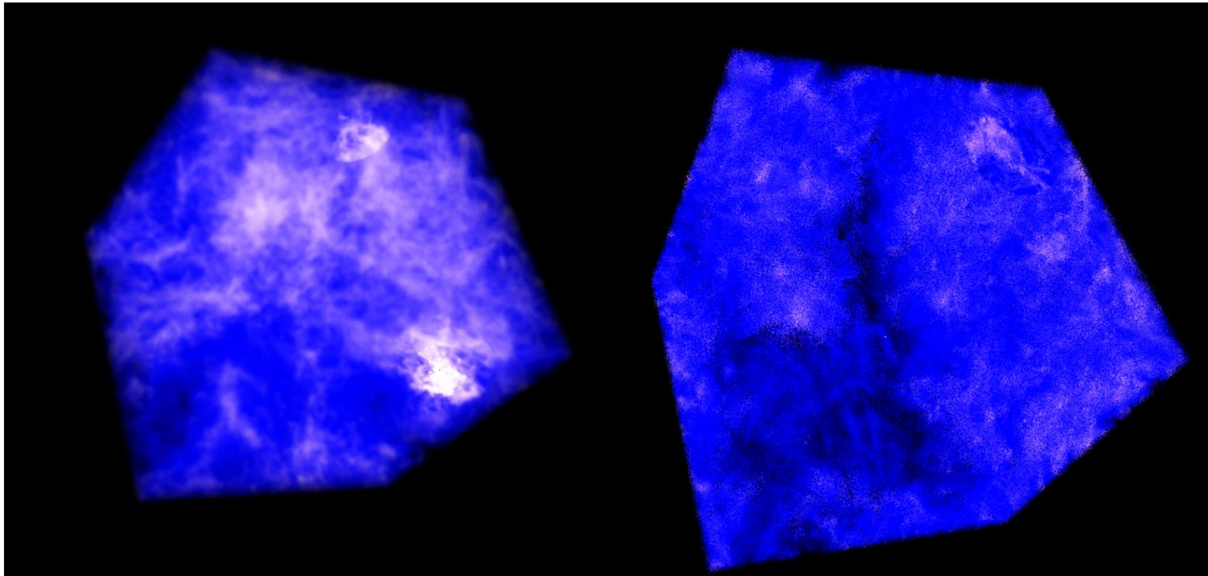


Figure 5.9.1-a Comparison between a Splotch produced image (left) and the Cinder application's produced image (right) of the same dataset.

The image that can be produced with the Cinder application is not as accurate as the Splotch produced image, however it is rendered in real time within an interactive 3D visualization.

5.9.2 Performance and Optimisation

The application has successfully been tested to be able to visualize at least 29,358,022 particles at an acceptable frame rate (more than 30 frames per second). This was performed on a computer with an Intel I5-3570 CPU at 3.40GHz, 16GB of RAM, and an Nvidia GTX 970 2GB graphics card. In order to render a dataset, the target computer must have at least the size of the dataset in free RAM. This is the largest potential issue for users of the application and would be the priority of further optimisation.

The application makes use of Splotch's OpenMP support which increases the load time of the datasets by allowing the application to use multiple threads (Microsoft, 2016b). The implementation of Splotch's support for Open MPI and CUDA could also be helpful for decreasing load times for larger datasets, but are not currently implemented in the final artefact.

The Gaussian blur is using a linear sampling method which takes advantage of fixed function GPU hardware, and can increase the performance of the blur by up to 60% compared to using a discrete sampling method (Rákos, 2010). The blur currently has to do two passes through the shader (once

horizontally, and once vertically) which gives it a complexity of $O(n)$ for each pixel, this is faster than a single pass Gaussian blur, which would have a complexity of $O(n^2)$. Blur performance could further be increased by down-sampling the original FBO to a half resolution, rending the blur FBO, and then up-sampling this back to the target resolution (Filip, 2014).

Although care has been given to allow the application to be as performance conscious as possible, given more time for the project, the focus would to be on optimising the application to work on a broader range of computers.

5.9.3 Methodology

The methodological approach taken during the time of the project served mainly as a framework, rather than truly influencing the project timeline or functionality. It allowed the project to flow in a way that focused on the development and implementation through testing, rather than excessive planning which could lead to time being wasted in an already short development cycle.

Due to the nature of the project and the feasibility of the Splotch implementation working at all, the methodology allowed the project to be broken up into several smaller parts which each focused on a single aspect of the application. This was helpful in tracking progress and was overall a good fit for a project of this kind.

5.9.4 Further Features

If the project were to be developed further than providing more optimisation, then new features would be added which would mostly take advantage of additional Cinder functionality. Further features could include:

5.9.4.1 Particle textures

Mapping textures onto individual particles could allow for a more aesthetically pleasing and realistic visualization. These could also be animated using Cinder’s ability to load the gif file format (Kepler, 2015).

5.9.4.2 Real time light producing particles

Using real time lighting on the visualization would further increase its accuracy and aesthetics. Due to the potential number of particles, this system would have to be very well designed so as to be as optimised as possible, and could potentially be a large undertaking.

5.9.4.3 Tested and supported on OS X

As Cinder officially supports the OS X operating system, the application should also be extended to be able to run on this platform. Support for OS X was outside the scope of this project, but as Splotch already has support for running on a Macintosh computer, porting and testing the application for this purpose would be a desired functionality.

5.9.4.4 Tested and supported on mobile platforms

iOS version 6.0 and later is listed as having official support for the Cinder library (Cinder, 2015d). It should therefore be possible for a port of the application to be running on devices supporting these versions.
Android support for Cinder is not listed as having official support and is still undergoing active development (chaoticbob, RFC: Cinder for Android and Linux, 2015). Targeted platforms are Android KitKat, Lollipop, and Marshmallow. This should be considered as a potential platform for the application when it has been provided with official support. As Android uses a version of the Linux kernel (Android Interfaces and Architecture, n.d), and the Cinder development is producing the

Android and Linux port in unison (chaoticbob, Android and Linux port #1200, 2015) a Linux port could also be developed alongside an Android version.

5.9.4.5 Further interactivity within the visualization

This could include informative text that appears over particular parts of the visualization This could be integrated with various hardware that Cinder supports- such as VR (virtual reality) headsets like the Oculus Rift (Hodgin, Oculus Rift: Gravity, 2013), or the Leap Motion device (Selikoff, 2013). These can even be combined to produce an extremely immersive experience (Hurlbut, 2014). Using this hardware together could produce an application where the users entire field of vision is encompassed by the HMD (head mounted display) whilst they can pan, and zoom using hand gestures with the Leap Motion.

5.9.5 Recommendations

A few specific recommendations can be suggested for any projects that are similar in nature.

5.9.5.1 Cinder

- The samples included with the Cinder download are the best way to understand how to use the library correctly. The documentation for Cinder is sparse and does not necessarily have an explanation of how to use the library correctly, but the samples are commented with useful advice which can help. The samples can also highlight useful functions, not mentioned in the guides.
- Create an account on the Cinder forum and ask for help or advice from users as there are a number of very experienced Cinder developers on the forum who are always willing to offer this.

5.9.5.2 Splotch Integration

The largest use of time in this project was the studying and understanding of the Splotch software in order to correctly implement it within the application. Minimise that time by:

- Having a good understanding of the Linux system in order to be able to test Splotch and its capabilities. Editing the parameter files and studying the resulting image produced through Splotch can be very insightful in understanding how parameters effect the image. A Splotch manual (see the first draft in the appendices) has been created in conjunction with this project which should also offer insight into the installation process as well as important parameter variables. The final version will be available with the Splotch download.
- Adding the Splotch files into a project within Visual Studio allows a better understanding of how the source files interact with one another. Visual Studios features can help better identify call hierarchies and program flow.
- Work backwards from the intended functionality. Look at what the functionality will perform and decide which functions and dependencies from Splotch need to be accessed in order to perform this functionality.
- Test often and well in order to highlight problems early on, so that they can be fixed before moving forward. It is critical to be able to perform proper debugging in order to be able to understand how the software is working.

6 INTEGRATING SPLOTCH FOR USE ON THE SWINBURNE GSTAR SUPERCOMPUTER

6.1 INTRODUCTION

During the latter stages of the previous projects development cycle, an opportunity was presented which would allow the chance to integrate Splotch to work within a new HPC environment. This was facilitated by developers of Splotch and would form a client based project through said developer for the Centre for Astrophysics and Supercomputing, based out of the Swinburne University of Technology in Melbourne, Australia.

The Green II HPC system at the supercomputing centre contains two facilities: the GPU Supercomputer for Theoretical Astrophysics Research (gSTAR), and the Swinburne Supercomputer for Theoretical Academic Research (swinSTAR) (Hassan, n.d). Both systems share a 3 Petabyte Lustre file system.

This project would be using the gSTAR system which is a collection of 50 nodes, each containing:

- 2 six-core Westmere processors at 2.66 GHz (each processor is 64-bit Intel Xeon 5650).
- 48 GB RAM.
- 2 NVIDIA Tesla C2070 GPUs (each with 6 GB RAM).
  (Hassan, n.d)

The main goal for the project would be to have a Splotch executable working on the gSTAR system that would allow the client to take advantage of the HPC environment in order to produce images using large datasets via Splotch. This executable would go on to be part of the Theoretical Astrophysical Observatory (TAO) interactive system (Theoretical Astrophysical Observatory (TAO), n.d).

The TAO is an online virtual laboratory that houses mock observations of galaxy survey data (Bernyk, et al., 2016). It uses the gSTAR system as a backend which hosts a scalable dataset cluster. Users interact with TAO through a web interface where they can select from various options to define the properties of the visualization requested. The user is the notified via email which allows them to then download the outputs to their local machine (Bernyk, et al., 2016, p. 2).

TAO offers a unique service for astronomers which requires no programming knowledge in order to use the system (Bernyk, et al., 2016, p. 3). This is in the spirit for keeping the barrier of access low, and is the exact same sentiment that was used for the creation of the Cinder application in the previous section of this text. Further similarities are noticed within the initial proposals for the Cinder application, which suggested a system similar to TAO (see section 5.4.1.1 and appendix 1). However, TAO aims to produce static and accurate images for astronomers (Bernyk, et al., 2016, p. 16), whereas the Cinder application focused on a broader interactive visualization for the public outreach and the Cinder community.

6.2 Initial Planning

Planning for this project took on a more practical approach as the timeframe for completion was short. Requirements for the project were gathered via email correspondence with the client and via discussions with the Splotch developer contact. Initial requirements gathered for the project were:

1. Splotch being compiled on the gSTAR system.
2. Splotch is able to produce an image of a sample dataset.

Initial requirements were kept relatively simple as there was a going to be a reasonable amount of time to understand how to work within a HPC environment. The plan was simply to copy the Splotch files onto a personal home directory, and then compile and run this with the snap_092 data that is provided with the Splotch download.

Working with a HPC environment was not the simplest adjustment to make. It entails working purely off of the Linux command line, which in itself requires a knowledge of the relevant commands. However, working with Splotch on a Linux installation for the previous project lent itself greatly to the task at hand, and meant that the learning period was not too great.

Two new concepts, were the use of environment modules, and the job queue for requesting jobs on the system. Environment modules are a way to "dynamically modify a user's environment via modulefiles" (Furlani, 1991). Use of modules was a relatively straight forward use of commands and offered a way to quickly switch out versions of modules when needed. It was however, cumbersome to re-add all the appropriate modules when working on a new node. These module commands could have been added via an executed script, however, as different module versions were being tested constantly, this was never done.

The job queue was also a relatively simple learning process. It was simply a case of understanding the commands, and also the hardware available for use. As the wall time for most of the use of the nodes was lower than 30 minutes, jobs were, if hardware requested was accurate, granted instantly.

Following these adjustments, Splotch was compiled on the gSTAR system using a generic Linux makefile option. The snap_092 data was then used to produce an image.

After further correspondence with the client via email, a sample test dataset was provided that the Splotch software would be required to process. This file, mini_millennium.h5, was a 372MB file in the HDF5 format. The output of the contents of the file showed that the data was stored as compound data, which is a format that is similar to a struct in C (The HDF Group, 2015), where member variables were stored under grouping 'datasets'.

The advantage of using HDF5 as a file format is that it is fast to access and fully portable on any platform, even with applications written in different programming languages (The HDF Group, 2011). It is also designed to support parallel I/O (input/output), and has no theoretical limit to the size of data that is can support (The HDF Group, 2011).

In regards to using this dataset within Splotch, Splotch already has a HDF5 parallel reader, however this reader could only read files which have 1D or 3D datasets with no groups. This meant that the reader would have to be adjusted to accommodate the dataset. Therefore, the new client requirement was:

1. Have Splotch be able to read and produce an image of the mini_millennium.h5 dataset on the gSTAR system.

For the purposes of the project workflow, this was broken down, at the start of development, into sequential subtasks which are as follows:

1. Gain an understanding of the HDF5 format.
2. Source a smaller compound HDF5 file for purposes of testing and studying the Splotch reader, and how it responds to parameters within the parameter file.
3. Adapt the reader to read HDF5 compound datasets.
4. Test the reader on the mini_millennium.h5 file.

Further goals not strictly relating to the requirements at this stage were as follows:

1. Compile Splotch to take advantage of using Open MPI.
2. Compile Splotch to take advantage of CUDA hardware.

Following the definition of these subtasks, the main work of the project was undertaken and the results of which can be found in the succeeding section.

6.3 Adaptation of the HDF5 Reader

The adaption of the reader was written in a way so as not to interfere with the original readers' functionality. The first step was therefore, to let the reader know that it would be dealing with compound data, as it would need to function different dependant on the datatype. As is possible with HDF5 compound data, a single file could hold multiple structures, each under a different dataset parent, therefore, a user would need to specify which dataset they wished to access. The simplest way to do this was to add two new parameters within the parameter file: `is_compound_data`- which is a Boolean value to tell the reader if it is dealing with compound data or not, and `dataset_name`- which is the name of the dataset that the user wishes to read.

```
bool isCom = false;
string comParam = params.find<string>("is_compound_data", "-1");
if (comParam.compare("TRUE") == 0) isCom = true;

if (isCom)
{
    string datasetName;
    datasetName = params.find<string>("dataset_name");
```

Figure 5.9.5-a Code snippet of the reader checking the new parameters.

In order to read from the compound dataset a structure was added to the reader which would act as the buffer to store the data. Each field of the data (in this case the x, y, and z position, colour information, smoothing length, intensity, and correlating snapshot of each particle) is replicated in the structure. The reader will create an array of these structures to the size of the amount of particles that the current parallel process has been assigned.

```
struct fileData
{
  float x;
  float y;
  float z;
  float C1;
  float C2;
  float C3;
  float r;
  float I;
  int snapshot;
};
```

Figure 5.9.5-b Reader buffer structure.

Following this the reader will create a new compound datatype the size of the structure, which the reader then inserts with the members that were defined in the structure. The reader loops over all the fields, using a switch statement to only insert the fields that have been specified in the parameter file. This will save the reader from reading fields that are not used. This then, gives us a compound datatype that is an effective copy of the structure, and is used to define how the data is read into the buffer when the file is read. Memory dataspace and dataset dataspace is also set before the read in a way that it is compatible with parallel processing.

After the file has been read into the data buffer, the objects are then closed, and the data can be accessed like a regular structure. The reader will then copy the data into the particle vector that is used by the rest of the Splotch software. This process will check if the particle has the correct user specified snapshot ID (set in the parameter file), and if so will then proceed with the copy. It will keep an independent count of the particles that have been copied, so that the vector can be resized so as not to later render empty points.

```
int64 j = 0;
for (int64 i=0; i<npart; ++i){
    //cout << fileDataBuffer[i].snapshot;
    if (fileDataBuffer[i].snapshot != snapshotID) {continue; }
    points[j].x = fileDataBuffer[i].x;
    points[j].y = fileDataBuffer[i].y;
    points[j].z = fileDataBuffer[i].z;
    points[j].e.r = fileDataBuffer[i].C1;
    points[j].e.g = fileDataBuffer[i].C2;
    points[j].e.b = fileDataBuffer[i].C3;
    points[j].r = fileDataBuffer[i].r;
    j++;
}
points.resize(j + 1);
```

Figure 5.9.5-c Code snippet of the data being copied into the points vector.

Further modifications to the reader include an added Boolean parameter named `print_snapIDs_in_file_ONLY`, which will only read the snapshot numbers that each particle corresponds to in order to collect all the snapshots that are available in a file. The reader then sorts the vector they are stored in and prints to console, before exiting. This is useful if the snapshots available are unknown to the user.

```
if (printSnaps)
{
    for (int64 i = 0; i <npart; ++i)
    {
     if(std::find(snapshotsInFile.begin(), snapshotsInFile.end(), fileDataBuffer[i].snapshot) != snapshotsInFile.end()) continue;
     else snapshotsInFile.push_back(fileDataBuffer[i].snapshot);
    }
    mpiMgr.barrier();
    if (mpiMgr.master()) cout << "TOTAL SNAPSHOTS IN FILE " << snapshotsInFile.size() << endl << "SNAPSHOTS AVAILABLE: " << endl;
    std::sort(snapshotsInFile.begin(), snapshotsInFile.end());
    for (int i = 0; i <snapshotsInFile.size(); i++)
    {
        if (mpiMgr.master()) cout << "Snapshot: " << snapshotsInFile.at(i) << endl;
    }
    planck_assert(false, "Only reading snapshots- Exiting...");
}
```

Figure 5.9.5-d Code snippet of the reader printing snapshot numbers.

6.4 FURTHER WORK

On completion of the adaptation of the HDF5 reader, Splotch was compiled to take advantage of the Open MPI library. This was a case of editing the makefile to link to the correct library locations on the gSTAR system. The same was also completed with the CUDA libraries at a later time.

Using 12 MPI tasks on a single node, a set of 55 images were rendered of each snapshot in the mini_millennium.h5 dataset. These were then rendered in a video to form a timeline of the data (please see files on disc) and, along with technical information regarding rendering time, sent to the client via email. Details on rendering times and tests can be found in a later section.

Following a call with the client, a larger dataset (181GB in size) was made available on the gSTAR system named sage-millennium.h5. Further requirements were also gathered in regards to how to prepare Splotch to be integrated onto the TAO interactive system. These include:

1. **A calculated camera place-** there should be some way for Splotch to calculate a camera position relative to the data, without the need to manually set one.
2. **Default brightness and smooth length values**- These should either default to a hard coded value to act as a starting value to further tweak, or they should be calculated relative to the data.

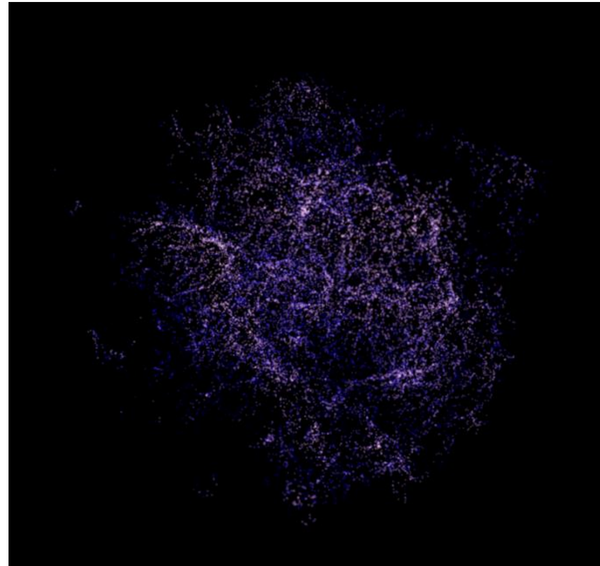


Figure 5.9.5-a Snapshot 39 of the mini_millennium.h5 dataset.

Deliverables were also agreed to be a compiled Splotch exe file, and an example parameter file.

Fortunately, in regards to the first requirement, a system like this was already in place within the Previewer for Splotch. Porting this over to the main Splotch system was a case of adapting the core Splotch code to accommodate it. A new `BoundingBox` structure was added to the `splotchutils.h` file- which has the ability to compute the maximum and minimum particle positions for each axis and thus define a box. A new class was also created named `Camera_Calculator` which would hold the functions needed for computing the camera position relative to the data.

```
if (params.find<double>("camera_x", -1) == -1 && params.find<double>("camera_y", -1) == -1 && params.find<double>("camera_z", -1) == -1)
{
  cout << "No camera position supplied. Calculating suitable position" << endl;
  string face = params.find<string>("face", "-1");
  int fov = params.find<int>("fov", 30);
  bbox.Compute(particle_data);
  camCalc.calculateCameraPosition(bbox, face, fov, campos, lookat, sky);
  centerpos = campos;

  cout << "Camera position-> x = " << campos.x << ", " << "y = " << campos.y << ", " <<  "z = " << campos.z << endl;
}
```

Figure 5.9.5-b Camera calculation code using the bounding box and new Camera_Calculator class.

In order to make these requirements as streamlined as possible, an automated camera position will be calculated if the any of the camera x, y, or z positons are not defined in the parameter file. A parameter can also be set to define the face of the bounding box for the camera to focus on if required. If this is not set, it will default to the front of the box. The field of view will also be taken into account when calculating a position in order to fit the entire box in the final image, but this too, will default to a defined value if it is not set in the parameter file.

The brightness and smoothing length values are two variables that are very specific to each dataset, and so it became difficult to determine a default value for these. One possible way to address the brightness, would be to devise an equation which would take into account the amount of particles to be rendered and increase or decrease the brightness value accordingly, i.e., if the scene has less particles, the brightness increases, and vice versa. The smoothing length would also have to be set in a similar way. This however, would need to be included in each reader, as there are slightly different methods of using it dependant on the reader.

6.5 OPTIMISATION AND TESTING

Initial benchmarks for the snapshot with the highest amount of particles in the mini_millennium.h5 file (this was snapshot 39 which has 37,428 particles) was at around 0.6s with 12 MPI tasks. Running the benchmark on a snapshot with a very small amount of particles (snapshot 12 has 28 particles) also seemed to take around 0.6s. This led to the interpretation that 0.6s was the 'base' time for this dataset, and any difference in particle amount was too small to make any meaningful difference in time. Therefore, the larger sage-millennium.h5 file was used for the basis of the testing. This dataset contained 752,849,670 particles across all of its snapshots.

Previous to the final version of the reader that was implemented on the gSTAR system (as described in section 6.3), an earlier adaptation was created and tested. This version, like the final version, used a structure to act as the buffer for the data from the file. However, unlike the final version, each field of data would have a separate array of structures, each with a single float member (4 bytes) holding the fields data. This means that, if $x = number\ of\ particles$, then the memory size during each fields read would be $4x$ bytes, e.g., in the sage-millennium.h5 dataset, if reading all particles, each fields read would take 3.01GB of memory.

The reader would copy all particles, from all snapshots, into the final particles vector, meaning that the particles vector most likely contained more points that were not going to be rendered, than it did points that were. During its read of the particles corresponding snapshot number, the reader would add the position of any particles with the incorrect snapshot to a separate vector. Finally, the reader would loop over this vector, and change the colour of any particle that the reader did not want to see in the final image to black.

To summarise, this meant that, although the final image would not include the incorrect particles, they were still being read into the particles vector. As each particle uses 30 bytes of data (Dolag, Reinecke, Gheller, & Imboden, 2008) this meant that the particle vector for the sage-millennium.h5 file would be 22.58GB in size, regardless of the snapshot specified. To give an example, snapshot 12 contains 187,507 particles. This would only have 5.625MB worth of particles and is just under 0.025% of the total particles in the file. However, the reader would still read the further 99.975% of the file.

An optimised version was therefore created as has been described in a previous section. The optimised version will still read the entirety of the data into the buffer using

$$(4 * number\ of\ particles) * number\ of\ fields$$

in bytes of memory at a single time. However, it has the advantage of checking if a particle has the correct snapshot before it is read into the particles vector. This saves the reader from having to create the vector of inactive particles and the subsequent loop process of setting them to black. It also has a lower chance of producing errors, as the particles vector only contains the points that it will eventually render. This too, helps the automated camera calculation, as it is not taking into account the invisible particles during its calculations. The problem however, with the current iteration of the reader will come when the dataset is large enough for this memory usage outlined above to be more than the system has to offer.

6.5.1 Benchmarks

The benchmarks for that were carried out on the dataset can be found in the table below. All benchmarks were performed on snapshot 28 of the sage-millennium.h5 dataset (containing 16,042,057 particles), using calculated camera positions at its 'FRONT' position. Splotch has been compiled with Open MPI version 1.8.3, CUDA version 7.5, GCC version 4.8.2, and HDF5 version

1.8.15. Configuration for CUDA is 1 GPU per MPI task. Benchmarks were performed a total of three times each, with the fastest time for each used in the comparison. If significant differences in time were observed after performing three benchmarks (more than 20 seconds) this time would be declared as an outlier and the benchmarks were performed twice more. The resulting image was deleted after each benchmark, before running the next. The fastest time was then used from the remaining non-outlier times, which generally have a range of less than 3 seconds. It was not uncommon for benchmarks to occasionally take a significantly longer than previous benchmarks and benchmarks for the first version of the reader seemed to show faster time after the first two benchmarks. The reason for this is currently unknown, but can perhaps be put down to hardware performance. Regardless, the fastest time was used so as to show the speed than can be achieved. The maximum amount of individual nodes that the gSTAR system would grant via the jobs system was 40.

Version	8(1)	16(1)	24(1)	24(2)	36(1)	40(1)
First Version	396.9753	64.6225	46.9483	26.3498	32.3037	28.2916
Second Version	114.6673	19.8179	12.4716	8.2426	8.8676	9.5299

Table 2 Table of benchmark showing each versions wall time in seconds against nodes (task per node) used.

Version	8(1)	16(1)	24(1)	24(2)	36(1)	40(1)
First Version	7	5	9	7	5	3
Second Version	5	3	5	3	3	3

Table 3 Table of number of benchmark performed per amount of nodes (tasks per node) used.

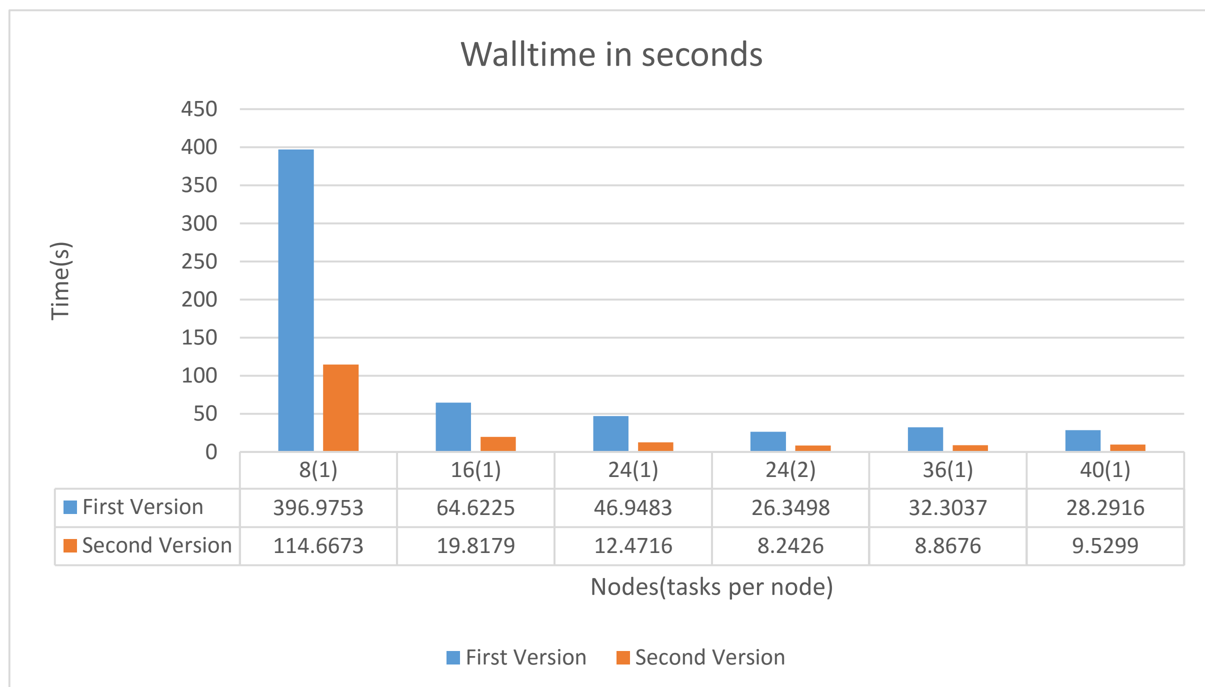
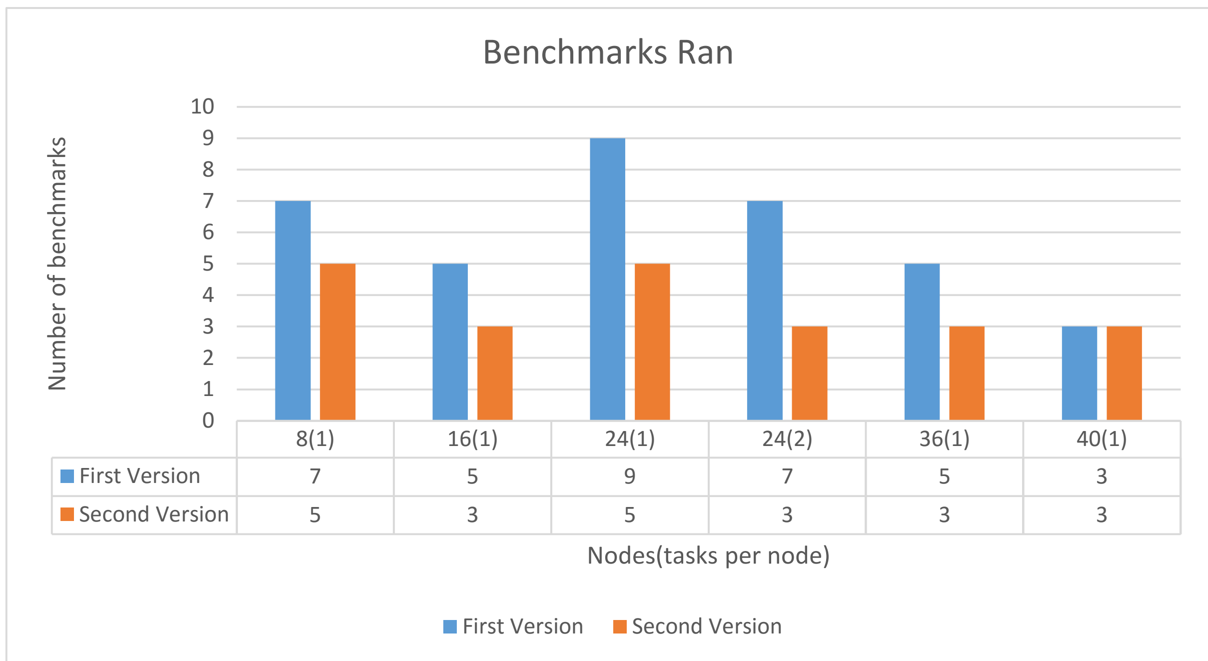


Figure 6.5.1-a Graph showing each versions wall time in seconds against nodes (task per node) used.



	8(1)	16(1)	24(1)	24(2)	36(1)	40(1)
First Version	7	5	9	7	5	3
Second Version	5	3	5	3	3	3

Figure 6.5.1-b Graph showing number of benchmark performed per amount of nodes (tasks per node) used.

The optimised version of the reader is showing significant gains in time taken. Looking at the amount of benchmarks that were ran using the conditions that were defined, it can also be said that the optimised version of the reader was more consistent with its results.

The benchmark performed was limited by the size of the dataset provided and the individual snapshots within, and ideally, could be performed on a larger dataset.

6.5.2 Further Optimisation

Although the optimised version of the reader is already showing large gains in time taken, the most optimal solution for the reader in terms of time and memory would be to, initially, read the snapshots of each particle and store these in memory. As a snapshot is an integer value (2 bytes), this would use $2 * number\ of\ particles$ bytes of memory. This would then be checked against the snapshot specified in the parameter file, reducing this down to $2 * snapshot\ particles$ bytes of data. Now that the reader knows which particles it should read, it selects only the areas of memory that contain the variables for these correct particles from the file, and reads these into the buffer, discarding the original list of correct particles once it has done so. This leaves a total size of

$$\big((4 * snapshot\ particles) * number\ of\ fields\big) + (2 * snapshot\ particles)$$

in bytes of memory at most, before the snapshots have been discarded.

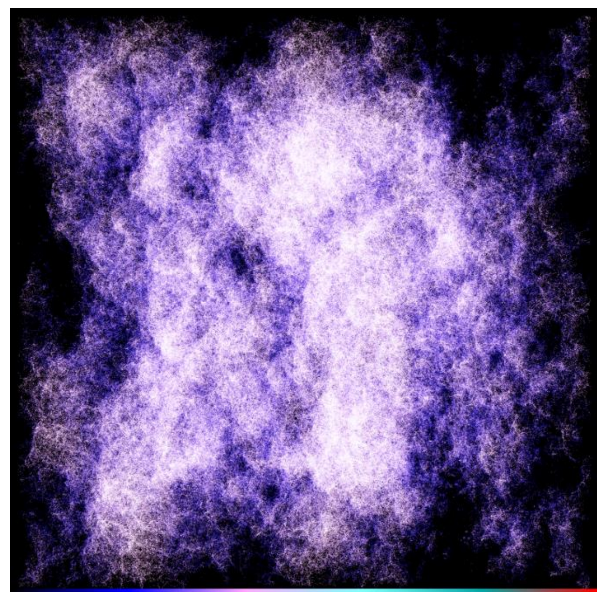


Figure 6.5.2-a Snapshot 28 of the sage-millennium.h5 dataset.

6.6 EVALUATION AND FUTURE WORK

6.6.1 Summery

If we compare the state of the final artefact against the requirements of the project, it would be accurate to say that the project has been a success. The Splotch software on the gSTAR system is successfully reading the data that was provided, and it is also providing an automated way to position the camera. Whilst it is also providing a default value for the brightness and smoothing factor parameters, these could be further improved by employing some kind of heuristic function which takes into account the amount of particles to be rendered. However, as the time scale for this project was short, these functions were not completed in time to be included here.

Working within a HPC environment was not a user friendly experience. All interaction with the system has to be performed via the command line, which can be difficult if a user has no experience with using it beforehand. The experience gained from studying Splotch in the previous Cinder project was of paramount importance for the relatively swift turnaround of the current project. This kind of environment would not be suitable for general use.

6.6.2 Performance

The Splotch software has been written in a way to make itself highly portable, and this has been shown with the ease that it was able to take advantage of the gSTAR systems hardware by utilising OpenMP, Open MPI, and CUDA libraries.

The HDF5 reader adaptation has been attempted to be as optimised as possible within the timeframe of the project. It has already been shown that the performance has been increased after experience with the HDF5 libraries increased, and there is also a suggested solution in place that could further improve memory use and wall time.

6.6.3 Workflow

Workflow for the project was not as optimal as it could have been, especially considering the short timeframe of the project. The majority of the written code was performed locally, then transferred via a secure copy command to the gSTAR system. The software was compiled, and then tested, with the resulting image being transferred back to the local machine for review. This method was not ideal, as it meant that time was unnecessarily spent on these transfer times. This could have been streamlined by editing files directly on the server using a local text editor (Ambros, 2014), or by employing other methods such as automated password entry (Smylers, 2011).

6.6.4 Future Work

The implementation of Splotch onto the gSTAR system is only the first step in the ultimate goal of including it within TAO. In order to move further towards this goal Splotch would have to implement the solutions that have already been discussed in regards to heuristic functions for brightness and smoothing values, as well as possibly implementing the further optimisations outlined for the reader. Any work to be completed after this would have to be discussed with the client, but would most likely involved the ability to define Splotch parameters from a web interface for the purposes of TAO.

7 Conclusions and Evaluation

7.1 Summery

The undertaking of both of the projects that have been discussed in this report have been for the purposes of exploring ways that the Splotch software can be implemented within different environments, and how that can be used to create useful and interesting applications which focus on accessibility. The Cinder project was focused on providing an application that would run within a non-Unix environment for a user that was not necessarily familiar with either a Linux, HPC, or even a scientific environment. Whilst optimisation and performance were important to this project, the main aim was to understand if it were possible to implement Splotch at all, and if so, what could Splotch provide to the Cinder community.

At the conclusion of the project, an application has been created that will allow a non-scientific user to easily use an implementation of the Splotch software that is able to produce real time visualizations. These visualizations are not as accurate as running the full Splotch software, but the application has traded accuracy for ease of use and accessibility. This application is now publicly available on the official Splotch GitHub repository (Ayling, 2016), and has been forwarded to the lead Entropy developer to make use of in the Entropy project.

The project for TAO has demonstrated how the Splotch software can exploit high performance hardware within a HPC environment. The work during this project focused on providing a way for Splotch to be adapted to work on the gSTAR system with datasets provided by the client. The ease of adapting Splotch for this purpose show its strength as portable software. An emphasis on this project was to allow the adaptation of the HDF5 reader to be as optimised as possible. This meant, reducing memory use, whilst also focusing on processing time. This was important as the data provided was larger than the maximum data that could be tested on the Cinder project by around a factor of twenty (9.2GB and 181GB). It also shows how Splotch can be used within a scientific environment by providing a service that will be used by astronomers on TAO.

This repot can hopefully serve as a guide for any projects which aim to implement the Splotch software, and offer some insights into how to undertake that. The Splotch software has proven itself to be a portable but powerful tool that could be implemented in a variety of environments, as has been shown in this report. This report will also serve as the basis for a submission to the 2016 Astronomical Data Analysis Software and Systems (ASASS) conference being held in Trieste, Italy (Istituto Nazionale di Astrofisica Osservatorio Astronomico di Trieste, 2016).

8 TABLE OF FIGURES

Figure 3.4-a Sample renderings in Splotch of small (left), medium (middle) and large (right) data sets. From (Jin, et al., 2010). 8
Figure 3.4-b Execution model of the Splotch code. From (Dykes, 2014). 8
Figure 4.1.1-a Waterfall model flowchart. 11
Figure 4.1.2-a Prototype model flowchart. 11
Figure 4.1.3-a Incremental build model flowchart. 12
Figure 5.4.1-a Application flow of remote connection solution. 15
Figure 5.4.1-b Application flow of solution. 16
Figure 5.6.2-a Application class diagram. 19
Figure 5.6.3-b Examples of loaded parameter files. The leftmost part of the image shows the parameter menu after having loaded the visualization- The upper most values above the separator are real time parameters that affect the renderer. 20
Figure 5.6.3-c Code snippet of loading GUI parameters. 20
Figure 5.6.4-a Code snippet of mapping particle position and colour into the SSBOs. 21
Figure 5.6.5-a Code snippet of the vertex shader. 21
Figure 5.6.5-b The Gaussian blur fragment shader. 22
Figure 5.6.5-c The same image with the blur on. Notice how areas with a higher density of particles appear brighter. 23
Figure 5.6.5-d A cropped image of the blur off. 23
Figure 5.7.3-a Code snippet of the modified function to work on Windows. 25
Figure 5.7.3-b Code snippet of part of the original load function for finding the path of the exe of the application. 25
Figure 5.7.4-a Screenshot of the dataset being displayed as spheres. 26
Figure 5.7.5-a Screenshot of the data being rendered via the initial Cinder renderer. 26
Figure 5.7.6-a Screenshot of the Visual Studio structure. 27
Figure 5.7.6-b Screenshot of the large 9.2GB RAMSES dataset being visualized within the Cinder application. 28
Figure 5.7.8-a Code snippet of dynamically loading the parameters. 29
Figure 5.9.1-a Comparison between a Splotch produced image (left) and the Cinder application's produced image (right) of the same dataset. 32
Figure 5.9.5-a Code snippet of the reader checking the new parameters. 37
Figure 5.9.5-b Reader buffer structure. 37
Figure 5.9.5-c Code snippet of the data being copied into the points vector. 38
Figure 5.9.5-d Code snippet of the reader printing snapshot numbers. 38
Figure 5.9.5-a Snapshot 39 of the mini_millennium.h5 dataset. 39
Figure 5.9.5-b Camera calculation code using the bounding box and new Camera_Calculator class. 39
Figure 6.5.1-a Graph showing each versions wall time in seconds against nodes (task per node) used. 41
Figure 6.5.1-b Graph showing number of benchmark performed per amount of nodes (tasks per node) used. 42
Figure 6.5.2-a Snapshot 28 of the sage-millennium.h5 dataset. 42

9 BIBLIOGRAPHY

Akten, M., Bereza, M., Buni, S., McNamee, D., & Dörfelt, M. (n.d). *DEUTSCHE BANK – HI-RES REALTIME ARTWORKS*. Retrieved from FIELD: https://www.field.io/project/deutsche-bank-hong-kong/

Ambros, G. (2014, November 25). *Editing files remotely via SSH on SublimeText 3*. Retrieved from wrgms.com: https://wrgms.com/editing-files-remotely-via-ssh-on-sublimetext-3/

Android Interfaces and Architecture. (n.d). Retrieved from android: https://source.android.com/devices/

Ayling, E. (2016, March 17). *Cinder Previewer- Windows*. Retrieved from github- splotch: https://github.com/splotchviz/splotch/releases/tag/cinderWindows-v1.0

Becciani, U., Costa, A., Antonuccio-Delogu, V., Caniglia, G., Comparato, M., Gheller, C., . . . Massimino, P. (2010). VisIVO - Integrated Tools and Services for Large-Scale Astrophysical Visualization. *The Publications of the Astronomical Society of the Pacific, 122*, 119-130.

Bernyk, M., Croton, D., Tonini, C., Hodkinson, L., Hassan, A., Garel, T., . . . Hegarty, S. (2016, January 25). The Theoretical Astrophysical Observatory: Cloud-Based Mock Galaxy Catalogues. *arXiv:1403.5270*. Retrieved from http://arxiv.org/pdf/1403.5270v4.pdf

Bersoff, E. H., & Davis, A. M. (1992). Impacts of Life Cycle Models on SOFTWARE CONFIGURATION Management. *Communications of the ACM, 34*(8), 104-118.

Boylan-Kolchin, M., Springel, V., White, S. D., Jenkins, A., & Lemson, G. (2009). Resolving cosmic structure formation with the Millennium-II Simulation. *Monthly Notices of the Royal Astronomical Society, 398*(3), 1150-1164.

chaoticbob. (2015, November 28). *Android and Linux port #1200*. Retrieved from github- Cinder: https://github.com/cinder/Cinder/pull/1200

chaoticbob. (2015, November 28). *RFC: Cinder for Android and Linux*. Retrieved from Cinder Forums: https://forum.libcinder.org/topic/rfc-cinder-for-android-and-linux

Christensen, M. H. (2011, February 19). *GPU versus CPU for pixel graphics*. Retrieved from Syntopia: http://blog.hvidtfeldts.net/index.php/2011/02/gpu-versus-cpu-for-pixel-graphics/

Cider. (2015c). *Cinder 0.9.0*. Retrieved from Cinder: https://libcinder.org/notes/v0.9.0

Cinder. (2015a). *About*. Retrieved from Cinder: https://libcinder.org/about

Cinder. (2015b). *Gallery*. Retrieved from Cinder: https://libcinder.org/gallery

Cinder. (2015d). *iOS Notes*. Retrieved from Cinder: https://libcinder.org/docs/branch/master/guides/ios-notes/index.html

Dolag, K., Borgani, S., Schindler, S., Diaferio, A., & Bykov, A. (2008). Simulation techniques for cosmological simulations. *Space Science Reviews, 4*(1), 229-268. Retrieved from http://arxiv.org/pdf/0801.1023v1.pdf

Dolag, K., Reinecke, M., Gheller, C., & Imboden, S. (2008, December 1). Splotch: visualizing cosmological simulations. *New Journal of Physics, 10*.

Dykes, T. (2014, February 26). Big Data Visualization on the MIC. *Many-Core Seminar Series*. Oxford. Retrieved from https://www.oerc.ox.ac.uk/sites/default/files/uploads/Oxford_BigDataonMIC_260214.pdf

Edwards, P. (2016, March 16). *Science Outreach*. Retrieved March 22, 2016, from Durham University: https://www.dur.ac.uk/science.outreach/

Europeon Space Agency. (2013, March 20). *N° 7–2013: PLANCK REVEALS AN ALMOST PERFECT UNIVERSE*. Retrieved from ESA: http://www.esa.int/For_Media/Press_Releases/Planck_reveals_an_almost_perfect_Universe

Filip, S. (2014, July 15). *An investigation of fast real-time GPU-based image blur algorithms*. Retrieved from Intel Developer Zone: https://software.intel.com/en-us/blogs/2014/07/15/an-investigation-of-fast-real-time-gpu-based-image-blur-algorithms

Furlani, J. L. (1991). Modules: Providing a flexible user environment. *Proceedings of the fifth large installation systems administration conference (LISA V)* (pp. 141-152). Berkeley: USENIX Association. Retrieved March 19, 2016, from http://modules.sourceforge.net/docs/Modules-Paper.pdf

Goldbaum, N. (2011, June 11). *Running your first SPH simulation*. Retrieved from astrobites: http://astrobites.com/2011/06/11/running-your-first-sph-simulation/

Hassan, A. H. (n.d). *About Green II*. Retrieved March 18, 2016, from Centre for Astrophysics and Supercomputing- Supercomputing @ Swinburne: http://supercomputing.swin.edu.au/about-green-ii/

Heitmann, K., Frontiere, N., Sewell, C., Habib, S., Pope, A., Finkel, H., . . . Bhattacharya, S. (2015). THE Q CONTINUUM SIMULATION: HARNESSING the POWER of GPU ACCELERATED SUPERCOMPUTERS. *Astrophysical Journal, Supplement Series, 219*(2). Retrieved from http://arxiv.org/pdf/1411.3396v1.pdf

Hodgin, R. (2013, June 24). *Boil Up: Realtime Feeding Frenzy*. Retrieved from Robert Hodgin: http://roberthodgin.com/portfolio/work/boil-up/

Hodgin, R. (2013, November 03). *Oculus Rift: Gravity*. Retrieved from Robert Hodgin: http://roberthodgin.com/portfolio/work/oculus-rift-gravity/

Hurlbut, J. (2014, December 1). *ORBIGON*. Retrieved from jameshurlbut: http://jameshurlbut.net/wp/portfolio/orbigon/

Ishiyama, T., Enoki, M., Kobayashi, M., Makiya, R., Nagashima, M., & Oogi, T. (2015). The v2GC simulations: Quantifying the dark side of the universe in the Planck cosmology. *Publications of the Astronomical Society of Japan, 4*(67). Retrieved from http://arxiv.org/pdf/1412.2860v2.pdf

Istituto Nazionale di Astrofisica Osservatorio Astronomico di Trieste. (2016). *ADASS 2016 - Trieste*. Retrieved from ADASS XXVI: http://www.adass2016.inaf.it/

JeGX. (2014, July 04). *GPU Buffers: Introduction to OpenGL 4.3 Shader Storage Buffers Objects*. Retrieved from Geeks3D: http://www.geeks3d.com/20140704/tutorial-introduction-to-opengl-4-3-shader-storage-buffers-objects-ssbo-demo/

Jin, Z., Krokos, M., Rivi, M., Gheller, C., Dolag, K., & Reinecke, M. (2010, May). High-performance astrophysical visualization using Splotch. *Procedia Computer Science, 1*(1), 1775-1784.

Jorgensen, P. C. (2013). *Software testing : a craftman's approach* (4th ed.). Boca Raton: Auerbach Publications.

Kepler, G. (2015, November 20). *Animated Gifs in Cinder*. Retrieved from The Grego: http://www.thegrego.com/2015/11/20/animated-gifs-in-cinder/

Kitware. (n.d). *Overview*. Retrieved from Paraview: http://www.paraview.org/overview/

Klypin, A., Trujillo-Gomez, S., & Primack, J. (2011, October 20). Dark Matter Halos in The Standard Cosmological Model: Results from The Bolshoi Simulation. *Astrophysical Journal, 740*(2), 1-17.

Klypin, A., Yepes, G., Gottlober, S., Prada, F., & Hess, S. (2016). MultiDark simulations: the story of dark matter halo concentrations and density profiles. *MNRAS Working Paper, 4*(457), 4340-4359.

kollision. (2011, November 29). *AUDI URBAN FUTURE*. Retrieved from kollision: http://kollision.dk/en/urbanfuture

Kosara, R. (2007). Visualization Criticism – The Missing Link Between Information Visualization and Art. *Information Visualization, 2007. IV '07. 11th International Conference* (pp. 631-636). Zurich: IEEE.

Lawrence Livermore National Laboratory. (n.d). *VisIt*. Retrieved from Weapons and Complex Integration: https://wci.llnl.gov/simulation/computer-codes/visit

Lengeling, T. S., & Castro, G. (2014). *Aether*. Retrieved from Works and Portfolio of Thomas Sanchez Lengeling: http://codigogenerativo.com/works/aether/

Lerner, L. (2015, October 29). *Researchers model birth of universe in one of largest cosmological simulations ever run*. Retrieved from Argonne National Laboratory: http://www.anl.gov/articles/researchers-model-birth-universe-one-largest-cosmological-simulations-ever-run

Markovic, K. (2016, February 26). *Entropy: Live Astronomy Documentary Meets an Electronic Music Performance*. Retrieved March 03, 2016, from Research Councils UK: http://gtr.rcuk.ac.uk/projects?ref=ST/N000293/1

Microsoft. (2016a). *Optimizing HLSL Shaders*. Retrieved from Windows Dev Center: https://msdn.microsoft.com/en-us/library/windows/desktop/cc627119(v=vs.85).aspx

Microsoft. (2016b). *OpenMP in Visual C++*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/tt15eb9t.aspx

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing* (3rd ed.). Hoboken, New Jersey: John Wiley & Sons.

Net Applications. (2016, February). *Desktop Operating System Market Share*. Retrieved March 04, 2016, from NetMarketShare: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0

Popa, B. (2015, January 10). *Microsoft Explains Why Windows 10 32-Bit Is Still Needed*. Retrieved from Softpedia: http://news.softpedia.com/news/Microsoft-Explains-Why-Windows-10-32-Bit-Is-Still-Needed-469563.shtml

Pressman, R. S. (2005). *Software engineering : a practitioner's approach.* Dubuque, Iowa: McGraw-Hill.

Processing Foundation. (n.da). *Cover*. Retrieved February 07, 2016, from Processing: https://processing.org/

Processing Foundation. (n.db). *Exhibition*. Retrieved February 07, 2016, from Processing: https://libcinder.org/gallery

Protalinski, E. (2015, October 01). *Windows 10 grabs 6.63% market share, Linux finally passes Windows Vista*. Retrieved from VentureBeat: http://venturebeat.com/2015/10/01/windows-10-grabs-6-63-market-share-linux-finally-passes-windows-vista/

Rákos, D. (2010, September 7). *Efficient Gaussian blur with linear sampling*. Retrieved from RasterGrid: http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/

Rayne, C. (2014, May 4). *Why Modern GPU's Perform Faster Than CPU's & Good At Parallel Computing Part 1*. Retrieved from Red Gaming Tech: http://www.redgamingtech.com/why-modern-gpus-perform-faster-than-cpus-good-at-parallel-computing-part-1/

Reas, C. (2007). *Processing : a programming handbook for visual designers and artists.* Cambridge: MIT Press.

Rijnieks, K. (2013). *Cinder – Begin Creative Coding.* Birmingham: Packt Publishing Ltd.

Rivi, M., Dykes, T., Krokos, M., & Dolag, K. (2014, July). GPU accelerated particle visualization with Splotch. *Astronomy and Computing, 5*, 9-18.

Royal Institution. (n.d). *About the CHRISTMAS LECTURES*. Retrieved from The Royal Institution: http://www.rigb.org/christmas-lectures/about

Rutter, D. (2012, May 24). *Ask Dan: What's with the 3Gb memory barrier?* Retrieved from Dan's Data: http://www.dansdata.com/askdan00015.htm

Samaroo, A., Thompson, G., & Hambling, B. (2015). *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide* (3rd ed.). Swindon: BCS.

Sample, I. (2015, September 1). *Christmas lectures to explore challenges of space flight*. Retrieved from The Guardian: https://www.theguardian.com/science/2015/sep/01/christmas-lectures-space-flight-royal-institution-kevin-fong

Schroeder, W., Martin, K., & Lorensen, B. (1996). *The visualization toolkit : an object-oriented approach to 3D graphics.* Upper Saddle River: Prentice Hall.

Sciacca, E., Bandieramonte, M., Becciani, U., Costa, A., Massimino, P., Pistagna, C., . . . Petta, C. (2013). VisIVO workflow-oriented science gateway for astrophysical visualization. *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (pp. 164-171). Belfast: IEEE.

Selikoff, N. (2013). *Beautiful Chaos*. Retrieved from Nathan Selikoff: http://nathanselikoff.com/works/beautiful-chaos

Shu, F. (1991). *The Physics of Astrophysics: Volume I Radiation.* New York: University Science Books.

Smylers. (2011, August 16). *SSH Can Do That? Productivity Tips for Working with Remote Servers*. Retrieved from blogs.perl.org: http://blogs.perl.org/users/smylers/2011/08/ssh-productivity-tips.html

Song, G., Zheng, Y., & Shen, H. (2006). Paraview-based collaborative visualization for the grid. *Lecture Notes in Computer Science, 3842*, 819-826.

Springel, V., White, S. D., Jenkins, A., Frenk, C. S., Yoshida, N., Gao, L., . . . Pearce, F. (2005, June 2). Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature, 435*(7), 629-636.

Springel, V., Yoshida, N., & White, S. D. (2001). GADGET: a code for collisionless and gasdynamical cosmological simulations. *New Astronomy, 6*(2), 79-117.

Stephens, T. (2011, September 29). *Scientists release most accurate simulation of the universe to date*. Retrieved from University of California, Santa Cruz- Newscenter: http://news.ucsc.edu/2011/09/bolshoi-simulation.html

Teyssier, R. (2002). Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES. *Astronomy and Astrophysics, 385*, 337-364. Retrieved from University of Zurich- Institute for Computational Science: http://www.ics.uzh.ch/~teyssier/ramses/RAMSES.html

The HDF Group. (2011, June 22). *WHY HDF?* Retrieved from The HDF Group: https://www.hdfgroup.org/why_hdf/

The HDF Group. (2015, May 20). *HDF5 TUTORIAL: ADVANCED TOPICS COMPOUND DATATYPES*. Retrieved from The HDF Group: https://www.hdfgroup.org/HDF5/Tutor/compound.html

The Illustris Collaboration. (2015). *About- Project Description*. Retrieved from Illustris Project: http://www.illustris-project.org/about/#public-two

Theoretical Astrophysical Observatory (TAO). (n.d). *About*. Retrieved March 18, 2016, from Theoretical Astrophysical Observatory: https://tao.asvo.org.au/tao/about/

Upswell. (n.d). *New Perspectives on Human Rights*. Retrieved February 07, 2016, from Upswell: http://www.hello-upswell.com/project/canadian-museum-human-rights/

Vogelsberger, M., Genel, S., Springel, V., Torrey, P., Sijacki, D., Xu, D., . . . Hernquist, L. (2014, October 21). Introducing the Illustris project: the evolution of galaxy populations across cosmic time. *MONTHLY NOTICES OF THE ROYAL ASTRONOMICAL SOCIETY, 2*(444), 1518-1547.

Wall, M. (2012, September 13). *Neil Armstrong Inspired Canadian Astronaut's Giant Leap*. Retrieved from Space: http://www.space.com/17593-neil-armstrong-chris-hadfield-canadian-astronaut.html

Weinstein, J., & Jaques, T. (2010). *Achieving Project Management Success in the Federal Government.* Vienna, Washington: Management Concepts, Inc.

Woodring, J., Heitmann, K., Ahrens, J., Fasel, P., Hsu, C.-H., Habib, S., & Pope, A. (2011). Analyzing and Visualizing Cosmological Simulations with ParaView. *The Astrophysical Journal Supplement Series, 195*.

10 Appendix 1 - Client Proposal Document

Integrating Splotch into a Cinder application

Cinder

Cinder is an open source C++ library, with support for OpenGL. It aims to give advance visualization abilities whilst being easy to learn. It has official support for Windows, OSX, iOS, and WinRT. It depends heavily on system libraries to work, which means that, in its current form, there is no official Linux support (although development is being done in this area).

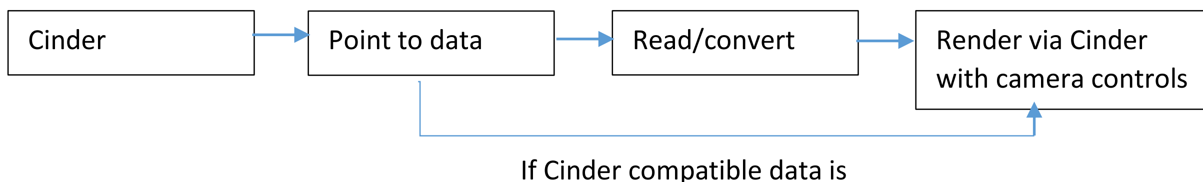
Splotch

Splotch is a software tool used to visualize cosmological data sets. It is written in C++ and designed to be used within a Linux environment. It supports various types of data including Gadget, and Ramses.

Potential Solutions

Creating a Splotch-Like plugin for Cinder

This would involve using Splotch's data readers for the purpose of extracting the data in a way which would allow Cinder to use and visualize it within its application. This would be similar to how the previewer works but would use Cinder as an interface instead to explore and possibly add additional information and interactivity to the visualization for the purposes of the Entropy project. This would allow a "splotch-like" program to be run on Windows and OSX.



If Cinder compatible data is

Using Cinder as an interface to run Splotch

Cinder would be used to set the parameters for the data which would then be given to Splotch (most likely remotely so Splotch can be running on Linux). Splotch would then pass the resulting image back to Cinder which would then be used to explore the visualization. As Cinder cannot currently run on Linux in a form that would be useable, the data would have to already be on the target machine (which would be running Linux) in order to eliminate the transfer time of the potentially very large files.

11 Appendix 2- Splotch Manual First Draft

Splotch Manual

For version 6.0

About

Splotch is a ray tracer primarily for visualization of SPH simulation output (e.g. Gadget data). Various types of data are supported, including AMR outputs such as RAMSES and Enzo data.

Current Status

Setup

1. Download the latest version of Splotch.
2. Unpack the .tar file.
3. Edit the makefile.
4. Run make from the terminal.
5. Run the Splotch application with a parameter file. EG "./Splotch6-generic demo.par"

If using the previewer, please see the Previewer Readme in the docs/ folder for instructions on how to setup Splotch.

Makefile

The makefile is commented in a way to allow you to easily edit Splotch to suit your needs. Various options are at the top of the file that you can uncomment depending on what features you require. If you are using a standard linux distro please uncomment "generic" as your system type. Otherwise uncomment the appropriate type.

To add a new system type to the makefile, you will need to know the path to the relevant libraries for the features you wish to use. E.g., MPI, CUDA, HDF5.

Parameter File

A few parameter files are included with the Splotch download to serve as examples. Listed below are some of the more important parameters.

Parameter	Type	Usage	Notes
infile	String	Path to data file	
simtype	Integer	Type of the data file	0,1- Plain binary file, 2- Gadget2, 3- Enzo, 4- Millenium, 5- Mpiio, 6- Mesh reader, 7- Hdf5, 8- Gadget HDF5, 10- VisiVO, 11- Tipsy, 13- RAMSES, 14- Bonsai, 15- Ascii, 16-Fits.
ptypes	Integer	Number of types of particle data to be read.	

pictype	Integer	File output format	0- TGA uncompressed, 1- PPM ASCII, 2- PPM binary, 3- TGA runlength-compressed
outfile	String	Name of output file	
xres/yres	Integer	Specify the output resolution of the image	Generally setting xres is enough, as yres will match it unless specified otherwise.
colorbar	Booleon	Include the color palette color bar beneath the image.	If you have more than one palette for the image, it will include both.
paletteX (where X = the particle type. EG palette0 for the first type, palette1 for the second etc)	String	Path to the appropriate palette file to be used.	Path should generally be “palettes/” followed by the chosen palette EG “palettes/OldSplotch.pal”
brightnessX (where X= the particle type.)	Float	Set the brightness value of the data.	e.g. “brightness0 = 1.2”
camera_x camera_y camera_z	Float	The position of the camera defined in XYZ coordinates.	

lookat_x lookat_y lookat_z	Float	The position that the camera will point at defined in XYZ coordinates.	
sky_x sky_y sky_z	Float	The up vector for the camera.	Generally 0, 0 ,1.
fov	Float	The field of view value for the camera.	

More information can be found here.

Notes

- The default path for the saved image is the Splotch directory.

12 Appendix 3- Test Cases

12.1 First Prototype

F1	System::setup
Scope	Function creates a sphere object. Function orients the camera.
Result	Test Passed 17/10/2015

F2	System::draw
Scope	Function draws spheres within 3D space.
Result	Test Passed 17/10/2015

F3	System::readFile
Scope	Function opens and reads a file into a vector.
Result	Resolved an issue related to out of range vector calls. Test Passed 18/10/2015

12.2 Second Prototype

A1	Application Compilable with Splotch files
Scope	Application is able to compile with the Splotch header files included.
Result	Fixed multiple instances of errors related to incorrect include paths due to file paths differing from the original Splotch file directories. Test Passed 15/12/2015

12.3 Third Prototype

F1	System::setup
Scope	Function is creating a string with the correct file path to a parameter file, and pass this into SimpleGUI::Load.
Result	Test Passed 05/01/2016

F4	SimpleGUI::Load
Scope	Function is extracting the path of the exe. Function passes the parameter path to Parameter::Load.
Result	Identified and solved issues which were a result of the function expect a Unix path. Function was modified to work with the Windows file system. Test Passed 05/01/2016

M1	Previewer is able to load a parameter file.
Scope	Application is able to load a parameter file.
Result	Test Passed 06/01/2016

M2	Previewer Files are using Splotch to load particles
Scope	Application is able to provide a parameter file path and have the particles be loaded.
Result	Identified a porting problem within walltime_c.c which needed the equivalent Windows libraries for some functions. Commented out functions which included unsupported OpenGL function calls. Application suffers from an out of range memcopy exception when reading the Ramses data. Test Failed 07/01/2016

12.4 Fourth Prototype

M1	Previewer Files are using Splotch to load particles
Scope	Application is able to provide a parameter file path and have the particles be loaded.
Result	Identified and fixed an issue relating to Splotch not being able to find the palette files. Application is able to load the smaller Gadget files. Test Passed 14/01/2016

M2	Application is able to render loaded particles within Cinder
Scope	Application is able to take the particle vector provided by Splotch and use this to render spheres at these positions.
Result	Application can only render particles at a factor of 100. Test Passed 14/01/2016

12.5 Fifth Prototype

F7	CinderRender::Draw
Scope	Function is drawing the FBO to screen.
Result	Test Passed 23/01/2016

S1	Particles.vert
Scope	Shader is mapping the colour of the particle. Shader is mapping particles to a quad. Shader is calculating the saturation, contrast, and brightness via a function.
Result	Issues fixed relating to vector4 to vector3 conversion. Test Passed 23/01/2016

S2	Particles.frag
Scope	Shader is discarding pixels correctly.
Result	Test Passed 23/01/2016

F2	System::draw
Scope	Function is calling CinderRender::Draw
Result	Test Passed 23/01/2016

12.6 Sixth Prototype

F5	CinderRender::Load
Scope	Function is finding shader files and loading them into a shader object. Function sets a clipping plane and camera lookat. Function creates SSBOs. Function creates an Index VBO. Function loads particle positions into a SSBO. Function is creating blur FBOs.
Result	Test Passed 25/01/2016

F7	CinderRender::Draw
Scope	Function is drawing the FBO to screen. Function is drawing the blur FBO over the original FBO using alpha blending.
Result	Test Passed 25/01/2016

F6	CinderRender::RenderToFBO
Scope	Function is correctly scoping and drawing to the correct FBOs. Function is passing correct variables to the shaders.
Result	Identified and fixed issue relating blur FBOs not correctly mapping to the screen size. Test Passed 25/01/2016

S3	blur_pass.vert
Scope	Shader is passing the texture coordinates through to the fragment shader.
Result	Issues fixed relating to different GLSL version keywords. Test Passed 25/01/2016

S4	blur.frag
Scope	Shader is applying the Gaussian blur correctly to each particle.
Result	Test Passed 25/01/2016

12.7 Seventh Prototype

A2	Application is loading as a 64bit program.
Scope	Application is correctly linking appropriate Cinder libraries and compiling as a 64bit program.
Result	Fixed issues with incorrect linking where compiler was still linking to x86 libraries. Test Passed 16/02/2016

A1	Application Compilable with Splotch files
Scope	Application is able to compile with the Splotch header files included.
Result	Test Passed 16/02/2016

M2	Previewer Files are using Splotch to load particles
Scope	Application is able to provide a parameter file path and have the particles be loaded.
Result	Application is now able to load the large Ramses dataset. Test Passed 16/02/2016

12.8 Eight Prototype

F5	CinderRender::Load
Scope	Function is finding shader files and loading them into a shader object. Function sets a clipping plane and camera lookat. Function creates SSBOs. Function creates an Index VBO. Function loads particle positions into a SSBO. Function is creating blur FBOs. Function is initialising blurStrength variable.
Result	Test Passed 22/02/2016

F7	CinderRender::Draw
Scope	Function is drawing the FBO to screen. Function is drawing the blur FBO over the original FBO using alpha blending, when a blur Boolean is toggled. Function is passing variables to shaders.
Result	Test Passed 23/02/2016

F6	CinderRender::RenderToFBO
Scope	Function is correctly scoping and drawing to the correct FBOs, when a blur Boolean is toggled. Function is passing correct variables to the shaders.
Result	Identified and fixed issue relating blur FBOs not correctly mapping to the screen size. Test Passed 23/02/2016

F8	System::createParmas
Scope	Function is correctly creating parameters on screen for rendering tweaks.
Result	Test Passed 23/02/2016

F2	System::draw
Scope	Function is calling CinderRender::Draw. Function is drawing the parameter box.
Result	Test Passed 01/03/2016

F9	System::toggleBlur()
Scope	Function is toggling the blurOn Boolean.
Result	Test Passed 23/02/2016

S2	Particles.frag
Scope	Shader is discarding pixels correctly.
Result	Test Passed 23/02/2016

S3	blur_pass.vert
Scope	Shader is passing the texture coordinates through to the fragment shader.
Result	Issues fixed relating to different GLSL version keywords. Test Passed 23/02/2016

S4	blur.frag
Scope	Shader is applying the Gaussian blur correctly to each particle. Shader is modifying the colour according to the colour modifier.
Result	Test Passed 23/02/2016

S1	Particles.vert
Scope	Shader is mapping the colour of the particle. Shader is modifying the colour according to the brightness modifier. Shader is mapping particles to a quad. Shader is calculating the saturation, contrast, and brightness via a function.
Result	Test Passed 23/02/2016

12.9 Ninth Prototype

S1	Particles.vert
Scope	Shader is mapping the colour of the particle. Shader is modifying the colour according to the brightness modifier. Shader is mapping particles to a quad. Shader is calculating the saturation, contrast, and brightness via a function. Shader is passing the particle size variable to the fragment shader.
Result	Test Passed 01/03/2016

S2	Particles.frag
Scope	Shader is discarding pixels correctly according to the particle size modifier.
Result	Test Passed 01/03/2016

F10	SpotchCinder::prepareSettings
Scope	Function is setting correct window resolution. Function is creating a vector to define allowed file types that can be selected. Function is creating an explorer window to get the path to the parameter file. Function is storing this path if the path is valid.
Result	Identified and fixed issues with how Cinder function expected the its arguments. Test Passed 01/03/2016

F1	SplotchCinder::setup
Scope	Function is passing the parameter file path to be loaded. Function is creating the box that holds the Cinder parameters on screen.
Result	Test Passed 01/03/2016

F8	SplotchCinder::createParmas
Scope	Function is correctly creating parameters on screen for rendering tweaks. Function is creating and storing reference to the parameter file. Function is creating a two dimensional dynamic array the size of the amount of parameters in the parameter file. Function is binding SplotchCinder::loadMainScene to a button. Function is storing parameter names and their variables in the array. Function is creating a Cinder parameter for each parameter in the array. Function is binding SplotchCinder::writeToParameterFile to a button.
Result	Identified and fixed issues relating to dynamic memory allocation. Test Passed 01/03/2016

F11	SplotchCinder::loadMainScene
Scope	Function is getting loaded particles. Function is passing loaded particles to CinderRender::Load. Function is clearing the parameter box. Function is deleting the parameter array. Function is creating the new parameters. Function is toggling the drawMainScene Boolean.
Result	Test Passed 01/03/2016

F12	SplotchCinder::writeToParameterFile
Scope	Function is opening the parameter file. Function is writing back to the parameter file line by line. Function is closing the file. Function is deleting the parameter box. Function is deleting the parameter array. Function is reloading the parameter file.
Result	Fixed issued where the wrong values would be written to parameters. Test Passed 01/03/2016

F2	SplotchCinder::draw
Scope	Function is calling CinderRender::Draw or clearing the screen dependant on the drawMainScene Boolean. Function is drawing the parameter box.
Result	Test Passed 01/03/2016

12.10 Not Tested

A3	Application can load HDF5 Files.
Scope	Application can target and load HDF5 files.
Result	Not Tested.

A4	Application can load Enzo Files.
Scope	Application can target and load Enzo files.
Result	Not Tested.

A5	Application can load Tipsy Files.
Scope	Application can target and load Tipsy files.
Result	Not Tested.