furakutaru

A Hardware-Accelerated System for Rendering Fractals by means of Automatic Calculation

Oliver Dixon

under the supervision of
David Clapham

フラクタル

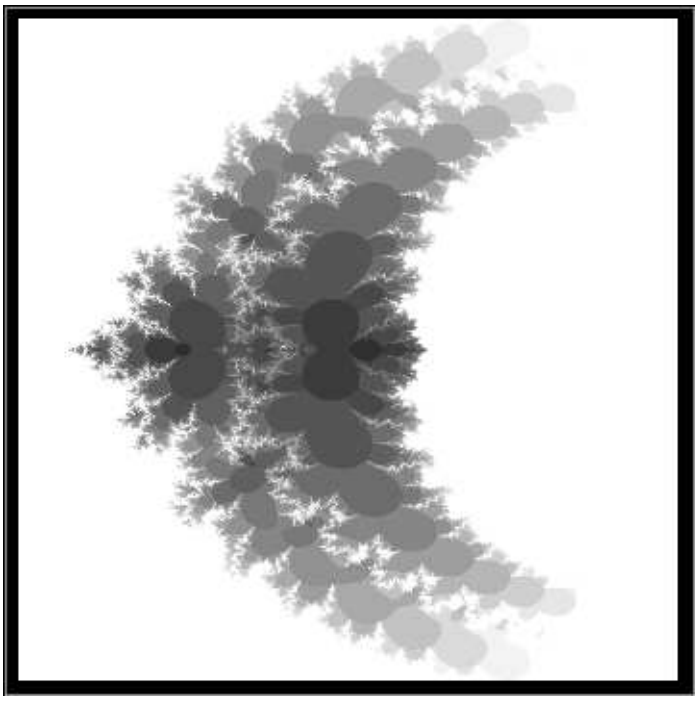
This report is presented as partial fulfilment of the *AQA A-Level Computer Science* qualification.

Department of A-Levels
Wakefield College
United Kingdom

Summer, 2019–Spring, 2020

Dedicated to the Gamma Function
and Related Articles

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} \mathrm{d}x \qquad \Re(z) > 0$$
$$= (z-1)! \qquad (\text{for } z \in \mathbb{Z}^+)$$



A High-Precision Fractal Iterating from $\Gamma(z_n)$

Contents

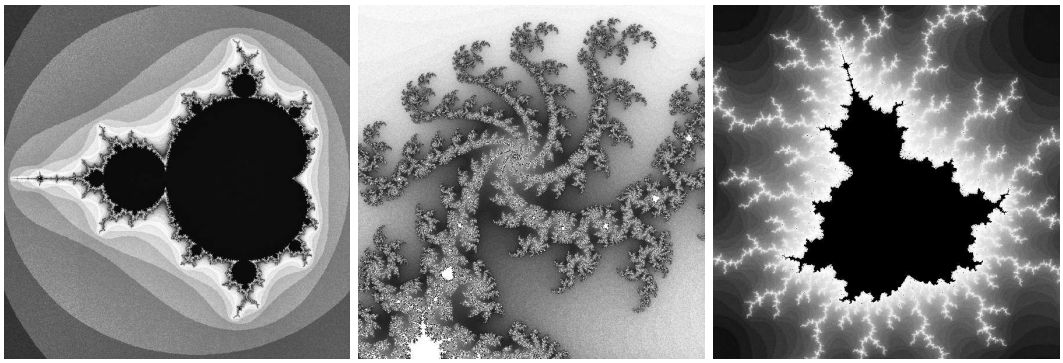
Stage I

Analysis

Section 1

Introduction to Fractals

From the layman's perspective, a fractal is a beautiful reflection of nature [Man82] which can be subject to an infinite zoom, revealing intricate and somewhat mysterious patterns. The most popular of these is known as a "Mandelbrot Set", visualised below.

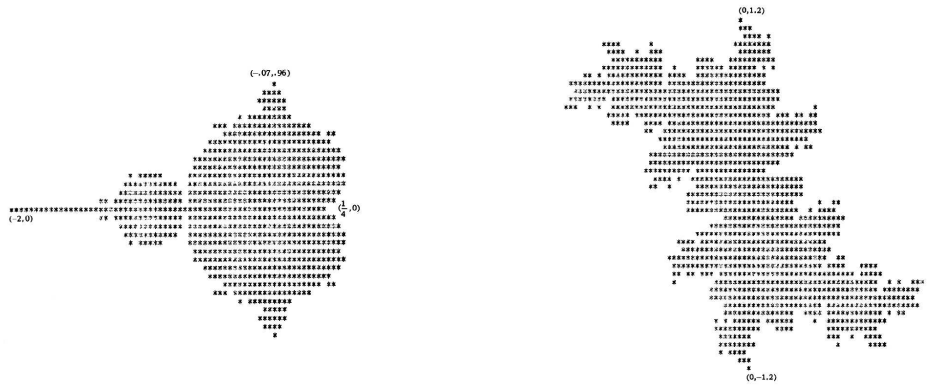


(a) The Initial View (b) Slightly Zoomed (c) Recurrence of (a)

Figure 1.1: The Mandelbrot Set at Three Levels of Zoom

From a mathematicians viewpoint however, fractals are far more impressive and beautiful subsets of the Euclidean space which possess the property of self-similarity [Hut81], hence creating the colloquially named effect of "infinite zoom". The systematic study of these irregular and complex shapes was pioneered by Douady and Hubbard in 1981 to 1985, naming their "Mandelbrot Set" after Benoit Mandelbrot, who had included an image of the shape in his 1980 chapter [Man80]. However, the very first visualisation of what is now known as the Mandelbrot Set was published in 1978 by Brooks and Matelski, whom were considering related sets.

The images were plotted using only the functions available to the phototypesetter. The Mandelbrot Set is plotted within the bounds of $-2 \leq \Re(C) \leq \frac{1}{4}$; these are the intersection points with the real axis [Met94]. The Julia Set is plotted with the constant $C = \frac{1}{10} + \frac{3}{5}i$. These ideas and concepts will be covered in detail throughout the later sections.



(a) The Mandelbrot Set: "*The set of C 's such that $f(z) = z^2 + C$ has a stable periodic orbit.*"

(b) The Julia Set: "*The set E for $f(z) = z^2 + C$ with $C = 0.1 + 0.6i$.*"

Figure 1.2: The Mandelbrot and Julia Sets with their original captions [BM81].

The formal definition of a generic fractal is beyond the scope of this computing project, however it would be advantageous to have a semi-formal understanding of the more specific fractals which *furakutaru* deals with: the Julia and Mandelbrot Sets.

1.1 Mathematical Overview: The Mandelbrot Set

When formally defining the Mandelbrot Set, consider the iterative function

$$z_{n+1} = (z_n)^2 + c \qquad c \in \mathbb{C} \tag{1.1}$$

where c is every point to be coloured on the complex plane and z is initialised to 0 (i.e. $z_0 = 0$) [Dou86]. Defining $\mathcal{M}$ to be the Mandelbrot Set, the point c is in the set if, and only if, $|z_n|$ remains bounded for $\forall n \in \mathbb{N}$. This can be formally described as

$$c \in \mathcal{M} \iff \limsup_{n\to\infty} |z_{n+1}| \leq d \tag{1.2}$$

where $d = 2$. In an effort to move towards iterative function composition, now consider the function

$$P_c(z) = z^2 + c \qquad c \in \mathbb{C} \tag{1.3}$$

and let $P_c^n(z)$ denote $P_c(z)$ applied to itself n times, where $n \in \mathbb{N}$.

Additionally, let d be the boundary condition, which can be thought of as the "cut-off" radius for which a point c no longer belongs to $\mathcal{M}$. The Mandelbrot Set can now be formally defined using set notation [Lei90].

$$\mathcal{M} = \{c \in \mathbb{C} : \exists d \in \mathbb{R}^+, \forall n \in \mathbb{N}, |P_n^c(z_0)| \leq d\} \tag{1.4}$$

$d = 2$ is selected as the radius, and any value exceeding this will tend to infinity [Ava09]. This generates the shapes seen in Figure 1.1, hence simplifying the general definition at Equation 1.4.

$$\mathcal{M} = \{c \in \mathbb{C} : \forall n \in \mathbb{C}, |P_n^c(0)| \leq 2\} \tag{1.5}$$

With regards to computer graphics, in the event that $c \in \mathcal{M}$, it is convention that it is coloured black, however any distinct colour would be valid. If the inverse is true, the colour of the pixel is determined by the rate at which it tends to infinity. This achieved by counting the iterations which are required to determine that the sequence tends to infinity; the algorithm must therefore define a maximum number of iterations, in the interests of avoiding an infinite loop. If this maximum is reached and the converge test is inconclusive (i.e. the subsequent points do not have a modulus less than or equal to d), it is assumed that $c \in \mathcal{M}$, and hence is coloured black. The algorithmic implementation and optimisations, of which there are many, shall be discussed in-detail during a later section.

1.2 Mathematical Overview: The Julia Set

The Julia Set (and Fatou Set) are sets defined by functions that act regularly and chaotically respectively. As with the general fractal, a rigorous understanding of the Sets and the terms "regular" and "chaotic" is out-of-scope for this project [Fel12], however it is important to be aware of the Julia Set due to its intrinsic relationship to the Mandelbrot Set.

The most popular and visually appealing system of Julia Sets is given by the infinite collection of complex polynomials[1] of the second degree. These behave very similarly to the Mandelbrot Set and hence form a beautiful connection, to the extent which they have the same iterative function and colouring methods (see Equation 1.1 in § 1.1). As opposed to the Mandelbrot Set however, c is an arbitrary complex constant and each point on the complex plane is taken, in-turn, to be the initial condition of z_0.

Due to the freedom provided by the constant c, a wide variety of unique shapes can be created using the same set of algorithms.

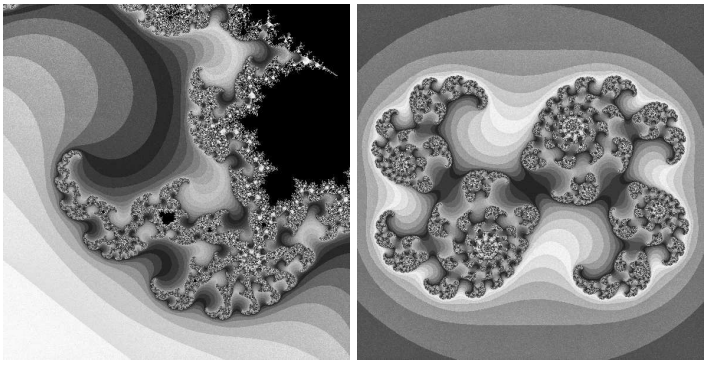
1.3 The Mathematical and Visual Connection Between the Mandelbrot and Julia Sets

These sets are very much interrelated, to the point which they can share certain patterns in certain locations on the plane. Recall that the Mandelbrot Set selects the initial condition, z_0, to be zero, while it allows c to be any point on the complex plane. Simultaneously, the Julia Set is quite

[1] A complex polynomial is a polynomial in which the coefficients and variables are complex.

the inverse, such that it allows c to be set to a constant, while stating that z_0 must be any point on the complex plane.

Suppose that once a point is chosen for the Mandelbrot Set, a Julia Set is rendered using the point for the constant c; this can be referred to as the "seed" of the Julia Set. Overall, the two shapes are incredibly distinct, however localised regions exhibit very similar visual elements [Lei90]; some fractal-rendering tools such as *G.N.U. XaoS*[2] even enable the user to render a real-time Julia Set in correspondence with the selection point of the Mandelbrot.



(a) The Mandelbrot Set, centred at $0.346 + 0.06i$

(b) The Julia Set, using the "seed" $c = 0.346 + 0.06i$

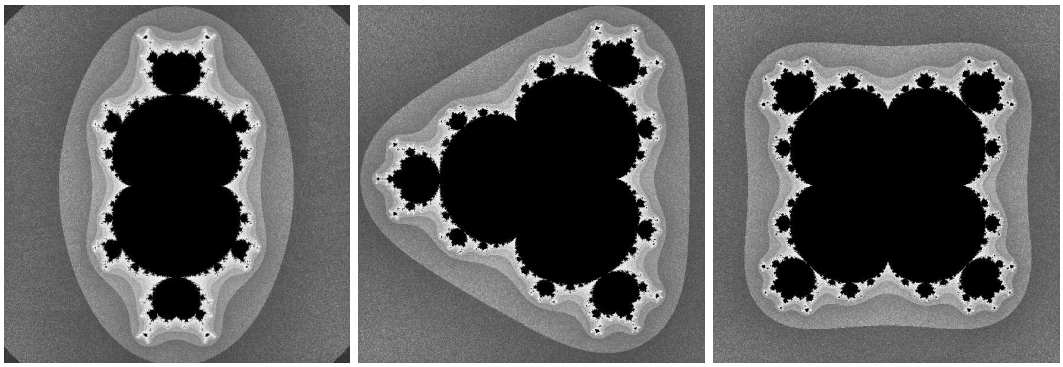
Figure 1.3: The Mandelbrot and Julia Sets focused upon the same point

1.4 "Multi-Brot" and "Multi-Julia" Sets

Equation 1.1 in § 1.1 stated that the iterative function for a Mandelbrot Set, and by extension Julia Set, was a quadratic. While this is the most well-known type, "multi-Brot" and "multi-Julia" sets also exist, such that an arbitrary degree p is used[3]. "Multi-Brot" sets can be denoted as $\mathcal{M}_p$ [Jun02].

$$z_{n+1} = (z_n)^p + c \qquad c \in \mathbb{C},\ p \in \mathbb{Z}_{\geq\ 2} \tag{1.6}$$

Upon plotting these sets on the complex plane, it can be seen that the value of p denotes the number of "branches". More formally, this is the statement that $\mathcal{M}_p$ has a rotational symmetry of $p-1$-fold [Sch99].



(a) $p = 3$ (b) $p = 4$ (c) $p = 5$

Figure 1.4: The sets $\mathcal{M}_{p\in\{3,4,5\}}$ exhibiting rotational symmetry of $p-1$

Aside from the general iterative formula (Equation 1.6), "multi-Brot" and "multi-Julia" sets are subject to identical mathematics as their "$p = 2$"-counterparts. Rendering algorithms prove to be slightly different in the general case, however high-quality implementations should be built with the knowledge of arbitrary degrees.

[2] *G.N.U. XaoS* is a popular fractal-rendering software package: `http://matek.hu/xaos/doku.php`

[3] It is possible to have $p \in \mathbb{R}$ cases, however fractional and negative degrees are beyond the scope of this essay. *furakutaru* should, however, have the ability to render fractals of a negative degree.

Section 2

Rendering Solutions

2.1 Analysis of Current Solution

Due to their extreme popularity amongst mathematicians and non-mathematicians alike, many software packages for generating fractals currently exist. The current and most prevalent software package is *Ultra Fractal*, and although fully featured, is not accessible for Linux users[1], nor people who do not wish to pay 99 Euros [Dev18]. This immediately excludes many amateurs and educational institutions. Additionally, it could be argued that *Ultra Fractal* does not support the advancement of the software development industry and fractal education due to its proprietary and commercial licensing, which further mandates the creation of a small, flexible, free, and open-source alternative.

There are many alternative packages to *Ultra Fractal*, however they all suffer from various weaknesses which would hinder access of the products to large groups of people, whether that be via high costs or platform-exclusivity. Such Windows-exclusive packages include *Apophysis*, *QuickMAN*, *ChaosPro*, *Chaoscope*, and *Sterling2*, and commercial packages include *Frax* and *Fraksl*.

2.2 Proposed Alternative Solution: *furakutaru*

As mentioned in § 2.1, the various shortcomings of existing packages such as *Ultra Fractal* mandate the creation of a lightweight alternative.

2.2.1 Identification of End-Users

The primary end-user of the product is the *School of Computing, Creative Technologies, and Engineering* at *[redacted]* [2] When a lecturer on the Computing course personally expressed his desire to teach the methods of rendering fractals, he also stated that there existed no all-round software package that was easy to use, met the requirements of platform-non-exclusivity, and preferably available at no cost. As the lecturers have a small amount of experience with fractal rendering, *furakutaru* should allow high level of flexibility in terms of customisation for those who are capable, but simultaneously possess basic ease-of-use principles in order to avoid the alienation of people less familiar. This will be achieved by providing a large range of pre-defined Julia Set configurations, while also building functionality to handle custom/user-entered parameters.

With regards to the level of interface-familiarity that the lecturers shall experience with *furakutaru*, although it is a rather unfamiliar interface, access to basic functionality will be extremely simplistic, such that they are presented with a canvas and a plethora of interactive keyboard shortcuts. Users will also be able to use the built-in command-line help interface when they cannot trivially find a method of performing the desired operation.

Additionally, as the software is primarily designed for a teaching environment, students must also be able to use the functionality with relative ease. The goal of *furakutaru* is to allow users to focus upon the content with which they are viewing and interacting. Many similar software packages mandate the exact opposite, forcing the user to concentrate on the potentially convoluted technical issues which arise as the result of a badly designed and possibly untested interface.

In addition to the very specific case of *[redacted]*, *furakutaru* also bears the advantage of being released under an extremely permissive licence: the MIT Licence (see Appendix C), allowing for the use of the software in any situation by anybody who desires its functionality, without any legal restrictions. Due to this permissive nature, *furakutaru* has an incredibly wide potential end-user group. In recognition of this extremely large and potentially inexperienced user-base, *furakutaru* shall provide extensive opportunities for assistance.

[1] Only Windows and MacOS X are currently supported.
[2] Further information on the aforementioned School and University can be found at [redacted].

2.2.2 Requirements of the Software

Keeping a constant outline of requirements is incredibly important when developing software, as in a 2002 study, it was shown that approximately 52% of 46 software developers stated requirements were "rarely updated" in the documentation when a change was made to the software [FL02]. Below is an exhaustive list of the requirements as defined by the primary client, which not only considers the needs of the lecturers but also the needs of the less-experienced students.

(i) Generate the Mandelbrot Set to an arbitrary degree, this should be entered interactively but also have an initial value which can be entered as a command-line parameter;

(ii) Generate the Julia Set to an arbitrary degree, using an arbitrary constant c (see § 1.2 for more information);

(iii) Allow interactivity with the generated renders, such that the user is able to pan, rotate, and zoom;

(iv) Provide a range of "pre-sets" for items (i), (ii), and (v), allowing inexperienced students to explore basic renders;

(v) Include features allowing the user to edit the colouring of the renders, providing the ability to manually enter colour values.

(vi) Include a feature to save the current render as an image to permanent storage, such as a local hard-disk or network file-system.

(vii) Provide features to load and save fractal configurations from portable text files, allowing lecturers to distribute a single configuration to many students.

2.2.3 Acceptable Limitations of the Software

In addition to the explicit requirements outlined in § 2.2.2, the customer has stated a number of features which they explicitly do not require.

(i) There is no requirement to save an animation of a render to disk[3];

(ii) There is no requirement to create a "pseudo-three-dimensional" effect, as this is irrelevant in the mathematics of fractals[4];

(iii) Additionally, it is not a requirement that users are able to switch between projection planes; the standard μ projection is completely adequate.

2.2.4 Data Sources and Destinations

Due to the nature of the program, *furakutaru* deals with very little permanent data (i.e. data that is not stored in R.A.M.). Most of the current systems mentioned in § 2.1 are very similar, however *furakutaru* aims to increase the autonomy and ease of loading and saving fractals in various states, rather than the user having to manually save formulas to a text file. *furakutaru* will also offer a wide variety of settings for elements of the program which are not directly related to the fractal-rendering process.

furakutaru will store its configuration files in the home directory of the user as "dot-files" [War14] on UNIX-like systems in the basic *CSV*[5] format, chosen due to its extreme simplicity and human-readability [Nur+09]; the exact syntax shall be examined further in the context of *furakutaru* in later sections. Below is a list of configuration file-types which should be saved and interpreted. In the interest of user-friendliness and general usability, it is also a requirement that the user is able to specify a directory to and from which files should be saved and loaded.

[3] Additionally, this superfluous functionality would be impractical from a developer's perspective due to the complexity of video formats.

[4] This refers to rotating a render around the real or imaginary axis, creating a "pseudo-three-dimensional" effect [Nor89]. The *pseudo* nature of this render is due to the rendering process by which the fractal is rendered in two dimensions, then is artificially projected onto a three-dimensional plane using some parameter, such as the colour, for the z positing.

[5] *CSV* is the "Comma-Separated Value" Format, wherein raw values are separated by the comma delimiter.

(i) Fractal Configuration: this is the most immediately obvious type of configuration file. It originates from users saving fractals from within *furakutaru*, and can be opened at a later date to restore the *exact* state. This fractal can then be interacted with in the same fashion as any other render;

(ii) Colour Palette: this originates from users defining their own colours from outside of the application, as the format proves to be so simplistic that users can directly input red-green-blue-encoded colours. Alpha values will not be supported. Colours will be stored using the PPM format in a one-dimensional format: 2^n`x1` dimensions.

Additionally, non-text binary information can be exported from *furakutaru*, such as the images of rendered fractals.

2.2.5 Data Volumes

As the main data is human-readable text, *furakutaru* will create an extremely small data-footprint on the user's storage medium, unless they were to create an absurdly unrealistic amount of data. Although a reasonable computer will be required to run the rendering processes, the application will be incredibly undemanding with regards to permanent data, and only may be strenuous to any modern storage medium if the user exports multiple-thousand images.

2.2.6 Data Dictionaries

The basic fractal-configuration file will consist of the data required by *furakutaru*, and no other metadata. If a user provides any file under the claim it is a valid *furakutaru* CSV, it will be assumed as such until an error is found in parsing.

Colour files should consist of the standard PPM header, followed by the pixel data. This is shown in Figure 2.1. The specifics of parsing a PPM file are discussed in the Design section.

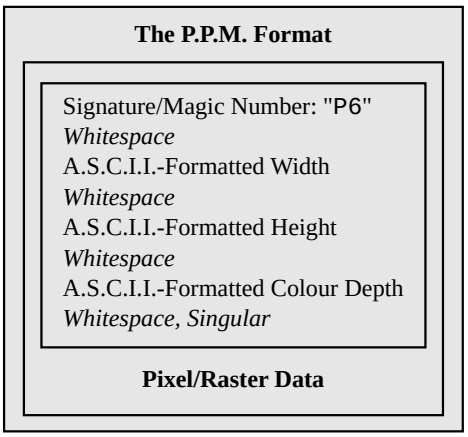


Figure 2.1: The PPM format, retrieved from [The16]. The pixel stream is uncompressed.

2.2.7 Use-Cases

Use-cases are a list of steps in which the actions between an "actor" and a system are described. The actor is typically representative of the user [Coc00], while the system refers to the software. Below, three goals are defined, and the required interaction between the actor and system is described.

- Rendering a custom Julia Set and save the configuration to a storage medium.

  (i) The actor initialises the system;

  (ii) The actor selects the "Julia Set" option;

  (iii) The system allows the actor the option to enter a focus point/"seed" and degree. The system initially provides the user with defaults;

  (iv) The actor confirms their choice of parameters;

  (v) The system renders the fractal to the screen;

(vi) The actor selects the appropriate option to save the current fractal configuration to a storage medium;

(vii) The actor enters a file path and confirms their intention to save;

(viii) The system takes the current parameters and writes a fractal configuration file at the specified path.

- Interactively rotating a pre-existing fractal configuration, followed by saving an image to a storage medium.

  (i) The actor initialises the system;

  (ii) The actor selects the appropriate option to load an existing fractal configuration;

  (iii) The system loads the fractal configuration and displays the appropriate on-screen render;

  (iv) The actor selects the appropriate option to rotate the fractal, and enters an appropriate degree of rotation;

  (v) The system re-renders the fractal with the specified rotation applied;

  (vi) The actor locates the option to save the current render as an image, and enters a file path for the new image;

  (vii) The system saves the current buffer contents to the specified path as an image.

- Plotting a "multi-Brot" (§ 1.4) set.

  (i) The actor initialises the system;

  (ii) The system presents the actor with a set of rendering options

  (iii) The actor selects a Mandelbrot Set and specifies the degree of the rendering polynomial;

  (iv) The system interprets the parameters and renders the fractal to the screen.

2.2.8 Objectives

The following *S.M.A.R.T.* targets[6] outline the specific objectives that must be completed in order to satisfy the user, whose basic requirements were outlined in § 2.2.2. Objectives are placed into three categories:

- System Objectives

  (i) The system shall have the functionality to dump information concerning the render. This should include common elements such as the type of set, degree, zoom level, and rotation amount;

  (ii) The rendering system shall provide "pre-sets" for colour palettes and popular fractal configurations;

  (iii) Include the functionality to save fractal configurations in their exact state to be shared and opened.

- Processing Objectives

  (i) Render both the Mandelbrot and Julia Sets;

  (ii) Correctly read and interpret the configuration files outlined in § 2.2.6;

  (iii) Respond correctly to external events sent by the Desktop Environment, such as an invalidation through the user resizing or moving the window.

- User Objectives

  (i) Provide the user with the option to save the current render as an image;

  (ii) Allow the user to pan, rotate, and zoom into the render;

  (iii) Allow the user to enter an arbitrary degree for the render function.

See Appendix B for the client's approval of these objectives and a questionnaire conducted to solidify the aims.

[6] *S.M.A.R.T.* targets are a way of setting high-quality targets to maximise their chance of achievement. The acronym *S.M.A.R.T.* stands in-place of "specific, measurable, attainable/achievable, relevant/realistic, and time-bound" [HG00].

2.2.9 Potential Solutions

There are a countless amount of technology stacks which enable a graphical user-interface, however it is incredibly important to select the correct option. As *furakutaru* aims to be as platform-independent and high-performance as possible, it was known from the outset of the development that a device-independent method of direct hardware access must be utilised.

This creates a slight contradiction however, as low-level access does not commonly allow for platform-non-exclusivity, while high-level access often severely hinders performance due to the over-abundance of abstraction layers. This gap has become less prevalent in recent years however, as languages running under virtual machines have seen significant developments with respect to performance. In a 2003 paper, research into the execution times of *Java* versus *C* and *Fortran* showed that compiled languages held negligible advantages over the V.M. candidate [Bul+03]. Simultaneously, a 2005 paper found that *Python*, while somewhat viable for parallel-processing scientific applications, is only optimal when combined with other languages for the high-intensity computations [CLM05].

In addition to the language, considerations also need to be made with regards to the programming interface. Lower-level languages such as *C* do not contain any sort of graphical abilities in its standard library, whereas the aforementioned higher-level languages such as *Java* and *Python* ship with G.U.I. capabilities much closer to the core language — *Java* uses *A.W.T.* (*Abstract Window Toolkit*) and *Swing* [Loy+02] while *Python* uses *Tkinter* [Lut01].

Due to the computationally intensive nature of *furakutaru*, a layer for high-performance rendering is also required. The *Simple DirectMedia Layer*, commonly known as *S.D.L.*, is a popular candidate due to its lightweight footprint in addition to its simple programming interface [Mit13]. A more basic approach is the *XCB* interfacing library for the X windowing system, which provides extremely low-level access to a client-server windowing environment [JR93], however this approach is generally infeasible due to time constraints and the amount of code required to have a fully functioning application. This can be somewhat mitigated by *OpenGL* (*Open Graphics Library*), as it is a layer that resides on a windowing interface, allowing direct manipulation of the hardware. As a fractal's pixels are rendered independently, hardware acceleration is a requirement due to its ability to execute the colouring algorithms in parallel — this is especially true on modern graphics-processing units [YHL11].

Below is a consolidation of the discussed approaches in order to conclude the advantages and disadvantages.

- ***Python* and *OpenGL*.** This approach would provide a portable and timely software solution, as the *Python* virtual machine has the capability to run on any modern, mainstream operating system. While *OpenGL* would also provide a fast hardware-interfacing layer, the severe performance issues of *Python* renders this to be an unlikely candidate due to the nature of the application and its explicit focus on speed;
- ***Java* and *OpenGL* in addition to a standard windowing library.** Similar to the previous solution, a *Java*-based solution would provide an extremely portable solution which would possess the advantage of having the ability to execute on anything for which the *Java* virtual machine can be compiled. Such a solution would also provide useful features such as garbage collection and an object-orientated approach, allowing for the clean structuring of complex programs. Unfortunately, this proposal suffers from the issue of the heavyweight nature of the *Java* virtual machine, such that *furakutaru* should not be recourse-intensive;
- ***C* and *The Simple DirectMedia Layer*.** A minimalist and lightweight approach, using the *Simple DirectMedia Layer* on a *C* base would allow for extremely low-level and fast access to rendering. Additionally, the *Simple DirectMedia Layer* is designed, by nature, to leave a very small footprint. The limitations suffered from this approach include the inability for *S.D.L.* to directly access and manipulate the hardware without additional layers, hence disallowing all hardware-accelerated parallelisation;
- ***C* and *XCB* with *OpenGL*.** A similar approach to the previous description, a *C*-based approach utilising the *X11* window manager via *XCB* allows for the lowest amount of abstraction possible. This minimalistic approach enables the maximum performance, and also provides direct hooks for the *OpenGL* subsystem.

From the candidates outlined above, *C* and *XCB* with *OpenGL* has been selected as the most suitable contender for developing the application. Its lightweight nature allows the possibility for a near-zero footprint on modern systems, in addition to its low levels of abstraction enabling maximum performance via hardware acceleration and parallelisation while incurring minimal overhead.

Appendix A

Syntax Specifications for Configuration Files

A.1 Fractal Configuration Syntax

- **Angle of Rotation.** The *angle of rotation* field denotes the number of degrees, in the range of zero to three-hundred-and-sixty, the amount to which a fractal is rotated past the positive x-axis. The initial value is zero, and the actual value can be floating point.
- **Degree of Polynomial.** The *degree of polynomial* field stores the greatest exponent of z in the generator-polynomial (see Equation 1.6 and § 1.4). This holds an initial value of two and otherwise must be any integer other than -1, 0, or 1.
- **Maximum Iterations.** *Maximum iterations* denotes the number of iterations must be used for determining the membership of a point in the set (set § 1.1). Initially holding a value of 1000, the field is an unsigned natural number, excluding zero.
- **Fractal Style.** The *fractal style* field is for storing the type of fractal, whether that be a Julia or Mandelbrot Set. This is stored as a nullable string-enumerator, such that valid values are `MANDELBROT` (‘`m`’) and `JULIA` (‘`j`’).
- **Constant/Seed.** Only applicable for Julia Sets, the *constant* field stores the seed from which to generate the Julia Set. See § 1.2 for more information. This is stored as a complex number with a default value of zero. Complex numbers will be stored as an separate values of their real and imaginary parts, as the *CSV* format does not have any regard for complex numbers: `[`$\Re(z)$`,` $\Im(z)$`]`. A lack of complex number support is common in even more comprehensive formats, such as *J.S.O.N.* and *X.M.L.* [Dro19].
- **Lower-Left-Most and Upper-Right-Most Points.** Stored as complex numbers, these two fields store the lower-left-most and upper-right-most points. This allows the canvas state to be saved and restored while maintaining the correct viewpoint. The fields have initial values of $-\frac{5}{2} - \frac{3}{2}i$ and $\frac{1}{2} + \frac{3}{2}i$ respectively.
- **Zoom Factor.** Applicable for all renders, this value determines the extent to which the fractal has been zoomed. It is a single floating-point real number with the initial value of 2.2.

Appendix B

E-Mail Correspondence Regarding Objective Approval and Client Questionnaire

B.1 Initial Meeting and Questionnaire

The following is an exchange between myself and the Client, in which the Client completed an initial questionnaire regarding their desires for *furakutaru*. The included document is a transcription of the questionnaire question-and-response sheet.

```
(Q.) On which platforms and/or operating systems do you require the
      Fractal-Rendering Systems?
(A.) "Our labs mainly make use of Linux, and other UNIX-like systems, primarily
      used for teaching the programming and security modules. Support for Mac
      OS X would be somewhat-desirable, as the Media Labs are Mac-exclusive,
      however the support for Linux and other P.O.S.I.X.-compliant systems is
      the first priority."

(Q.) As with all mathematical and visual calculations, fractal-computation can
      be intensive. What sort of hardware should the System target?
(A.) "Many of our labs have recently received dedicated graphics-processing
       units to facilitate the Games-Programming modules. Programming with
       Mathematics shall also be taught in these labs, so whilst the System
       should be as lightweight as possible, it should also take advantage of
       the high-spec hardware available to the University."

(Q.) Is it of significant importance that users are able to save and re-load
       fractal configurations to and fro permanent storage?
(A.) "Our modules are generally taught over long periods of time. In addition
       to the ability to have pre-sets, it would be incredibly advantageous if
       a user was capable of saving a fractal configuration for later
       interactions. It would also be of great value if the System was capable
       of exporting renders to a common image format, such as P.N.G."

(Q.) You mention the concept of a user interacting with a fractal. What
       operations do you desire in this regard?
(A.) "Basic exploration is a must, including panning and zooming. Other
       facilities are highly appreciated and will surely prove useful, however
       the two integral elements of interaction are most important to our
       use-case. The System must also allow a user to request information
       regarding the current fractal, which consists of a data-dump displaying
       the various properties of the current render. Users should also be
       allowed to dynamically change the maximum iteration count."

(Q.) The Mandelbrot and Julia Sets are the most common in the fractal
       communities. Should the System possess the ability to generate any
       additional fractal configurations, such as a Newton Fractal or
       Sierpinski Triangle?
(A.) "The Mandelbrot and Julia Sets are completely adequate for our use-cases
       at the University. However, it would be interesting and
       potentially useful if the System was capable of rendering the
       aforementioned fractals in differing degrees, including both negative
       and positive integer exponents."

(Q.) Are there any additional features which the System should include?
(A.) "Colouring always struck me as an important feature of fractals. In
       addition to providing a higher aesthetic value, I see colour as a
```

```
convenient method of teaching the concept of indexing into a
one-dimensional texture. A single colour configuration would be
satisfactory, however it would be well-received if the System was able
to load various colour configurations without requiring a
re-compilation."
```

B.2 Conclusive E-Mail

The following verbatim text was taken from an e-mail conversation between myself and the Client on 11th July, 2019, confirming the validity of the objectives that had been defined. This exchange was drafted as a summary of the above questionnaire.

```
From: DIX18002469 - Oliver Dixon
Sent: 11 July 2019 05:52
To: [REDACTED]
Subject: Regarding the *furakutaru* Objectives

Dear [REDACTED];

Please see below for the list of objectives that have been outlined for your fractal-
    rendering software.

 -- The system shall have a "status" overlay, displaying information regarding the render;
 -- The rendering system shall provide "pre-sets" for colour palettes and popular fractal
     configurations;
 -- Include the functionality to save fractal configurations in their exact state to be
     shared and opened;
 -- Render both the Mandelbrot and Julia sets;
 -- Correctly read and interpret the configuration files;
 -- Possess the ability to use custom and externally defined colour palettes;
 -- Provide the user with the option to save the current render as an image;
 -- Allow the user to pan, rotate, and zoom into the render;
 -- Allow the user to enter an arbitrary degree for the render function.

It would be helpful to know whether the objectives are to your and the University's
    satisfaction.
I thank you in advance and look forward to your response.

Yours Sincerely,
Oliver Dixon.
```

```
From: [REDACTED]
Sent: 11 July 2019 13:53
To: DIX18002469 - Oliver Dixon
Subject: RE: Regarding the *furakutaru* Objectives

Dear Oliver, thank you for the listed objectives. I confirm these are to the satisfaction
    of myself and the University. Additionally, I should mention that while the rendering
    of only the Mandelbrot and Julia sets are entirely satisfactory, the inclusion of
    supplementary fractal presets is welcomed.

Yours Sincerely,
[REDACTED]
```

Appendix C

MIT Licence

The following licence text was imported, in-verbatim, from the *furakutaru* `LICENSE` file.

```
MIT License

Copyright (c) 2019 Oliver Dixon

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

. . . and don't be evil . . .

Stage II

Design

Section 3

Overall System Design

3.1 Overview of the System

The *furakutaru* System should interface with the user using primarily visual means, as it is an aesthetically focused application; command-line arguments should also be interpreted to provide an initial state of the render and interface. The System will be comprised of a front-end rendering view, in which the visual representation of the Set is presented, and the *OpenGL* processes which execute, in parallel, on the dedicated Graphics-Processing Unit cores. Upon application instantiation, a user-customisable render is displayed, and once the loading is complete, the user is allowed to zoom, pan, and rotate the render using their keyboard. Figure 3.1 visualises the ways in which the *furakutaru* system interprets input from various sources, and subsequently distributes the load of calculation to the various GPU cores.

Multi-threading was considered, such that the user would be able to interact with the previous render whilst other operations were being performed on another thread, however this was ultimately abandoned due to its ineffectiveness, as software-implemented multi-threading is mostly emulated on the CPU, and thus offers little actual performance gain. By design, *OpenGL* pixel-independent operations are performed on separate GPU cores due to the design of the *OpenGL* implementation [Vas10]; this renders the application, although not multi-threaded in a traditional sense, capable of parallel-processing nonetheless.

The Set-rendering and colouring algorithms, described in Appendix 4, are to be implemented using the *OpenGL Shader Language*, commonly abbreviated GLSL The C-like language, in addition to natively supporting mass-parallel execution across all available GPU cores, effectively enables the programmer to write a modularised, and thus easily maintainable, code-base, such that the rendering and application-control algorithms are kept entirely separate.[1]

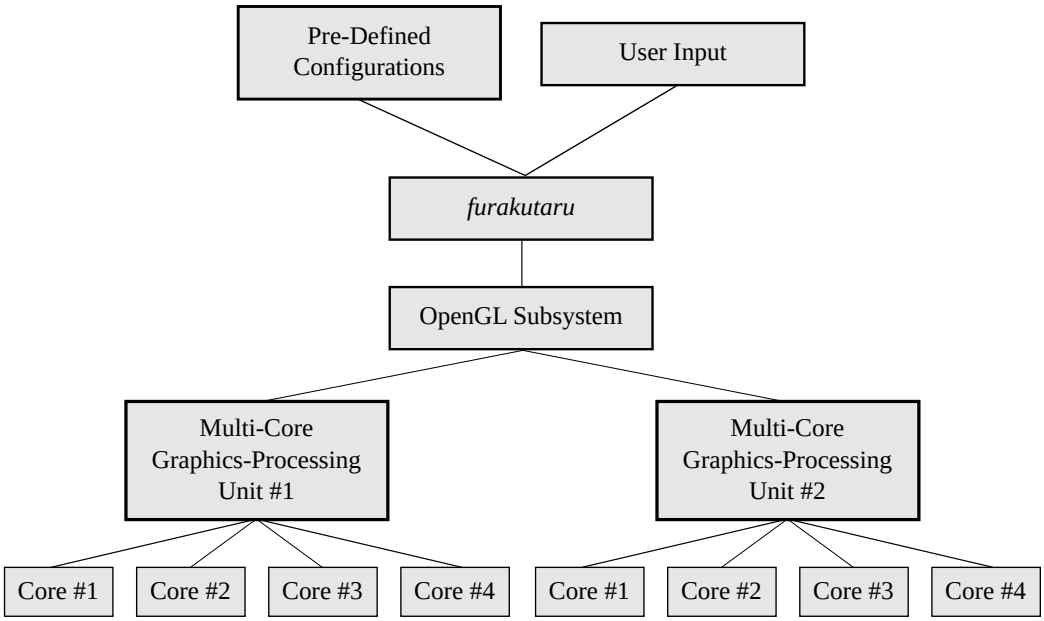


Figure 3.1: A Hierarchical Diagram demonstrating the ways in which both the configuration files and the user interact with *furakutaru*, and how that is subsequently distributed to the various GPU cores. In reality, thousands of cores would exist on a single GPU [ND10].

[1] As C is a procedural language, and thus has no concept of object-orientation or a class-model, good programming practices are integral to ensure maintainability of the code. An example of this is assuming a function which takes a pointer to a structure to be a method and class respectively.

3.2 User-Interaction with the System and HCI Rationale

As briefly mentioned in the Analysis, *furakutaru* should support, at a minimum, the ability for users to rotate, pan, zoom, create fractals, and load fractals. The ability to export fractals as images is a desirable feature for the aesthetic value, however as *furakutaru* is primarily designed as an educational tool, the such an ability is not a priority. Figure 3.2 summaries the methods in which a user can interact with the System, and a non-comprehensive list of the possible actions.

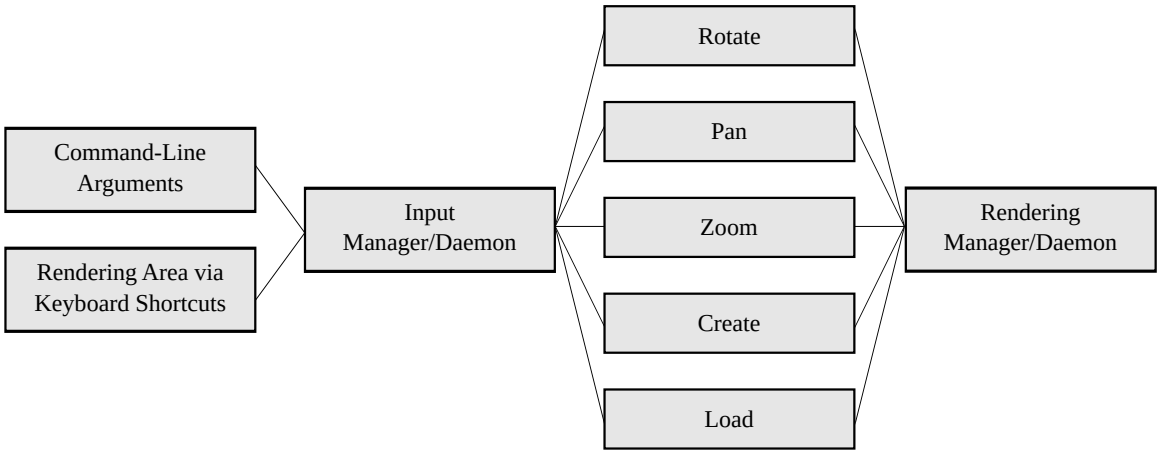


Figure 3.2: A representation of the relationships between an end-user, the *furakutaru* daemons, and the rendering system.

As *furakutaru* attempts to be minimal, it does not use a standard user-interface design library or framework, such as the C# *WinForms* or Java *Swing*. This creates the dilemma of the user-interface, as it all must be manually implemented in a non-standard fashion, which is likely irrespective of the windowing manager under which it is being executed. Minimalistic window managers such as *i3*—the windowing manager of the test system—or something as heavy as *GNOME* have very different methods of displaying various form-like elements, such as windows, combo boxes, and buttons. In order to mitigate the discrepancies between window managers, much of the *furakutaru* human-computer interaction will be performed through keyboard shortcuts, activated by the user when focused on the main rendering area.

This design principle of keyboard-exclusivity has been adopted throughout many applications [LR98] due to the additional speed it offers, in addition to the fact that many UNIX users—the primary target group of *furakutaru*—are accustomed to using terminal-only applications, in which the keyboard is ubiquitous.

It is because of this that an extensive range of command-line options and switches should be implemented, which allows the user to specify the initial state of the application, in addition to controlling specific options.

3.2.1 The Command-Line Interface

A well-documented and flexible command-line interface allows users to unambiguously control the *furakutaru* system, but also incorporate its usage into automatic tasks, such as shell scripts. Figure 3.3 presents a simple command-line invocation over a remote connection, demonstrating the usefulness of a strong command-line interface for a UNIX-like environment.

```
$ ssh name@workstation
$ export DISPLAY=:0
$ ./furakutaru --type mandelbrot --degree 5 --lock
```

Figure 3.3: A simple command-line invocation of *furakutaru* on an *X* server via a Secure Shell connection, instructing the render to place a Mandelbrot Set of the fifth degree, and lock it from receiving any sort of input to pan, rotate, or zoom the image.

The details of the command-line interface, including a comprehensive listing of the options and their respective uses and implementations, will be discussed in the Technical Implementation stage. Figure 3.5 shows the way in which the System can be initiated to a specific state, and then controlled with keyboard triggers.

For features which require some level of interaction, such as the input of a file path from which to load a fractal configuration, can also be entered via the command-line in the default input buffer—`stdin`. Although not making use of a full graphical user-interface, this method of input is a reliable and universal method of accepting input from the user, it does not require an X server to be running on the machine issuing commands, as shown in Figure 3.3. This allows for a remote machine to continue controlling the operations of *furakutaru* not just for its invocation, but also throughout its execution.

3.2.2 Feedback from the Renderer

The HCI philosophy of *furakutaru* also mandates that the user must be constantly able to retrieve information regarding the current state of the render. Figure 3.6 shows an example of a user using a pre-defined keyboard shortcut to request feedback. In practice, the feedback should be far more comprehensive, showing any additional information that may be useful, such as the configuration file from which the current fractal originates. If verbosity was requested upon command-line invocation, information should be printed to feedback buffer whenever a change to the render is made due to a keyboard shortcut, such as a rotate or zoom—c.f. Figure 3.7.

This design philosophy is heavily inspired by that of the *UNIX Set of Principles*, originally described in 1974 [MPT74].

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features";
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input;
3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them;
4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

Most importantly, however, is a further comment made in 1994, in which the significance of a 'universal interface' is described, allowing I./O. congruency between otherwise-unrelated applications [Sal94].

> "This is the UNIX philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle **text streams, because that is a universal interface**."

3.2.3 Input Validation and Sanitisation

When a user invokes an action requiring additional interaction, such as the input of a file path, it is integral that the data provided is checked for validity; this is to ensure the integrity of the state of *furakutaru* at all times. Attempting to read from files that do not exist, or even memory which is disallowed, could lead to a segmentation fault and subsequent complete crash of the System[2].

Below is a list of both valid and erroneous inputs to various prompts, and the ways in which they should be handled by the System. An example of the latter can be seen in Figure 3.4.

[2] An unexpected shutdown, such as one due to a `SIGSEGV` (segmentation fault/violation), could lead to subsequent issues on older operating systems with inferior memory management. Although modern systems typically clean up memory after a process exits, by any means, some archaic or embedded systems do not clean the leakage [Dhu+03].

- "*Enter the path from which to load the fractal configuration:* **`/home/user/myfractal.frac`**". This is a valid input, providing the file exists and the user under which *furakutaru* is running has read permissions. The System should proceed to open, load, interpret, and close the file.
- "*Enter a fractal type;* ***`(m)andelbrot or (j)ulia:`*** **`yes`**". This is an erroneous response, as the choice of 'M' or 'J' was clearly given in the prompt. In this event, either a default should be assumed, or an error message should be issued and the interactive operation halted.
- "*Enter a positive integer for the fractal degree:* **`2.5`**". This is another erroneous response, as the given data is not of the request data-type. An error should be issued, in addition to a system-issued error thrown by a standard string-to-integer function.

```
$ ./furakutaru
  Enter a positive integer for the fractal degree:
  2.5
  Error:  invalid data-type, '2.5'.
  Enter a positive integer for the fractal degree:   awaiting input
```

Figure 3.4: An example error message caused by a user's inability to enter a positive integer.

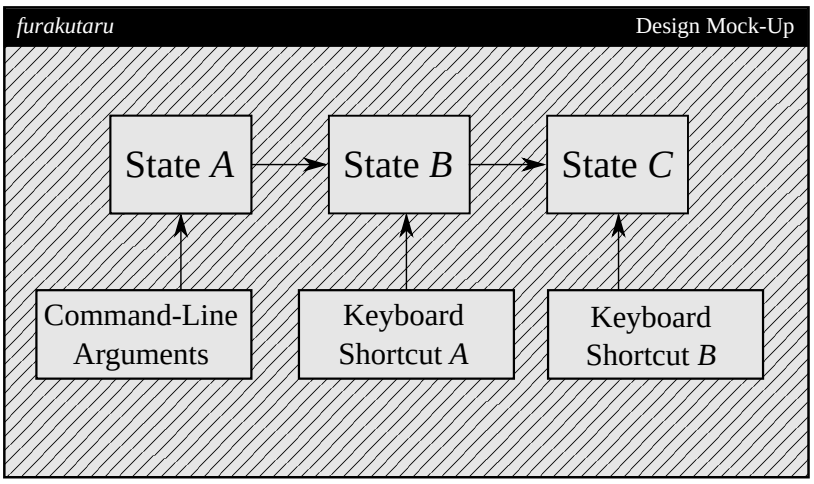


Figure 3.5: A simple interaction between the *furakutaru* system and the various methods of input. The command-line arguments resulted in State *A*, and subsequent user-inputted keyboard shortcuts are used to switch between the State *B* and State *C*.

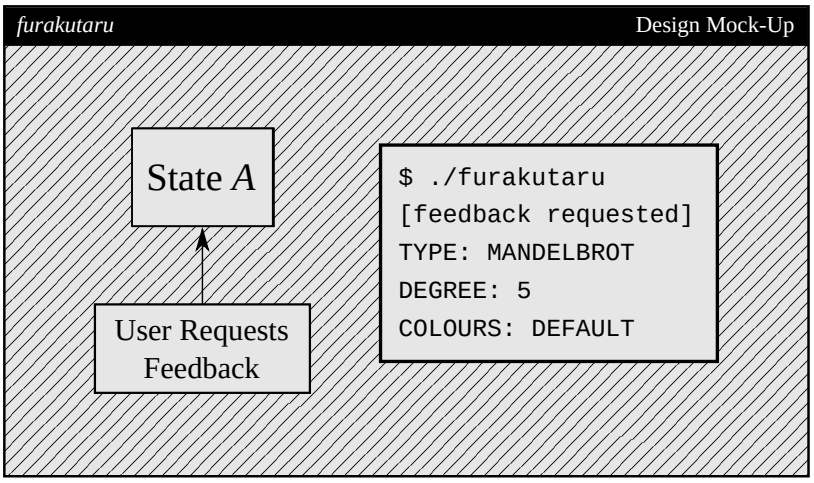


Figure 3.6: In this example, 'State *A*' represents a Mandelbrot Set of the fifth degree, rendered with the default colour scheme. Upon requesting feedback from the renderer, a concise, human-readable report is output to the console—usually, on UNIX-like systems, this is `stdout`.

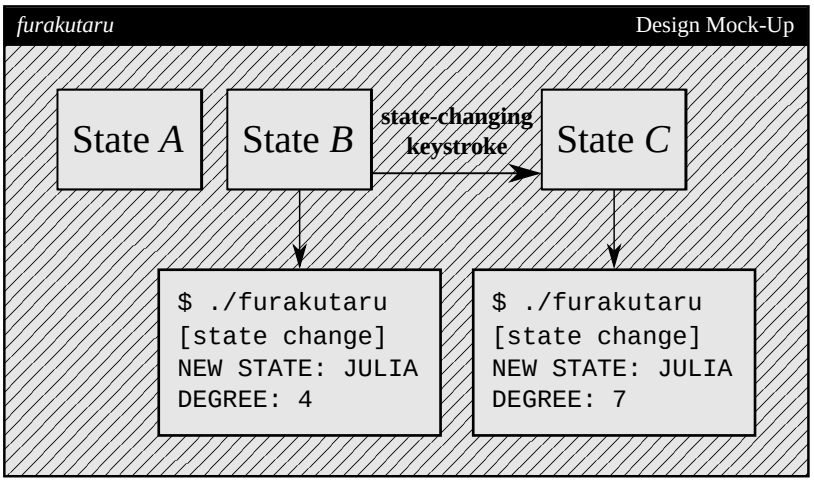


Figure 3.7: With the flag of verbosity, *furakutaru* should report every change to the current render. During development, this could include time to render frames, and any other debugging messages.

Section 4

Plotting and Colouring Algorithms

4.1 The Mandelbrot Set: The *Escape-Time* Algorithm

The *Escape-Time* algorithm, or E.T.A., is the most popular method of fractal rendering due to its inherent simplicity. Described in the analysis, the algorithm uses a threshold and a maximum iteration count to determine the convergent or divergent nature of any particular point. Algorithm 1 describes a basic, unoptimised implementation of the *Escape-Time* algorithm.

Algorithm 1 The basic *Escape-Time* algorithm mapping to an $\mathbb{R}^2$ plane. Values should be normalised in the ranges $x : [-2.5, 1]$ and $y : [-1, 1]$, represented by $min_{x,y}$ and $max_{x,y}$. The draw(x, y, c) pseudo-function draws a pixel of colour c to the position (x, y). The constant t can be assumed to be equal to '2' in most cases, as this draws the standard Mandelbrot set.

Require: Maximum number of iterations: $N \in \mathbb{N}^1$
Require: Threshold for which $P \in \mathcal{M}$: $t \in \mathbb{R}^+$
Require: The number of colours in the palette: $C \in \mathbb{N}_{>1}$
Require: A colour palette of size C: *palette*
for all $P(x, yi)$ in the viewport **do**
  $x_0 \leftarrow (\Re(P) - \min(x)) \,/\, (\max(x) - \min(x))$
  $y_0 \leftarrow (\Im(P) - \min(y)) \,/\, (\max(y) - \min(y))$
  $x \leftarrow 0$
  $y \leftarrow 0$
  $n \leftarrow 0$
  while $x^2 + y^2 \leq t^2$ **and** $n < N$ **do**
    $x_{temp} \leftarrow x^2 - y^2 + x_0$
    $y \leftarrow 2xy + y_0$
    $x \leftarrow x_{temp}$
    $n \leftarrow n + 1$
  end while
  draw$(x,\ y,\ palette[n \bmod C])$
end for

This can be further optimised to reduce the number of instructions, specifically multiplications, inside the inner `while` loop [HH95]. Other variations of the E.T.A. have been formulated, none of which are direct optimisations, but rather speed-quality trade-offs, unsuitable for a general-purpose application [Liu+15].

Algorithm 2 A trivially optimised *E.T.A.*, using the fact that $(x+yi)^2 = x^2 + 2xyi - y^2$ to reduce the number of multiplications in the inner loop. For conciseness, the **Require** statements from Algorithm 1 have been implicitly inherited here, and only the inner loop and surrounding lines are shown.

while $x + y \leq t^2$ **and** $n < N$ **do**
  $x_{temp} \leftarrow x - y + x_0$
  $y \leftarrow (2xy + y_0)^2$
  $x \leftarrow (x_{temp})^2$
  $n \leftarrow n + 1$
end while

4.2 The Julia Set: The Extended *Escape-Time* Algorithm

An analogue to the *Escape-Time* algorithm can be used for rendering the Julia set; this is attributable to the fact that the Mandelbrot and Julia sets are very similar in their mathematical

nature, sharing many geometric properties which can be reflected by their methods of rendering. Unlike the basic *Escape-Time* algorithm, the Extended variant suffers from the need of elementary trigonometric functions: sine, cosine, and the arc-tangent.

This method of rendering is commonly grouped as an Iterated Function System [HPS91].

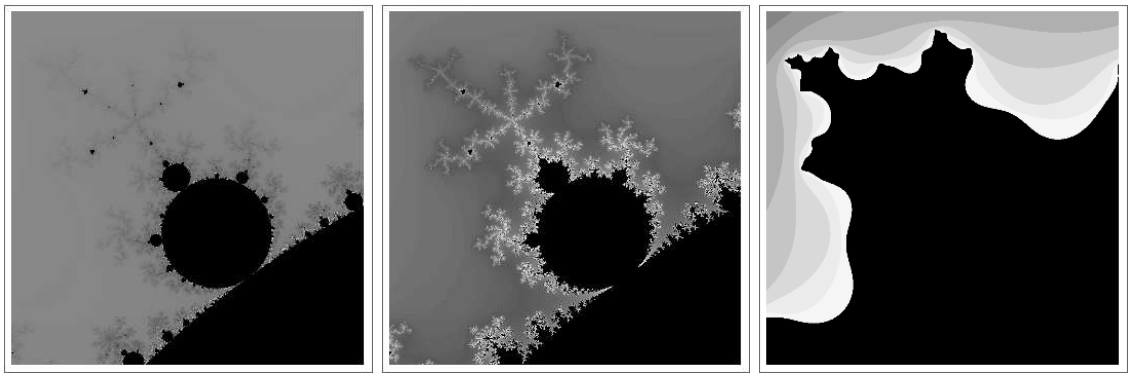
Algorithm 3 The pseudocode for rendering the Julia Set to an arbitrary degree n, from an initial constant c, in which $c \in \mathcal{M}$, where $\mathcal{M}$ represents the Mandelbrot Set. Complex analysis operators $\Re$, $\Im$, and Arg are used respectively to denote the real part, imaginary part, and argument of a complex number.

Require: Maximum number of iterations: $N \in \mathbb{N}^1$
Require: The escape radius, $R > 0$
Require: The number of colours in the palette: $C \in \mathbb{N}_{>1}$
Require: A colour palette of size C: *palette*
for all $P(x, yi)$ in the viewport **do**
  $x \leftarrow (\Re(P) + R)\,/2R$
  $y \leftarrow (\Im(P) + R)\,/2R$
  $n \leftarrow 0$
  while $x^2 + y^2 < R^2$ **and** $n < N$ **do**
    $x_{temp} \leftarrow (x^2 + y^2)^{n/2} \cos(n\text{Arg}(P)) + \Re(c)$
    $y \leftarrow (x^2 + y^2)^{n/2} \sin(n\text{Arg}(P)) + \Im(c)$
    $x \leftarrow x_{temp}$
    $n \leftarrow n + 1$
  end while
  draw(x, y, *palette*[n mod C])
end for

4.3 Methods of Colouring

In Algorithm 1 and Algorithm 3, the `draw` function is assumed, such that `draw(`x, y, *colour*`)` colours the pixel (x, y) with *colour*, where *colour* is the dereferenced index of a pre-defined colour palette, with the element being dependent on the number of iterations performed.

Figure 4.1 presents an extreme exemplar of the severe aliasing which basic colouring can ensue. Because of this, a variety of aesthetically pleasing methods have colouring have been devised, allowing a larger colour palette than the basic divergence algorithm would permit [YH04]. Each of the following algorithms index into a palette to determine the colour of the pixel, however obtain the index with more novel and unique means than calculating the modulus of the final iteration with the maximum iterations.



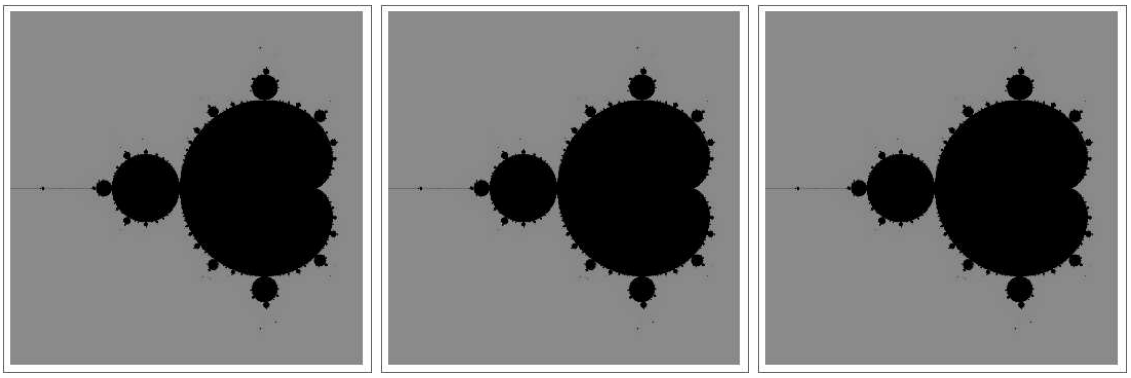
(a) Maximum Iterations → 1000 (b) Maximum Iterations → 100 (c) Maximum Iterations → 10

Figure 4.1: Severe Aliasing caused by the Basic Divergence Colouring Method

4.3.1 Histogram Colouring

Whilst still suffering from the effect of aliasing, Histogram Colouring renders the "banding" as far less visually significant. This method of colouring also causes the render to become independent of

the maximum iterations chosen. Figure 4.2 shows the same histogram-coloured fractal with differing iteration limits. This quad-pass nature of the method creates issues for low-performance computers attempting to render for large viewports, however, as three out of the four passes must iterate through every pixel.



(a) Maximum Iterations → 10000 (b) Maximum Iterations → 1000 (c) Maximum Iterations → 100

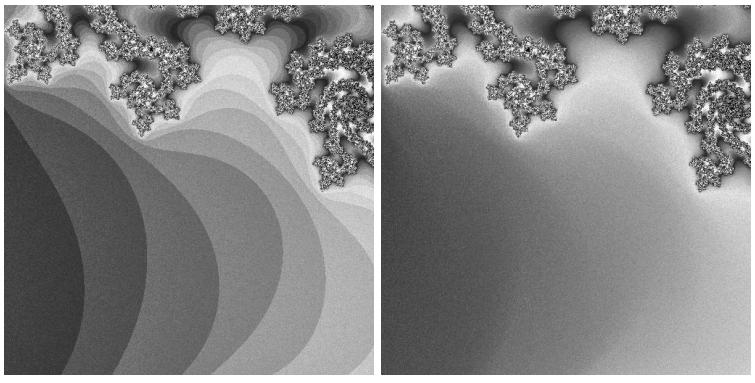
Figure 4.2: The Effect of Histogram Colouring on Identical Plots

The algorithm concerns creating an array map to hold the number of iterations which each pixel endured before escaping the radius: an array of size n `x` m, where n and m are the width and height of the viewport, in pixels. Upon each iteration, as shown in Algorithm 2, the appropriate member in the two-dimensional array should be populated with the bail-out time.

After the two-dimensional array has been fully populated and the viewport entirely traversed, the histogram is generated, as shown in the various phases of Algorithm 4 (see overleaf); this involves taking a summation of the distribution values and creating a normalised value to be used as the colour palette index.

4.3.2 Re-Normalisation and Linear Interpolation

The very obvious aliasing on differing colours can be visually unappealing. It is possible for linear interpolation, using a very large bail-out radius—in the order of 2^8—can be used to create a smoothened effect [Gar+00], considered to be of greater aesthetic value. The significant difference is shown in Figure 4.3.



(a) A scene rendered using the E.T.A. with Histogram Colouring, as described in § 4.3.1.

(b) The same scene rendered using the method of Linear Interpolation to "smooth" the colour changes.

Figure 4.3: The Removal of Aliasing using the Continued Linear Interpolation Method

Algorithm 4 Collate the collected iteration counts for each pixel in a histogram-like structure: the iteration distribution.

Require: The width and height of the viewport
Require: Maximum number of iterations: $N \in \mathbb{N}^1$
Require: A two-dimensional array containing the iteration counts for each pixel

Pass 1 Collate the iteration counts into a basic distribution.

```
1  x ← 0
2  y ← 0
3  repeat
4      repeat
5          c ← counts[x][y]
6          dist[c] ← dist[c] + 1
7          y ← y + 1
8      until y > height
9      x ← x + 1
10 until x > width
```

Pass 2 Take the summation of all elements in the distribution.

```
1  sum ← 0
2  n ← 0
3  repeat
4      sum ← sum + dist[c]
5      n ← n + 1
6  until n > N
```

Pass 3 Index the distribution array and normalise the values by the total.

```
1  x ← 0
2  y ← 0
3  repeat
4      repeat
5          i ← 0
6          c ← counts[x][y]
7          repeat
8              counts[x][y] ← counts[x][y] + dist[i]/sum
9              i ← i + 1
10         until i > c
11         y ← y + 1
12     until y > height
13     x ← x + 1
14 until x > width
```

The colouring algorithm is based upon the *Potential Function*; an integral property describing the relationship between the Mandelbrot and Julia sets—the details of which are beyond the scope of this project report. The function, denoted as $\phi(z)$, is defined as follows, where P is the power to which the set has been raised, and k is the iteration cardinal. The following series of equations is the summary of a long derivation; the latter result was derived by selecting a very large bailout radius, $N = 10^{100}$, to evaluate the limit [Kim15; Bri88].

$$\phi(z) = \lim_{k\to\infty} \frac{\log|z_k|}{P^k} \tag{4.1}$$

$$\mu(z) = k - \log_P\left(\frac{\log|z_k|}{\log(N)}\right) \tag{4.2}$$

$$\frac{\log|z_k|}{P^k} = \frac{\log(N)}{P^{\mu(z)}} \tag{4.3}$$

Algorithm 5 presents a modified Escape-Time Algorithm, as first shown in Algorithm 1 using the above results.

Algorithm 5 Pseudocode for rendering a Mandelbrot Set with Smoothened Colouring, adapted from the Escape-Time Algorithm. The `frac` function provides the fractional part of a floating-point number. The `interpolate` function in a standard linear interpolation, with the purpose of blending R.G.B. components.

Require: Maximum number of iterations: $N \in \mathbb{N}^1$
Require: The bailout radius, t
Require: A colour palette: *palette*
for all $P(x, yi)$ in the viewport **do**
    $x_0 \leftarrow (\Re(P) - \min(x)) \,/\, (\max(x) - \min(x))$
    $y_0 \leftarrow (\Im(P) - \min(y)) \,/\, (\max(y) - \min(y))$
    $x \leftarrow 0$
    $y \leftarrow 0$
    $n \leftarrow 0$
    while $x^2 + y^2 \leq t^2$ **and** $n < N$ **do**
        $x_{temp} \leftarrow x^2 - y^2 + x_0$
        $y \leftarrow 2xy + y_0$
        $x \leftarrow x_{temp}$
        $n \leftarrow n + 1$
    end while
    if $n < N$ **then**
        $lg \leftarrow \log(x^2 + y^2)$
        $exp \leftarrow \log(lg/\log(2))/\log(2)$
        $n \leftarrow n + 1 - exp$
    end if
    plot($\Re(P), \Im(P)$, interpolate(*palette*[floor(n)], *palette*[floor(n) + 1]), frac(n))
end for

Side-Definition The linear interpolation method, `interpolate`

function interpolate(a, b, c)
    return $(1 - c)a + cb$
end function

Section 5

Noteworthy Auxiliary Algorithms

The following sections describe *furakutaru* algorithms which are mere auxiliary helpers, and do not serve directly to the rendering of the fractal. These are to be executed exclusively on the CPU, and thus implemented in *C*.

5.1 PPM Loader

As mentioned in the Analysis section, colours palettes are to be saved as a one-dimensional PPM file. As PPM files are capable of storing headers and comments, it is important that the *furakutaru* loader is capable of reading them correctly, as an error in parsing could lead to erroneous rendering, or the more likely situation of complete absence thereof. Algorithm 6 provides a simplified algorithm for extracting non-comment and non-white-space data from the file, provided it resides in an in-R.A.M. buffer.

Algorithm 6 Pseudocode for parsing the header of a PPM file. This algorithm overwrites the original buffer with a parsed one, saving memory.

Require: A null-terminated buffer containing the PPM data: *buf*.
Require: A constant storing the comment marker. Usually a hash, '`#`', this is represented by the constant *C_CHAR* .

```
1  count ← 0
2  char ← 0
3  while ch is valid do
4      ch ← next character from buf
5      if ch is not white-space then
6          if ch = C_CHAR then
7              while ch is valid and ch is not a new-line character do
8                  ch ← next character from buf
9              end while
10         end if
11         Next position in buf ← ch
12         count ← count + 1
13     end if
14 end while
```

5.2 Pixel-Retrieval

Similar to Algorithm 6, Algorithm 7 uses a pre-populated buffer to retrieve the pixels from the data-stream section of the one-dimensional image. This algorithm assumes that all the necessary safety checks have been performed, and the data in the buffer is valid and of the correct size, as indicated by the header values.

Algorithm 7 assumes little-endian, however a big-endian implementation would simply need to perform the bitwise equation in the opposite direction, shifting the blue value by the highest amount, followed by the green, followed by the red—the latter of which is not shifted.

5.3 The Processing of Command-Line Arguments

As one of the more intricate elements of program-design, the handling of arbitrary, user-entered strings must be performed with care to ensure robustness and validity. The employment and heavy reliance of command-line arguments by *furakutaru* renders the processing of such as an integral factor which requires the greatest amount of attention. Figure 5.1 displays a flowchart for the top-most argument-processor, and Figure 5.2 details the argument sub-processor.

Algorithm 7 Assuming little-endian, pack the data from the PPM buffer into a separate pixel array consisting of R.G.B. values.

Require: A height and width of the buffer. This is assumed to be an accurate representation of the amount of data in the buffer: *height* and *width* respectively.
Require: A pre-populated buffer of pixel data, with the first element of the buffer being the first byte of pixel data: *buf*.
Require: A pixel array of the appropriate size to store all loaded pixels: *pixels*.

```
1  px_count ← width * height
2  red ← -1
3  green ← -1
4  blue ← -1
5  count ← 0
6  for count < px_count do
7      red ← buf[count]
8      green ← buf[count + 1]
9      blue ← buf[count + 2]
10     pixels[count] ← ( ( red & 0xFF ) ≪ 16 ) | ( ( green & 0xFF ) ≪ 8 ) | ( blue & 0xFF )
11     count ← count + 1
12 end for
```

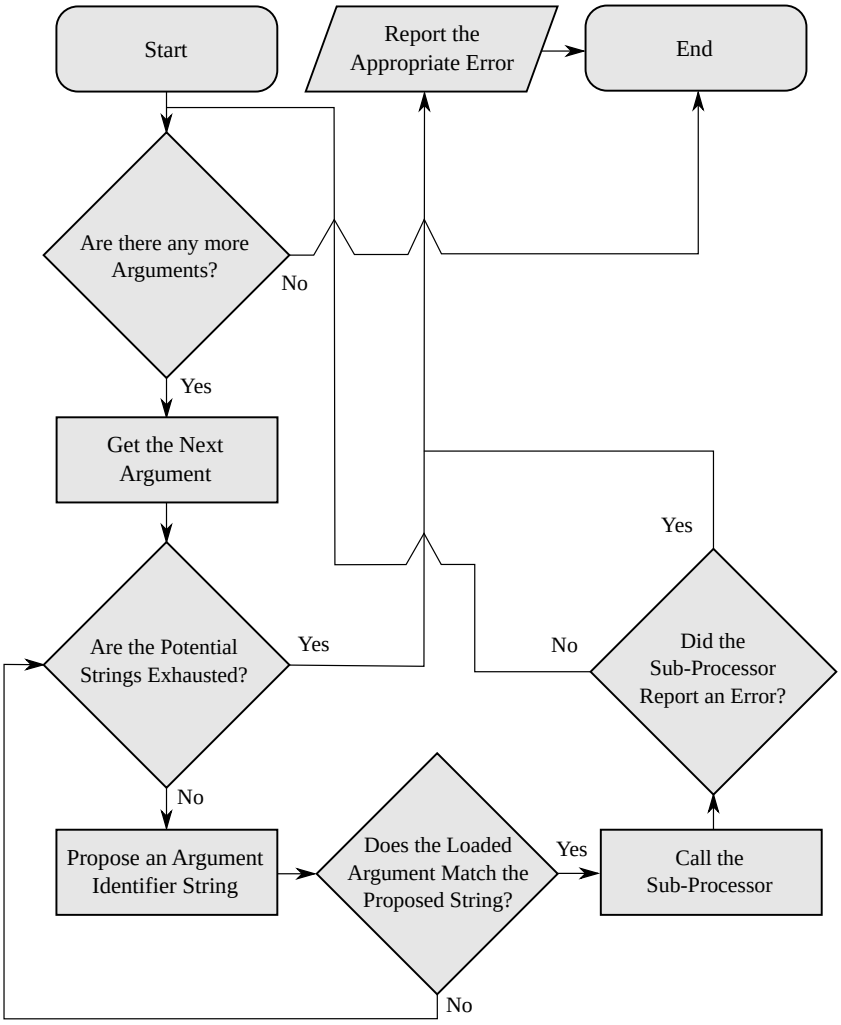


Figure 5.1: A flowchart demonstrating the top-most argument-processor. See Figure 5.2 for the sub-processor.

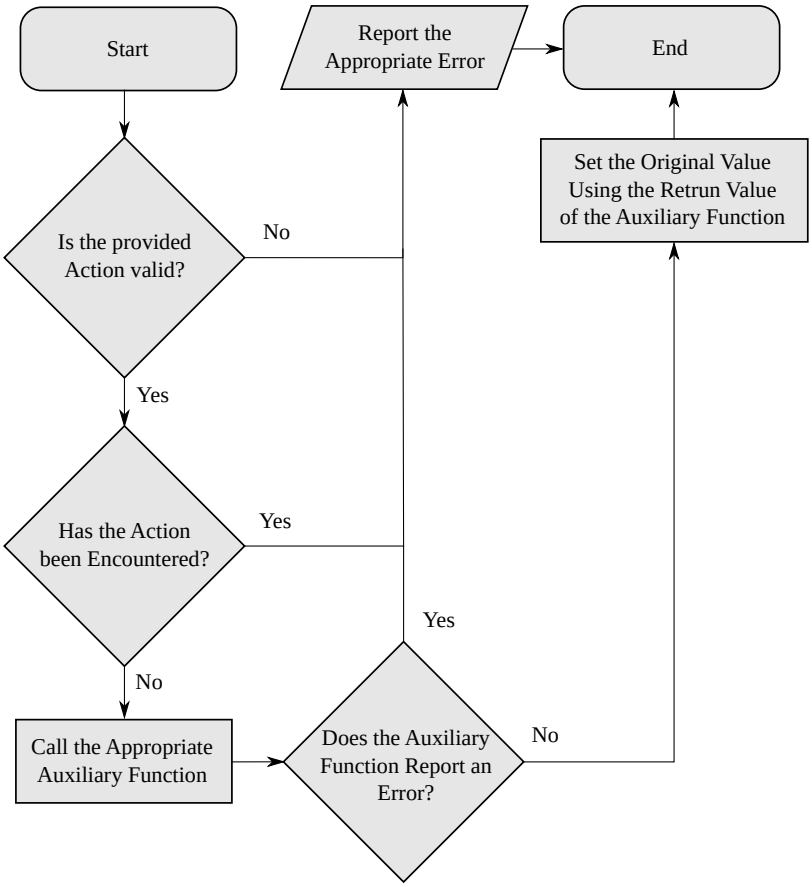


Figure 5.2: A flowchart demonstrating the sub-processor, responsible for the setting of values using auxiliary helper functions to set the various data-types, such as floats and integers in a specified range.

5.4 Finding Suitable Directories

As mentioned throughout the Analysis, it is a client-defined requirement of *furakutaru* that the System is capable of outputting multiple types of files to permanent storage. Namely, this includes a fractal configuration CSV, and a TGA export. The algorithm shown in Figure 5.3 generates a path to which the System can write, using a predefined `ATTEMPT_LIMIT` constant. In practice, this can be a reasonably low integer: less than ten.

The algorithm uses a built-in pseudorandom number generator, seeded once using the current time, to generate random numbers to append to a variable prefix and base directory, suffixed by a variable extension. This process is repeated until a non-existent path is found, or `ATTEMPT_LIMIT` is exceeded.

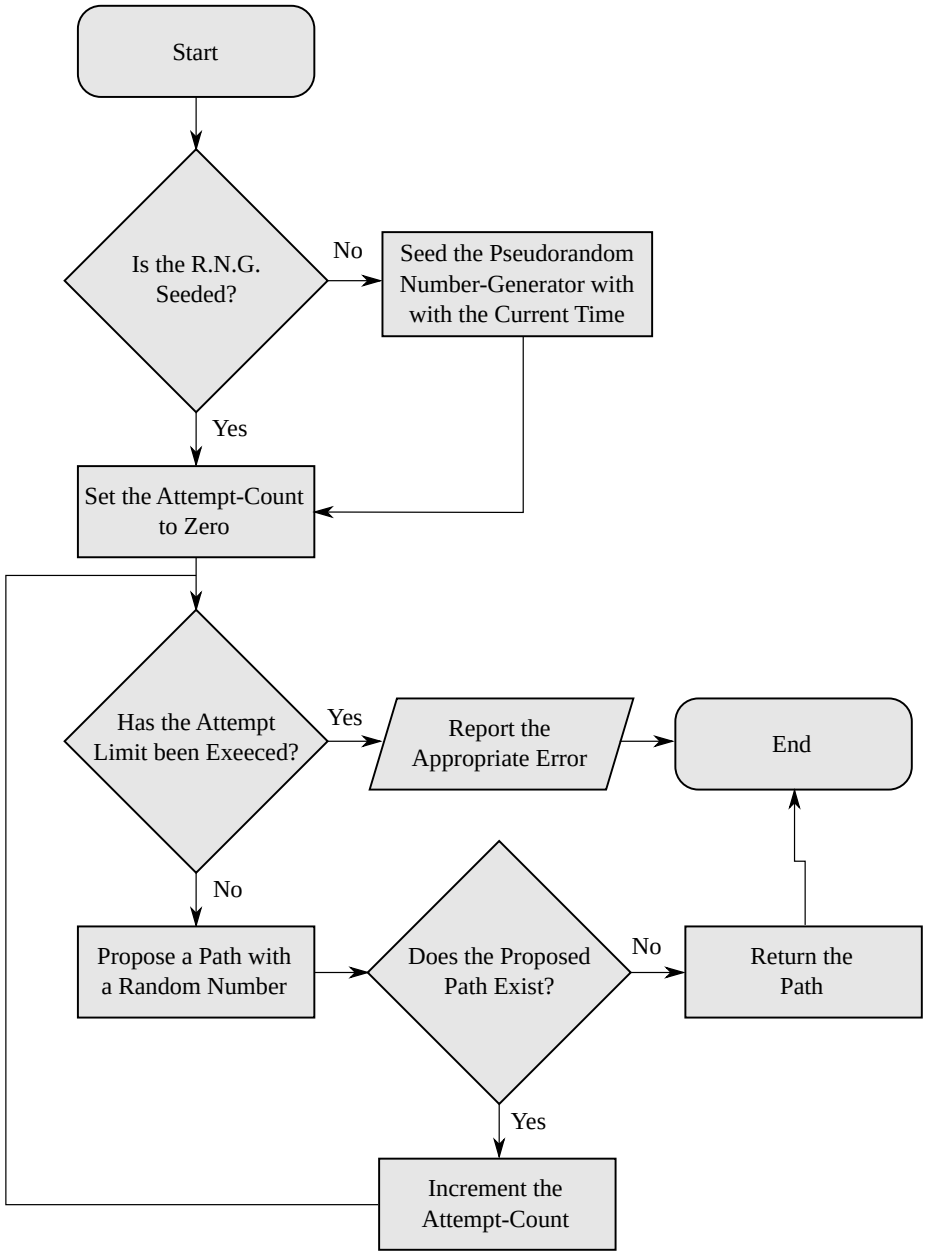


Figure 5.3: A flowchart demonstrating the *furakutaru* algorithm used to generate a pseudorandom path.

Stage III

Technical Solution

Section 6

CPU Code-Listing and Commentary

The code of *furakutaru* in its entirety is placed in the Appendices at the end of this Stage. The following Sections provide a commentary of the integral elements of the Source. This chapter concerns the *C* code, which is ultimately executed on the CPU, and does not concern direct execution on the graphics device.

Every function in the presented Source is preceded with summary briefly describing its function, arguments, return value, and any additional notes or warnings, such as instructions to the caller.

Due to the inherent issues of global variables in medium-to-large projects, the use of global variables on the stack has been entirely mitigated; *furakutaru* attempts to retain as manageable and modular as possible, and global variables do not support, but instead hinder, that effort. The only exception is granted to the very rare occasion in which *OpenGL* requires constant access to an original array of points. Globally defined symbols are restricted to pre-processor declarations, largely consisting of plain `#define` statements.

Other common symbols are exclusively signatory, and do not effect execution of the program at run-time—this includes function prototypes, structures, and enumerator type definitions.

6.1 Common Data Structures

furakutaru utilises various common data structures as a method of managing grouped values. The most prevalent concerns the state of the general renderer at some point in time, whether that be the initial state, or a current representation.

Many of the members are obvious, and are discussed at length during the Analysis, such as the maximum iterations, the centre co-ordinate, and the seed. These values are used extensively throughout *furakutaru* as a means of maintaining an asynchronous relationship with the GLSL Shaders.

Also included are the `x_disp_inf` and `x_fb_inf` structures, used primarily for the initialisation and destruction of the *OpenGL* and *X* contexts. Source-Code Listing 1 lists these structures.

```
1  /* struct x_fb_inf: information regarding the visual and frame buffer.
2   * a_pos_addr is the address of the 'a_pos' input variable in the Vertex Shader. */
3
4  struct x_fb_inf {
5          int v_id;
6          GLXFBConfig fbuf_config;
7          xcb_colormap_t colourmap;
8          GLXDrawable glx_draw;
9          GLXWindow glx_win;
10         GLXContext glx_ctx;
11         GLint a_pos_addr;
12 };
13
14 /* Currently, the only supported fractals are the Mandelbrot and Julia sets. */
15 #define FRACTAL_COUNT ( 2 )
16
17 enum fractal_type {
18         FRAC_UNDEFINED  = -1,
19         FRAC_MANDELBROT =  0,
20         FRAC_JULIA      =  1
21 };
22
23 /* struct render_state: a bridge between C and the G.L.S.L.\ Shaders; stores the
24  * state of the render at some point in time. */
25
26 struct render_state {
27         GLint max_iterations, degree, angle;
28         GLfloat scale, centre_x, centre_y, seed_x, seed_y;
29         int verbosity, locking;
30         enum fractal_type type;
31         GLuint program;
32         char * colour_path, * out_dir;
33         GLfloat rot_matrix [ 16 ]; /* 4x4 matrix */
34 };
35
```

Source-Code Listing 1: The common data structures used for grouping common attributes throughout *furakutaru*.

6.2 General Initialisation and Clean-Up

This section concerns the elements of *furakutaru* which control the initialisation of *OpenGL* and the connection to the X server using the *glX* and *XCB* interfaces. The following list presents the simplified order-of-operations which must occur to gain a valid *OpenGL*-accelerated 2-D. drawing arena.

1. Open the *X* display and provide the connection to an *XCB* port;
2. Set *XCB* as the owner of the event queue—this allows direct event-driven programming through the interface of *XCB*, entirely mitigating the need for *Xlib* post-initialisation;
3. Find and use the first valid frame-buffer configuration[1];
4. Initialise a new *OpenGL* context using the A.P.I. provided by *glX*. This initialises the *OpenGL* subsystem;
5. On some systems, a colour map must be manually created. This is to ensure the correct colour space is used;
6. Create the X window under the command of *XCB*;
7. Interfacing with *glX*, create a drawable area occupying the window where the Set renders can be drawn.

6.2.1 Initialisation Status Codes

In order to retain a common internal interface through which *furakutaru* functions can communicate, the vast majority of functions involved in the initialisation process return not an arbitrary unnamed numerical value, but a value of type ***enum*** `init_status`. As the large majority of error potential lies in the initialisation of the complex *OpenGL* A.P.I. instance, Source-Code Listing 2 presents a unified set of known errors which various initialisation functions may return, based on their status.

Regardless of the function, `STATUS_OK` represents a successful operation, and any other code indicates a failure, the latter indicating what is usually a fatal signal.

```
enum init_status {
        STATUS_OK         =   0,
        STATUS_XDISP      =  -1, /* could not open X display */
        STATUS_XCBCONN    =  -2, /* could not get X.C.B.\ connection */
        STATUS_XCBSCREEN  =  -3, /* could not find an X.C.B.\ screen */
        STATUS_FRAMEBUFF  =  -4, /* could not find any valid frame buffer
                                  * configurations */
        STATUS_OPENGLCTX  =  -5, /* could not create the OpenGL context */
        STATUS_COLOURMAP  =  -6, /* could not create the colour map */
        STATUS_WINDOW     =  -7, /* could not create the window: this could mean
                                  * an error with xcb_create_window_checked, or
                                  * an error with mapping it to the
                                  * connection */
        STATUS_OPENGLSUPP =  -8, /* OpenGL is not supported by the X server */
        STATUS_GLXWIN     =  -9, /* could not create the glX window */
        STATUS_MALLOC     = -10, /* malloc failed */
        STATUS_FREAD      = -11, /* fread did not read as much as expected */
        STATUS_FOPEN      = -12, /* fopen could not stat or open the file */
        STATUS_SHLINK     = -13, /* shader-program linking failed */
        STATUS_SHCOMP     = -14, /* shader compilation failed */
        STATUS_OPENGLEXT  = -15, /* required OpenGL extensions not supported */
        STATUS_OPENGLPROG = -16, /* OpenGL 'program' could not be created */
        STATUS_VER_ATTR   = -17  /* could not get a_pos from the V. shader */
};
```

Source-Code Listing 2: The unified Initialisation Status enumerator, designed to provide a common interface through which internal *furakutaru* functions can communicate.

Similar enumerators are used throughout *furakutaru* for the purpose of grouping status codes for a particular operation. Tightly coupled with the status enumerators are the functions which parse the code and provide a user-friendly string which accurately describes the status. Source-Code Listing 3 presents the first of such a function.

[1] With modern systems and video drivers, tens—sometimes hundreds—of display configurations are supported, often differing in resolution and refresh rate. The *RandR* extension, interfaced with by the command-line utility `xrandr` or the *OpenGL* A.P.I., is able to query the valid frame-buffer configurations on its host system.

```
1  /* parse_init_error: returns a human-readable description of an init_status
2   * code. */
3
4  static const char * parse_init_error ( enum init_status status )
5  {
6          switch ( status ) {
7                  case STATUS_XDISP:       return "could not open X display";
8                  case STATUS_XCBCONN:     return "could not get X.C.B. connection";
9                  case STATUS_XCBSCREEN:   return "could not find an X.C.B. screen";
10                 case STATUS_FRAMEBUFF:   return "could not find any valid " \
11                                                "frame buffer configurations";
12                 case STATUS_OPENGLCTX:   return "could not create the " \
13                                                "OpenGL context";
14                 case STATUS_COLOURMAP:   return "could not create the colour map";
15                 case STATUS_WINDOW:      return "could not create the window";
16                 case STATUS_OPENGLSUPP:  return "OpenGL is not supported by " \
17                                                "the current X server";
18                 case STATUS_GLXWIN:      return "could not create the glX window";
19                 case STATUS_MALLOC:      return "could not allocate memory";
20                 case STATUS_FREAD:       return "fread did not read as much " \
21                                                "as expected";
22                 case STATUS_FOPEN:       return "fopen could not stat the file";
23                 case STATUS_SHCOMP:      return "the shader could not be " \
24                                                 "compiled";
25                 case STATUS_SHLINK:      return "the shader could not be linked";
26                 case STATUS_OPENGLEXT:   return "the current OpenGL does not " \
27                                                "support the required "\
28                                                "extensions";
29                 case STATUS_OPENGLPROG:  return "the OpenGL 'program' could " \
30                                                "not be created";
31                 case STATUS_VER_ATTR:    return "the vertex shader is invalid";
32                 default:                 return "unknown error";
33         }
34 }
35
```

Source-Code Listing 3: The parser for a `init_status` enumerator code, able to return a human-readable string.

6.2.2 The Initialisation Assistant: `init_assistant`

The vast majority of the initialisation code is placed in the `init_assistant` function. This is used to not only keep the entry-point function to a manageable size, but also to reduce the stack size of the program during execution, as disposable, temporary variables used only for initialisation can be discarded when they are no longer required[2]. Source-Code Listing 4 lists the sources for the helper function. As it produces custom error messages directly to an output device—`stdout` or `stderr`—it can only return a binary success-fail indicator, the latter acting as an indicator for a fatal, unrecoverable status.

6.2.3 *XCB* Initialisation: `init_xcb`

furakutaru prefers *XCB* to *Xlib* as an interface to the *X* Windowing System, as *Xlib* is archaic, outdated, and bloated. Modern applications written in *XCB* enjoy a low-footprint and common programming interface through which they can confer with the *X* Server implementation.

Unfortunately, the *X11-OpenGL* bridge, *glX*, has a hard requirement of *Xlib* for the purposes of initialisation, and a pure *XCB* solution would mandate a full re-write of *glX*, and possibly some parts of *OpenGL* or the *X* System. Because of this, *furakutaru* employs the following stages for initialisation, aimed at satisfying the ordinances of *glX*, whilst minimising the effect of *Xlib*.

The first function performed by the Initialisation Assistant creates a direct connection to the *X* server by opening a Display, and handing the connection to the custody of *XCB*, allowing the latter to manage the event queue. Source-Code Listing 5 lists the `init_xcb` function.

This function can take an initialised or uninitialised structure as its argument, as all the members are initially set to `NULL`[3], or the appropriate equivalent for its datatype. The connection is retrieved, and the *XCB* Event Manager is set to the owner of the general Server Event Queue—this is achieved by the *Xlib* built-in `XSetEventQueueOwner`. This function conforms with the Common Status Interface outlined in Source-Code Listing 2.

[2]This is important due to the heavy usage of the stack at other points during the initialisation, such as allocating buffers of up to 4096 characters long.

[3]In every implementation of the standard library, `NULL` is defined to be zero cast to a void pointer: `#define NULL ( void * )0`.

```
/* [ exposed ] init_assistant: initialise all the auxiliary elements of the
 * rendering system --- X, OpenGL, Shaders, and the Texture. Returns -1 on
 * error, zero on success. If an error is thrown, there is no need for callers
 * to perform manual clean-ups. */

int init_assistant ( struct x_disp_inf * disp_inf, struct x_fb_inf *
                fb_inf, GLuint program_list [ FRACTAL_COUNT ],
                const char * colour_path )
{
        enum init_status status = STATUS_OK;
        enum ppm_status ppm_stat = PPM_OK;

        fb_inf->a_pos_addr = -1;

        /* initialise X.C.B.\ and OpenGL */
        if ( ( status = init_xcb ( disp_inf ) ) != STATUS_OK ) {
                fprintf ( stderr, "X.C.B. initialisation error %d: %s\n",
                                status, parse_init_error ( status ) );
                return -1;
        }

        if ( ( status = init_opengl ( disp_inf, fb_inf ) ) != STATUS_OK ) {
                XCloseDisplay ( disp_inf->disp );
                fprintf ( stderr, "OpenGL initialisation error %d: %s\n",
                                status, parse_init_error ( status ) );
                return -1;
        }

        /* initialise the OpenGL shaders and get access to the vertex
         * position */
        if ( ( status = init_shaders ( program_list )) != STATUS_OK ||
                        ( status = get_vertex_attribute (
                                        *program_list,
                                        & ( fb_inf->a_pos_addr ) ) )
                        != STATUS_OK ) {
                fprintf ( stderr, "Shader initialisation error %d: %s\n",
                                status, parse_init_error ( status ) );
                clean_all ( disp_inf, fb_inf, program_list );
                return -1;
        }

        /* load the colour texture */
        if ( ( ppm_stat = load_texture ( ( colour_path ) ? colour_path :
                                COLOUR_FILE_DEFAULT ) ) == -1 ) {
                fprintf ( stderr, "Texture error %d: %s\n", ppm_stat,
                                parse_ppm_error ( ppm_stat ) );
                clean_all ( disp_inf, fb_inf, program_list );
                return -1;
        }

        return 0;
}

```

Source-Code Listing 4: The general helper function for initialising the sub-components of the System, including X, *OpenGL*, and the connection between them.

This function is very unlikely to fail, and would only usually return `STATUS_XDISP` if it was executed from a virtual teletypewriter (`tty`) or pseudo-terminal (`pts`) without the `DISPLAY` shell variable appropriately set.

6.2.4 Find an Appropriate *X* Screen: `find_xcb_screen`

As with *Xlib*, *XCB* also mandates that an appropriate *X* screen is selected[4]. *furakuraru* automatically assumes the first valid screen as the preference. Source-Code Listing 6 calculates the first valid screen, from a list provided by *Xlib*, and provides *XCB* with its respective I.D.

This function returns -1 if there are no appropriate screens, or zero otherwise. In the Common Status Interface, this correlates to the `STATUS_XCBSCREEN` and `STATUS_OK` codes respectively.

6.2.5 *OpenGL* Initialisation: `init_OpenGL`

Assuming the success of the *XCB* initialisation, the initialisation assistant proceeds to create a software-bridge between the *X* connection and the hardware-implemented *OpenGL* interface. This

[4] In the context of *X Windows*, a "screen" refers to a sub-instance of the *X* server, and not the common assumption of a different monitor—an arbitrary number of monitors can run on a single screen, and it is very rare that a user would run multiple screens simultaneously. Because of this, *furakutaru* regards the first valid screen as the rendering target.

```
/* init_xcb: initialise the X.C.B.instance with the display from Xlib. Returns
 * STATUS_OK on success, anything else is an error. */

static enum init_status init_xcb ( struct x_disp_inf * disp_inf )
{
        const char * disp_id = NULL;

        disp_inf->screen_id = -1;
        disp_inf->disp = NULL;
        disp_inf->xcb_screen = NULL;
        disp_inf->xcb_conn = NULL;

        /* Open Xlib display. Although X.C.B.\ is preferred throughout, the
         * initialisation process for OpenGL *requires* Xlib. */

        if ( ( disp_inf->disp = XOpenDisplay ( disp_id ) ) == NULL )
                return STATUS_XDISP;

        /* Grab the X.C.B.\ connection from the Xlib display and set it as the
         * owner of the event queue. */

        if ( ( disp_inf->xcb_conn = XGetXCBConnection ( disp_inf->disp ) )
                        == NULL ) {
                XCloseDisplay ( disp_inf->disp );
                return STATUS_XCBCONN;
        }

        XSetEventQueueOwner ( disp_inf->disp, XCBOwnsEventQueue );
        if ( find_xcb_screen ( disp_inf ) == -1 ) {
                XCloseDisplay ( disp_inf->disp );
                return STATUS_XCBSCREEN;
        }

        return STATUS_OK;
}

```

Source-Code Listing 5: Establishes a connection between a new *Xlib-X* connection and the *XCB* interface.

```
/* find_xcb_screen: finds and sets the X.C.B.\ screen from the Display,
 * provided by Xlib. Returns -1 on error, 0 on success. */

static int find_xcb_screen ( struct x_disp_inf * disp_inf )
{
        int screen_id = DefaultScreen ( disp_inf->disp );
        xcb_screen_iterator_t si =
                xcb_setup_roots_iterator (
                                xcb_get_setup ( disp_inf->xcb_conn )
                );

        /* iterate down the pool of potential screens */
        while ( si.rem && screen_id > 0 ) {
                xcb_screen_next ( &si );
                screen_id--;
        }

        disp_inf->screen_id = screen_id;
        disp_inf->xcb_screen = si.data;

        return ( !si.data ) ? -1 : 0;
}

```

Source-Code Listing 6: Detecting the first valid *X* screen, and providing its I.D. to the *XCB* instance.

includes querying the contenders for a plausible frame-buffer, and instantiating other appropriate properties on the *X* window; this is shown in Source-Code Listing 7. The function also ensures that the *X* server supports the *OpenGL* Extension, as older or non-accelerated Servers cannot be assumed to have this capability.

6.2.6 Locating an Appropriate Frame-Buffer: `find_valid_fb`

As mentioned in the introduction, modern video cards, since the period of Cathode Ray-tube monitors and T.V.s, have many different modes, many of which differ on resolution and refresh rate. They often taking the limiting factor of the V.D.U. hardware itself, but also the cable over which the signal is to be transmitted—for example, H.D.M.I. is incapable of transmitting 4K video in almost

```
1  /* init_opengl: initialises a window with the OpenGL context. Returns STATUS_OK on
2   * success, and something else on failure. */
3
4  static enum init_status init_opengl ( struct x_disp_inf * disp_inf,
5                  struct x_fb_inf * fb_inf )
6  {
7          enum init_status status = STATUS_OK;
8          int sup_dum = 0; /* dummy variable for querying OpenGL support from X */
9          fb_inf->glx_ctx = 0;
10
11         if ( find_valid_fb ( disp_inf, fb_inf ) == -1 )
12                 return STATUS_FRAMEBUFF;
13
14         /* Create the OpenGL context and colour map */
15
16         if ( !glXQueryExtension ( disp_inf->disp, &sup_dum, &sup_dum ) )
17                 return STATUS_OPENGLSUPP;
18
19         fb_inf->glx_ctx = glXCreateNewContext (
20                                 disp_inf->disp,
21                                 fb_inf->fbuf_config,
22                                 GLX_RGBA_TYPE,
23                                 0, True
24                         );
25
26         if ( !fb_inf->glx_ctx )
27                 return STATUS_OPENGLCTX;
28
29         if ( create_colourmap ( disp_inf, fb_inf ) == -1 ) {
30                 glXDestroyContext ( disp_inf->disp, fb_inf->glx_ctx );
31                 return STATUS_COLOURMAP;
32         }
33
34         /* Create the window and set up OpenGL */
35
36         if ( create_window ( disp_inf, fb_inf ) == -1 ) {
37                 xcb_free_colormap ( disp_inf->xcb_conn, fb_inf->colourmap );
38                 glXDestroyContext ( disp_inf->disp, fb_inf->glx_ctx );
39                 return STATUS_WINDOW;
40         }
41
42         if ( ( status = init_glx ( disp_inf, fb_inf ) ) != STATUS_OK ) {
43                 xcb_destroy_window ( disp_inf->xcb_conn, disp_inf->win );
44                 xcb_free_colormap ( disp_inf->xcb_conn, fb_inf->colourmap );
45                 glXDestroyContext ( disp_inf->disp, fb_inf->glx_ctx );
46
47                 return status;
48         }
49
50         return STATUS_OK;
51 }
52
```

Source-Code Listing 7: Creates a bridge between the *X* connection and *OpenGL*.

all circumstances, however Display Ports are capable of transmitting up to an 8K signal.

It is therefore the role of any graphics application to query the *X* server to determine an appropriate frame-buffer mode on which to transmit. The first encountered is almost always the correct choice, as it is the one which the *X* server is assuming, and transmitting on a different mode to the Server can cause grave issues—C.R.T. V.D.U.s could even become permanently damaged.

Source-Code Listing 8 does not conform to the Common Status Interface used within *furakutaru*, as its return value is a binary choice: success or failure. The status code `STATUS_FRAMEBUFF` is exclusively reserved for status-returning callers that wish to redirect an error to their respective parent. If the search is successful, a `v_id`, or 'visual I.D.', is placed in the `x_fb_inf` structure which was passed as the second argument.

The final `free` call is included due to the fact that many run-time memory profilers will report an potential leakage from the `glXGetFBConfigs` function, however the proprietary nature of *NVIDIA*'s *glX* implementation means debuggers cannot step into its subsequent `free` call[5].

6.2.7 Creating the Colour-Mapping: `create_colourmap`

This function is not required for most systems, as *OpenGL* is able to provide hints to the *glX* A.P.I. regarding the supported colour-spaces. However, it is good practice to ensure the manual cre-

[5] A call-stack analyser, such as `callgrind`, can detect and mitigate this risk. `callgrind` reports that `glXGetFBConfigs` makes a call to `XextCreateExtension`—a part of the open-sourced *X* Server implementation—which calls a `malloc`. From this, it can be seen that its clean-up counterpart, `XextDestroyExtension`, which calls a subsequent `XFree`, is eventually called. See https://opensource.apple.com/source/X11/X11-0.40/xc/lib/Xext/extutil.c for more information regarding the latter complaint.

```
/* find_valid_fb: finds the first valid frame buffer configuration from the
 * display in disp_inf and places it in fbuf_config. Returns 0 on success, -1 on
 * failure. */

static int find_valid_fb ( struct x_disp_inf * disp_inf,
                struct x_fb_inf * fb_inf )
{
        XVisualInfo * visual_inf = NULL;
        int fbuf_config_count = 0;

        fb_inf->v_id = 0;
        fb_inf->fbuf_config = NULL;

        /* find all frame buffer configurations from OpenGL */
        GLXFBConfig * fbuf_configs = glXGetFBConfigs (
                        disp_inf->disp,
                        disp_inf->screen_id,
                        &fbuf_config_count );

        if ( !fbuf_configs || fbuf_config_count == 0 )
                return -1;

        /* get the first valid frame buffer configuration; STATUS_FRAMEBUFF is
         * returned if there does not exist such a configuration */

        for ( int i = 0; fb_inf->v_id == 0 && i < fbuf_config_count; ++i ) {
               fb_inf->fbuf_config = fbuf_configs[i];
               if ( ( visual_inf = glXGetVisualFromFBConfig
                                        ( disp_inf->disp,
                                                fb_inf->fbuf_config ) ) == NULL )
                       continue;
               fb_inf->v_id = visual_inf->visualid;
               XFree ( visual_inf );
        }

        free ( fbuf_configs );
        return ( fb_inf->v_id == 0 ) ? -1 : 0;
}

```

Source-Code Listing 8: The `find_valid_fb` function querying the *X* server for various contenders for the appropriate frame-buffer mode.

ation of a colour map, as it ensures complete portability across older devices and video-transmission standards implementing older *OpenGL* versions [Fou09]. This function is listed in Source-Code Listing 9.

```
/* create_colourmap: creates a colour map with the given X.C.B.\ connection and
 * screen. Returns -1 on error, and 0 on success. The colour map is placed in
 * fb_inf.colourmap. */

static int create_colourmap ( struct x_disp_inf * disp_inf,
                struct x_fb_inf * fb_inf )
{
        fb_inf->colourmap = xcb_generate_id ( disp_inf->xcb_conn );
        xcb_generic_error_t * error = NULL;

        error = xcb_request_check ( disp_inf->xcb_conn,
                        xcb_create_colormap_checked (
                                disp_inf->xcb_conn,
                                XCB_COLORMAP_ALLOC_NONE,
                                fb_inf->colourmap,
                                disp_inf->xcb_screen->root,
                                fb_inf->v_id
                        )
                );

        if ( error ) {
                free ( error );
                return -1;
        }

        return 0;
}

```

Source-Code Listing 9: Create a colour-mapping for the systems which cannot automatically create one for the *OpenGL* context, namely those with less than sixteen-million colours.

6.2.8 Creating the Main Window: `create_window`

`create_window`, as shown in Source-Code Listing 10, creates a window under *XCB* and subscribes it to certain events, as specified in `event_mask`. *XCB* provides twenty-five options to which a window can subscribe; in this case, `XCB_EVENT_MASK_KEY_PRESS` and `XCB_EVENT_MASK_EXPOSURE`, used for detecting key-presses and window-invalidations respectively, are the only events required for a keyboard-only application.

The function then provides basic hints to the Window Manager as the window is being created, such as the title, ideal starting positions—half the available screen dimensions, for stacking and similar W.M.s—, and dimensions. It then ensures that the window is visible, to mitigate the incompetencies of some Window Managers, such as *Gnome*, which have severe difficulties managing windows and ensuring their visibility.

6.2.9 Binding the *XCB* Window to an *OpenGL* Subsystem: `init_glx`

The *glX* provides a bridge-like A.P.I. between the *X* Server and *OpenGL* subsystem. `init_glx`, shown in Source-Code Listing 11, provides a wrapper for the functions `glXCreateWindow` and `glXMakeContextCurrent`, intended for *OpenGL*-window binding and context-setting respectively.

Throughout the source of *furakutaru*, functions prefixed with `glX` are exclusively referring to the rendering context which has been bound to the *X* window, and do not directly concern the relationship between the application and window manager. Similar to other wrapper/helper initialisation functions, this function does not conform to the Common Status Interface, as its success is determined by a binary value; the code `STATUS_GLXWIN` exists for the assistance of parent functions which wish to report their own point of failure.

This function also instantiates and makes use of the *OpenGL Extension-Wrangler Library*, or *GLEW* Serving to the robustness of *furakutaru*, this is used to query the local *OpenGL* Server for support of various functionality, as the System is designed to be as backwardly compatible as possible. The `GLEW_ARB_vertex_shader` test ensures that the *OpenGL* implementation supports Vertex Shaders—a point discussed in-depth later.

6.2.10 Preventing Memory Leaks with Memory-Management: `clean_all`

The `clean_all` function, listed in Source-Code Listing 12, frees all the memory used by the initialisation process outlined above. It should be called immediately before the exit of *furakutaru*, under the vast majority of circumstances—the notable exception being if the initialisation process failed extremely early for unusual version, e.g. if the *X* server was not running.

6.3 Initialisation of the *GLSL* Shaders

As outlined in the Design section, Shader programs are C-like auxiliary programs used by *OpenGL* for parallel execution across homogeneous GPU cores, causing a significant performance increase for all platforms with a dedicated or integrated graphics unit. The various functions outlined in this section share the Common Status Interface for the initialisation section, as the vast majority of the operations are akin.

6.3.1 Initialising and Linking the Shader Programs: `init_shaders`

The *OpenGL* shaders are divided into two primary categories: Fragment and Vertex Shaders. Fragment Shaders perform the vast majority of the calculations, being responsible for the plotting and colouring of the pixels. The Vertex Shader, on the other hand, generates an appropriate transform matrix which is utilised by the Fragment Shader to determine the viewport in which to render pixels.

Although a Vertex Shader is not strictly required, and *OpenGL* will use an implicit identity matrix to perform no extraneous transformation, the rotation feature of *furakutaru* mandates a basic transformation to be performed, using a standard four-by-four rotation matrix.

Because of this, the Vertex Shader requires linkage to each of the Fragment Shaders. In *furakutaru*, this implies linking the Mandelbrot Set Fragment Shader and Vertex Shader to one *OpenGL* program, and the Julia Set Fragment Shader and Vertex Shader to another, where the Vertex Shader is common throughout.

```
/* create_window: Creates the window using the OpenGL context. Returns -1 on
 * error, zero on success. */

/* The window is subscribed to the following events:
 *  - XCB_EVENT_MASK_KEY_PRESS [ key-press ];
 *  - XCB_EVENT_MASK_EXPOSURE [ needs to be re-drawn ] */

static int create_window ( struct x_disp_inf * disp_inf,
                struct x_fb_inf * fb_inf )
{
        const uint32_t event_mask = XCB_EVENT_MASK_KEY_PRESS |
                XCB_EVENT_MASK_EXPOSURE;
        const uint32_t values[] = { event_mask, fb_inf->colourmap };
        const uint32_t value_mask = XCB_CW_EVENT_MASK | XCB_CW_COLORMAP;
        xcb_generic_error_t * error = NULL;

        /* Create the window, settings its initial dimensions to half that of
         * the screen. This will only have an effect with some windowing
         * environments, such that the window size is not pre-determined (e.g.
         * most stacking W.M.s). */

        disp_inf->win_width = disp_inf->xcb_screen->width_in_pixels / 2;
        disp_inf->win_height = disp_inf->xcb_screen->height_in_pixels / 2;

        disp_inf->win = xcb_generate_id ( disp_inf->xcb_conn );
        error = xcb_request_check ( disp_inf->xcb_conn,
                        xcb_create_window_checked (
                                disp_inf->xcb_conn,
                                XCB_COPY_FROM_PARENT,
                                disp_inf->win,
                                disp_inf->xcb_screen->root,
                                WIN_STARTX, WIN_STARTY,
                                disp_inf->win_width,
                                disp_inf->win_height, 0,
                                XCB_WINDOW_CLASS_INPUT_OUTPUT,
                                fb_inf->v_id,
                                value_mask,
                                values
                        )
                );

        /* set the window title to WIN_TITLE */
        xcb_change_property ( disp_inf->xcb_conn, XCB_PROP_MODE_REPLACE,
                        disp_inf->win, XCB_ATOM_WM_NAME, XCB_ATOM_STRING,
                        8, strlen ( WIN_TITLE ), WIN_TITLE );

        if ( error ) {
                free ( error );
                return -1;
        }

        /* ensure the window is visible */
        error = xcb_request_check (
                        disp_inf->xcb_conn,
                        xcb_map_window (
                                disp_inf->xcb_conn,
                                disp_inf->win
                        )
                );

        if ( error ) {
                xcb_destroy_window ( disp_inf->xcb_conn, disp_inf->win );
                free ( error );
                return -1;
        }

        return 0;
}

```

Source-Code Listing 10: Creates an *XCB* window using the hints provided. This is not yet linked to *OpenGL*, and is the standard routine for creating an *XCB* window when utilising the *Xlib* bridge.

The `init_shaders` function, shown in Source-Code Listing 13, performs this elegantly, loading the Vertex Shader once, and linking to each Fragment Shader as they are loaded sequentially.

6.3.2 Loading the Auxiliary Files into a Buffer: `populate_buffer`

Loaded dynamically to facilitate the creation of custom fractals by advanced users, the function `populate_buffer`, as shown in Source-Code Listing 14, dynamically allocates a buffer and fills it with the file specified file. As explained in the function description, the provided buffer is only expected to be freed by the caller if the function succeeds, as `populate_buffer` frees the buffer itself

```
/* init_glx: initialises the glX to a window and sets the OpenGL context.
 * Returns -1 on error and 0 on success. */

static enum init_status init_glx ( struct x_disp_inf * disp_inf,
                struct x_fb_inf * fb_inf )
{
        fb_inf->glx_win = glXCreateWindow ( disp_inf->disp,
                                fb_inf->fbuf_config,
                                disp_inf->win, 0
                        );

        if ( fb_inf->glx_win == 0 )
                return -1;

        fb_inf->glx_draw = fb_inf->glx_win;

        if ( glXMakeContextCurrent ( disp_inf->disp, fb_inf->glx_draw,
                                fb_inf->glx_draw, fb_inf->glx_ctx ) == False ) {
                glXDestroyWindow ( disp_inf->disp, fb_inf->glx_win );
                return STATUS_GLXWIN;
        }

        glewInit ( );

        if ( !GLEW_ARB_vertex_shader ) {
                glXDestroyContext ( disp_inf->disp, fb_inf->glx_ctx );
                return STATUS_OPENGLEXT;
        }

        return STATUS_OK;
}

```

Source-Code Listing 11: The binding of an *OpenGL* drawing instance to the *furakutaru* window.

```
/* [ exposed ] clean_all: de-allocate all memory taken by the standard X and
 * OpenGL functions */

void clean_all ( struct x_disp_inf * disp_inf, struct x_fb_inf * fb_inf,
                 GLuint program_list [ FRACTAL_COUNT ] )
{
        if ( program_list )
                for ( int i = 0; i < FRACTAL_COUNT; i++ )
                        glDeleteProgram ( program_list [ i ] );

        glXDestroyWindow ( disp_inf->disp, fb_inf->glx_win );
        xcb_destroy_window ( disp_inf->xcb_conn, disp_inf->win );
        xcb_free_colormap ( disp_inf->xcb_conn, fb_inf->colourmap );
        glXDestroyContext ( disp_inf->disp, fb_inf->glx_ctx );
        XCloseDisplay ( disp_inf->disp );
}

```

Source-Code Listing 12: The complete clean-up of *furakutaru*, ensuring there are no memory leaks. This has been verified extensively with `valgrind`'s `memcheck` utility.

if the loading fails, as it is expected the buffer contents will be of no further use.

6.3.3 Compiling the Shader: `setup_shader`

As the shaders are loaded dynamically, they also mandate dynamic compilation. As shaders are typically small, this is instantaneous on even the slowest of processors. Taking a buffer of shader source and a destination shader I.D., the `setup_shader` function, listed in Source-Code Listing 15, attempts to compile the shader. This function conforms to the Common Status Interface, and thus returns `STATUS_SHCOMP` to indicate a shader-compilation error, or `STATUS_OK` to indicate success.

6.3.4 Debugging the Compile-Link Procedure: `print_gl_log`

OpenGL provides a slightly unintuitive interface for checking the status. This can be done globally, providing there is a valid instance, as *OpenGL* is a state- machine. In many cases, under normal circumstances of operation, this function will never be called, however it is extraordinarily useful for debugging and reporting an out-of-date *OpenGL* version to the user.

The `print_gl_log` function, shown in Source-Code Listing 16, directly prints the log to the standard error output device. Through the use of function pointers, this function has been kept relatively concise.

```
/* init_shaders: initialise FRACTAL_COUNT programs, each with their respective
 * fragment shader, and a common vertex shader. Returns the appropriate
 * init_status; prints the OpenGL logs if required, or if PRINT_FULL_GL_LOG is
 * defined. */

static enum init_status init_shaders ( GLuint program_list [ FRACTAL_COUNT ] )
{
        GLchar * frag_buf = NULL, * vert_buf = NULL;
        GLuint frag_sdr_id = 0,
               vert_sdr_id = glCreateShader ( GL_VERTEX_SHADER );
        enum init_status status = STATUS_OK;
        const char * vert_shader_path = AUX_DIR "/shaders/vertex.glsl",
                * frag_shader_paths [ FRACTAL_COUNT ] = {
                        AUX_DIR "/shaders/mandelbrot.glsl",
                        AUX_DIR "/shaders/julia.glsl"
                };

        /* load the common vertex shader */
        if ( ( status = populate_buffer ( &vert_buf, vert_shader_path ) )
                        != STATUS_OK ||
             ( status = setup_shader ( ( GLchar ** )&vert_buf, &vert_sdr_id ) )
                        != STATUS_OK ) {
                glDeleteShader ( vert_sdr_id );
                free ( vert_buf );
                return status;
        }

        /* load all the fragment shaders, assigning a new program for each,
         * whilst linking the common vertex shader to each one */

        for ( int i = 0; i < FRACTAL_COUNT; i++ ) {
                program_list [ i ] = glCreateProgram ( );
                frag_sdr_id = glCreateShader ( GL_FRAGMENT_SHADER );

                if ( ( status = populate_buffer ( &frag_buf,
                                                  frag_shader_paths [ i ] ) )
                                != STATUS_OK ||
                     ( status = setup_shader ( ( GLchar ** )&frag_buf,
                                               &frag_sdr_id ) )
                                != STATUS_OK ) {
                        glDetachShader ( program_list [ i ], frag_sdr_id );
                        glDeleteShader ( frag_sdr_id );
                        glDeleteShader ( vert_sdr_id );

                        free ( frag_buf );
                        free ( vert_buf );
                        return status;
                }

                /* attach and link the fragment and vertex shader */
                glAttachShader ( program_list [ i ], vert_sdr_id );
                glAttachShader ( program_list [ i ], frag_sdr_id );
                glLinkProgram ( program_list [ i ] );

#ifdef PRINT_FULL_GL_LOG
                if ( print_gl_log ( program_list [ i ],
                                    GL_LINK_STATUS, 1 ) == -1 ) {
#else
                if ( print_gl_log ( program_list [ i ],
                                    GL_LINK_STATUS, 0 ) == -1 ) {
#endif
                        glDeleteShader ( vert_sdr_id );

                        free ( frag_buf );
                        free ( vert_buf );
                        return STATUS_SHLINK;
                }

                glDetachShader ( program_list [ i ], frag_sdr_id );
                glDetachShader ( program_list [ i ], vert_sdr_id );
                glDeleteShader ( frag_sdr_id );
                free ( frag_buf );
        }

        glDeleteShader ( vert_sdr_id );
        free ( vert_buf );
        return STATUS_OK;
}
```

Source-Code Listing 13: The initialisation of the Shaders, linking each Fragment Shader to a separate *OpenGL* program, with a common Vertex Shader.

If the compile-time flag `PRINT_FULL_GL_LOG` is enabled, all the warnings from *OpenGL* are displayed, even if it does not prevent compilation and normal usage.

```
/* populate_buffer: allocate some memory enough to store the entirety of the
 * file referenced by 'path', and then populate the buffer. Returns STATUS_OK on
 * success, and either STATUS_MALLOC or STATUS_FREAD on error. If STATUS_OK is
 * reported, 'buf' is expected to be freed by the caller. */

static enum init_status populate_buffer ( char ** buf, const char * path )
{
        FILE * fp = NULL;
        unsigned long size = 0;

        if ( ( fp = fopen( path, "r" ) ) == NULL )
                return STATUS_FOPEN;

        fseek ( fp, 0, SEEK_END );
        size = ftell ( fp );
        fseek ( fp, 0, SEEK_SET );

        if ( ( *buf = calloc ( size + 1, sizeof ( char ) ) ) == NULL ) {
                fclose ( fp );
                return STATUS_MALLOC;
        }

        if ( fread ( *buf, sizeof ( char ), size, fp ) < size ) {
                fclose ( fp );
                free ( *buf );

                return STATUS_FREAD;
        }

        fclose ( fp );
        return STATUS_OK;
}

```

Source-Code Listing 14: Loads a shader into a dynamically allocated buffer.

```
/* setup_shader: sets up the shader, the characters of which are already loaded
 * into data. The appropriate init_status is returned. This function frees and
 * nulls its first argument. */

static enum init_status setup_shader ( GLchar ** data, GLuint * sdr_id )
{
        glShaderSource ( *sdr_id, 1, ( const GLchar * const * )data, NULL );
        glCompileShader ( *sdr_id );

        free ( *data );
        *data = NULL;

        return ( print_gl_log ( *sdr_id, GL_COMPILE_STATUS, 0 ) == -1 ) ?
                STATUS_SHCOMP : STATUS_OK;
}

```

Source-Code Listing 15: Dynamically compiles a shader loaded from a file.

6.3.5 Retrieving the Vertex Shader Position: `get_vertex_attribute`

The function `get_vertex_attribute`, detailed in Source-Code Listing 17, is a robust mechanism for retrieving the position of the `a_pos` attribute in the Vertex Shader. It also serves as a method of effectively defining the viewport to ensure no amount of rotation can clip the viewport. As $\pi/4$ radians, or 45 degrees is the most extreme level of deviation from zero, the usual vertices of the viewport have been multiplied[6] by the factor $1/\sin(\pi/4)$.

6.4 Processing the PPM Colour File

As mentioned in the Design section, the use of one-dimensional images as colour palettes are used as customisable textures in *furakutaru*. PPM is an extremely simple image format for reading, and can easily be created using many popular image-manipulation tools, such as *The G.I.M.P.*

6.4.1 PPM Common Status Codes and Parsers: `parse_ppm_error`

Similar to the initialisation status codes, the PPM loader makes similar use of enumerators. A parser also exists to generate human-readable strings from the various error codes which can be

[6] This creates a rendering arena which is larger than the window, meaning that unless the user is rotated at $\pi/4$, or equivalent, the renderer could be drawing pixels which are wasted. Modern *OpenGL* however, has the capability to detect this and only draw pixels when within the window of which it the rendering area is a child.

```
/* print_gl_log: prints the OpenGL for the specified operation, 'target'. If
 * this is not set to GL_COMPILE_STATUS or GL_LINK_STATUS, 1 is returned. If the
 * specified operation failed, or 'force' is set, the appropriate log is printed
 * to stderr. In the case of 'force' not being set but the operation failing, -1
 * is returned. On success, zero is returned. */

static int print_gl_log ( GLuint id, GLenum target, int force ) {
        GLint res = GL_FALSE;
        GLsizei len = 0;
        GLchar * log = NULL;

        void ( *metaquery_func ) ( GLuint, GLenum, GLint * ) = NULL;
        void ( *query_func ) ( GLuint, GLsizei, GLsizei *, GLchar * ) = NULL;

        switch ( target ) {
                case GL_COMPILE_STATUS:
                        metaquery_func = glGetShaderiv;
                        query_func = glGetShaderInfoLog;
                        break;
                case GL_LINK_STATUS:
                        metaquery_func = glGetProgramiv;
                        query_func = glGetProgramInfoLog;
                        break;
                default:
                        return 1;
        }

        metaquery_func ( id, target, &res );

        /* check the success of the operation */
        if ( res == GL_FALSE || force ) {
                metaquery_func ( id, GL_INFO_LOG_LENGTH, &len );
                if ( len > 0 && ( log = malloc ( sizeof ( char ) *
                                                 ( len + 1 ) ) ) ) {
                        /* print an appropriate log */
                        query_func ( id, len, NULL, log );
                        log [ len ] = '\0';

                        fputs ( log, stderr );
                        fputc ( '\n', stderr );
                        free ( log );
                }

                return ( force ) ? ( ( res == GL_FALSE ) ? -1 : 0 ) : -1;
        }

        return 0;
}

```

Source-Code Listing 16: A reporting mechanism from the *OpenGL* subsystem to the user.

thrown, as shown in Source-Code Listing 18.

6.4.2 Loading the Texture: `load_texture`

The parent-most function, `load_texture`, is instantiated by the Initialisation Assistant once upon the initialisation of *furakutaru*. This loads the colour texture and sends it to the *OpenGL* instance, for the length of program-execution. This function conforms to the Common PPM Status Interface, shown previously in Source-Code Listing 18.

6.4.3 Loading the Image from the File: `load_image`

The `load_image` function, as shown in Source-Code Listing 20, wraps the various functions used in order to load parse the header and pixel data from a PPM file. Also conforming to the Common PPM Status Interface, this function passes the return value sent by one of its auxiliary functions, or `PPM_OK` on success.

`load_image` also performs a basic check on the PPM file being passed, to ensure it is not corrupted and is intended as a PPM file. This is achieved by checking the first two bytes of the file, ensuring they match the expected values of `P6`, as defined by the PPM format specification [The16]. Unworthy of its own heading, the function `verify_ppm_sig` is listed in Source-Code Listing 21.

6.4.4 Parsing the PPM Header Section: `read_hdr`

Well-defined in [The16], the header is a simple-to-parse and unambiguous structure, the only difficulty being posed by the inclusion of comments. The `read_hdr` function, shown in Source-Code

```
/* A rotation of pi/4 is the most severe clipping that can occur */
#define CLIP_REDUCTION ( 1.414213562 ) /* 1 / sin ( pi/4 ) */

/* viewport_vertices: the vertices of each "quad" in the viewport. This needs to
 * be global, as OpenGL needs reliable and unprovoked access to it throughout
 * the execution of the program. */

const GLfloat viewport_vertices [ ] = {
        -1.0 * CLIP_REDUCTION,  1.0 * CLIP_REDUCTION,
        -1.0 * CLIP_REDUCTION, -1.0 * CLIP_REDUCTION,
         1.0 * CLIP_REDUCTION,  1.0 * CLIP_REDUCTION,
         1.0 * CLIP_REDUCTION, -1.0 * CLIP_REDUCTION
};

/* get_vertex_attribute: obtains direct access to the 'a_pos' variable in the
 * vertex shader. Returns STATUS_OK on success, or STATUS_VER_ATTR on
 * error. This function also provides the vertices of the viewport to the vertex
 * shader. */

static enum init_status get_vertex_attribute ( GLuint program, GLint * addr )
{
        const char * var_name = "a_pos";

        /* get direct access to the 'a_pos' variable in the vertex shader */
        if ( ( *addr = glGetAttribLocation ( program, var_name ) ) == -1 )
                return STATUS_VER_ATTR;

        glVertexAttribPointer ( *addr, 2, GL_FLOAT, GL_FALSE, 0,
                        viewport_vertices );
        return STATUS_OK;
}

```

Source-Code Listing 17: Retrieving the address of the variable `a_pos` from the Vertex Shader on the GPU, and setting a clip-aware viewport matrix.

```
enum ppm_status {
        PPM_OK     =  0,
        PPM_FOPEN  = -1, /* fopen could not open the specified file */
        PPM_BADSIG = -2, /* the file signature was not as-expected */
        PPM_MALLOC = -3, /* could not allocate memory for the image */
        PPM_WIDTH  = -4, /* invalid width */
        PPM_HEIGHT = -5, /* invalid height */
        PPM_MAXVAL = -6, /* bad maximum value */
        PPM_EOF    = -7, /* unexpected E.O.F.\ when reading pixel data */
        PPM_INCHDR = -8  /* incomplete header */
};

/* [ exposed ] parse_ppm_error: returns a human-readable message from a
 * 'ppm_status' error code. */

const char * parse_ppm_error ( enum ppm_status status )
{
        switch ( status ) {
                case PPM_FOPEN:  return "fopen could not stat the file";
                case PPM_BADSIG: return "the file signature was not as-expected";
                case PPM_MALLOC: return "could not allocate memory for the " \
                                        "image";
                case PPM_WIDTH:  return "invalid width";
                case PPM_HEIGHT: return "invalid height";
                case PPM_MAXVAL: return "unsupported maximum value";
                case PPM_EOF:    return "unexpected end-of-file";
                case PPM_INCHDR: return "incomplete header";
                default:         return "unknown error";
        }
}

```

Source-Code Listing 18: The PPM status codes and the related parser function, `parse_ppm_error`

Listing 22, reads directly from the provided file and populates the appropriate members in the `ppm_data` structure.

This function follows the Common PPM Status Interface, reporting `PPM_INCHDR`, `PPM_WIDTH`, `PPM_HEIGHT`, and `PPM_MAXVAL` as appropriate.

6.4.5 Parsing Assistant and Loading: `parse_assist`

As the PPM Format Specification mandates the allowance of comments, being defined as a single line beginning with a hash '`#`' character. This function, listed in Source-Code Listing 23, reads

```
/* load_texture: uses the P.P.M. helpers to load a colour texture  */

static enum ppm_status load_texture ( const char * path )
{
        enum ppm_status status = PPM_OK;
        uint32_t * image = NULL;

        glBindTexture ( GL_TEXTURE_1D, 1 );
        glTexParameteri ( GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
        glTexParameteri ( GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
        glTexParameteri ( GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT );

        if ( ( status = load_image ( path, &image ) ) != PPM_OK )
                return status;

        glTexImage1D ( GL_TEXTURE_1D, 0, 4, 256, 0, GL_BGRA,
                        GL_UNSIGNED_BYTE, image );
        glEnable ( GL_TEXTURE_1D );

        free ( image );
        return PPM_OK;
}

```

Source-Code Listing 19: Loading the texture from a file, passing to the *OpenGL* subsystem for permanent use.

```
/* [ exposed ] load_image: loads the pixel data from P.P.M. file, located at
 * path, into the array *data. If this function fails at any point, there is no
 * need for the caller to free the array *data. */

enum ppm_status load_image ( const char * path, uint32_t ** data )
{
        enum ppm_status status = PPM_OK;
        struct ppm_data image = {
                .fp = NULL,
                .width = -1, .height = -1,
                .pixels = NULL
        };

        /* open the file and verify the signature */
        if ( ( image.fp = fopen ( path, "r" ) ) == NULL )
                return PPM_FOPEN;

        if ( verify_ppm_sig ( image.fp ) == -1 ) {
                fclose ( image.fp );
                return PPM_BADSIG;
        }

        /* read the header and pixel data */
        if ( ( status = read_hdr ( &image ) ) != PPM_OK ||
                        ( status = read_px_data ( &image ) ) != PPM_OK ) {
                fclose ( image.fp );
                return status;
        }

        *data = image.pixels;
        fclose ( image.fp );
        return PPM_OK;
}

```

Source-Code Listing 20: The wrapper function for parsing the PPM headers and pixel data-stream.

```
/* verify_ppm_sig: ensures that the file signature of 'fp' matches the expected
 * values. Returns -1 on failure, zero on success. THIS FUNCTION SETS THE FILE
 * POINTER BACK TO THE START [ +2 ]. */

static int verify_ppm_sig ( FILE * fp )
{
        /* http://netpbm.sourceforge.net/doc/ppm.html */
        const char sig [ ] = { 'P', '6' };

        fseek ( fp, 0, SEEK_SET );
        return ( fgetc ( fp ) != sig [ 0 ] ||
                        fgetc ( fp ) != sig [ 1 ] ) ? -1 : 0;
}

```

Source-Code Listing 21: A method of ensuring the file provided is intended to be of the PPM format. This is a near-instantaneous method of simple validation.

```c
#define BLOCK_SZ ( 64 )
#define COMMENT_CHAR ( '#' )
#define NEWLINE "\n\r"
#define MAX_VAL ( 255 )

struct ppm_data {
        FILE * fp;
        size_t width, height;
        uint32_t * pixels;
};

/* read_hdr: read the header of the P.P.M. file to determine the size (and
 * length of required pixel array). This function populates the width and height
 * fields of its argument. */

static enum ppm_status read_hdr ( struct ppm_data * image )
{
        char buffer [ BLOCK_SZ ];

        fseek ( image->fp, 0, SEEK_SET );
        parse_assist ( image, buffer );

        /* width */
        if ( !parse_assist ( image, buffer ) )
                return PPM_INCHDR;
        if ( !isdigit ( *buffer ) )
                return PPM_WIDTH;
        image->width = atoi ( buffer );

        /* height */
        if ( !parse_assist ( image, buffer ) )
                return PPM_INCHDR;
        if ( !isdigit ( *buffer ) )
                return PPM_HEIGHT;
        image->height = atoi ( buffer );

        /* ensure the maximum value is correct */
        if ( !parse_assist ( image, buffer ) )
                return PPM_INCHDR;
        if ( !isdigit ( *buffer ) || atoi ( buffer ) != MAX_VAL )
                return PPM_MAXVAL;

        return PPM_OK;
}

```

Source-Code Listing 22: The reading of the PPM header, populating the respective members of the appropriate structure with well-defined error-reporting.

characters directly from the file and skips comment lines while trimming extraneous white-space.

6.4.6 Reading Pixel Data: `UNPACK_DATA` and `read_px_data`

As with all data streams from potentially foreign lands, a mismatch in Endianness of the processor and that of the file is catastrophic, as each bit is read backwards. Whilst Big Endian is reasonably rare on modern processors, autonomous detection of Endianness has been included in *furakutaru*, and is used to define the way in which data is unpacked as it is read from the file.

Using this, the function `read_px_data`, as shown in Source-Code Listing 24, loads data from the pixel section (immediately succeeding the header), and places it in a buffer.

This function is compliant with the PPM Common Status Interface, and can return `PPM_MALLOC` or `PPM_EOF` to report a particular error.

6.5 Argument-Processing and User-Engagement

As outlined in the Analysis and Design sections, *furakutaru* aims to be a text-based System, vastly increasing its portability over a complex and unnecessary G.U.I. application. Because of this goal, the standard UNIX method of passing command-line arguments is used. The following section provides a commentary on the various methods and functions used to manage the command-line arguments, with the goal of creating a robust and efficient command-line application.

As it is also necessary that users are readily aware of the various arguments, this section details the ways in which *furakutaru* reports the argument documentation to the user.

```
/* parse_assist: populates the buffer whilst trimming white-space. White-space
 * is defined as a true result from ctype/isspace, or any single character from
 * NEWLINE. */

static int parse_assist ( struct ppm_data * image, char * buffer )
{
        int count = 0, ch = -1;

        while ( ( ch = fgetc ( image->fp ) ) != -1 &&
                        !isspace ( ch ) && count < BLOCK_SZ - 1 ) {
                if ( ch == COMMENT_CHAR ) {
                        /* encountered a comment;
                         * continue until something else is encountered */
                        while ( ( ch = fgetc ( image->fp ) ) != -1 &&
                                        ch != NEWLINE [ 0 ] &&
                                        ch != NEWLINE [ 1 ] );
                        ch = fgetc ( image->fp );
                        if ( ch == NEWLINE [ 0 ] || ch == NEWLINE [ 1 ] )
                                continue;
                }

                *buffer++ = ch;
                count++;
        }

        /* return to original position and null-terminate the string
         * in 'buffer' */

        /* trim additional white-space */
        while ( ( ch = fgetc ( image->fp ) ) != -1 && isspace ( ch ) );

        *buffer = '\0';
        ungetc ( ch, image->fp );

        return count;
}

```

Source-Code Listing 23: The parser helper-function used for skipping comments and other undesirable elements which should not reside in the buffer for final processing.

6.5.1 Common Enumerators and Helpers: Bit-by-Bit Management and Common Error Codes

As with the other substantial sections of the program, *furakutaru* defines a common interface with which to deal with errors, whilst also providing informative human-readable strings which can be output to the user. In addition to providing an informative error-interface, an enumerator type is established to track the specified options. This not only facilitates the robustness of the argument-processor, by disallowing multiple definitions of the same argument, but also mitigates the need for individual boolean flags for each flag, as they are managed on a bit-by-bit basis inside a sixteen-bit unsigned integer, defined as `uint16_t`.

Source-Code Listing 25 lists these enumerators and functions.

6.5.2 Common Argument Error Status Code Parser: `parse_args_error`

Source-Code Listing 26 lists the function which generates a human-readable string from the respective `arg_status` enumerator.

6.5.3 The Primary Argument-Processor: `process_args`

Akin to many previously discussed functions, the function `process_args`, listed in Source-Code Listing 27, performs the string-matching on the received argument list, searching for an argument's full name, or its shorter counterpart. Upon finding a result, the power-of-two nature of the argument position list, shown in Source-Code Listing 25, is used to efficiently calculate the argument position based on the index of the string to which it matched.

As this function is intended as a direct descendent of the entry-point, it produces its own error messages and merely reports a binary success/failure value to the caller, as an indication to continue or abort execution.

```
/* Determine the endian-ness for systems without LITTLE_ENDIAN or BIG_ENDIAN
 * defined by the standard library implementation. I got this great snippet from
 * somewhere years ago, not sure where, but it was Public Domain and a
 * reasonably standard check when dealing with files and ( especially ) network
 * sockets.
 *
 * When the endian-ness is determined, an appropriate bitwise packing method is
 * defined. */

#if !defined ( LITTLE_ENDIAN ) && !defined ( BIG_ENDIAN )
#if  defined(__i386__) || defined(__ia64__) || defined ( WIN32 ) || \
        ( defined ( __alpha__ ) || defined (__alpha ) ) || \
        defined ( __arm__ ) || \
        ( defined ( __mips__ ) && defined ( __MIPSEL__ ) ) || \
        defined ( __SYMBIAN32__ ) || \
        defined ( __x86_64__ ) || \
        defined ( __LITTLE_ENDIAN__ )
#define LITTLE_ENDIAN
#else
#define BIG_ENDIAN
#endif  /* general endian-ness check */
#endif  /* !defined ( LITTLE_ENDIAN ) && !defined ( BIG_ENDIAN ) */

#ifdef LITTLE_ENDIAN
#define UNPACK_DATA(r, g, b) ( ( ( r & 0xff ) << 16 ) | \
                ( ( g & 0xff ) << 8 ) | ( b & 0xff ) )
#else
#define UNPACK_DATA(r, g, b) ( ( ( b & 0xff ) << 16 ) | \
                ( ( g & 0xff ) << 8 ) | ( r & 0xff ) )
#endif

/* read_px_data: read the pixel data and place it into image->pixels. It is the
 * caller's responsibility to free the allocated memory space. */

static enum ppm_status read_px_data ( struct ppm_data * image )
{
        size_t px_count = image->width * image->height;
        int r = -1, g = -1, b = -1;

        if ( ( image->pixels = malloc ( px_count *
                                        sizeof ( uint32_t ) ) ) == NULL )
                return PPM_MALLOC;

        for ( size_t i = 0; i < px_count; i++ ) {
                if ( ( r = fgetc ( image->fp ) ) == -1 ||
                                ( g = fgetc ( image->fp ) ) == -1 ||
                                ( b = fgetc ( image->fp ) ) == -1 ) {
                        free ( image->pixels );
                        return PPM_EOF;
                }

                image->pixels [ i ] = UNPACK_DATA ( r, g, b );
        }

        return PPM_OK;
}

```

Source-Code Listing 24: Reading the pixel data using the appropriate Endianness

6.5.4 The Argument Sub-Processor: `arg_subprocessor`

In order to maintain the conciseness of the general string-matching argument-processor, the sub-processor exists to provide an intermediary between the low-level string-matching and the consideration of arguments on an individual basis. The `arg_subprocessor` function, listed in Source-Code Listing 28, has knowledge of each individual argument which *furakutaru* supports.

Using this, the sub-processor utilises function pointers to provide the correct auxiliary operand-processing function to the argument. For example, as the `ARG_ITERATION` and `ARG_DEGREE` arguments expect and integer, they are assigned the `set_int` function, which will reject floating-point or otherwise-invalid entries.

The `degree_sanitise` function ensures that the render degree is not set to an invalid value, which would cause the renderer to draw nothing. If this is the unfortunate case, the degree is set to the default.

The argument sub-processor uses the Common Argument Status Interface, outlined in Source-Code Listing 25, to report the accurate error to the primary processor.

```
#define SET_BIT(val, n) ( val |= n )
#define CHK_BIT(val, n) ( val &  n )

enum arg_positions {
        ARG_UNKNOWN   =     0,
        ARG_VERBOSITY =     1,
        ARG_LOCK      =     2,
        ARG_CENTRE_X  =     4,
        ARG_CENTRE_Y  =     8,
        ARG_SEED_X    =    16,
        ARG_SEED_Y    =    32,
        ARG_TYPE      =    64,
        ARG_ITERATION =   128,
        ARG_SCALE     =   256,
        ARG_DEGREE    =   512,
        ARG_COLOUR    =  1024,
        ARG_OUTDIR    =  2048,
        ARG_CONFIG    =  4096,
        ARG_ASSUMECFG =  8192,
        ARG_ROTATE    = 16384,
        ARG_SHOWHELP  = 32768
};

enum arg_status {
        ARG_S_HELP    =  3, /* the help should be displayed */
        ARG_S_OK_ACFG =  2, /* OK, --assumeconfig */
        ARG_S_OK      =  1, /* OK */
        ARG_S_OK_ADV  =  0, /* OK, advance the argument */
        ARG_S_ALONE   = -1, /* argument expected an operand */
        ARG_S_INVALID = -2, /* argument received an operand, but it was
                             * invalid */
        ARG_S_RANGE   = -3, /* operand was out-of-range */
        ARG_S_UNKNOWN = -4, /* argument string was not recognised */
        ARG_S_DDEFINE = -5, /* argument was doubly-defined */
        ARG_S_TOOLONG = -6, /* path operand exceeds PATH_MAX */
        ARG_S_ER_ACFG = -7  /* --assumeconfig without --config */
};

```

Source-Code Listing 25: The ways in which *furakutaru* manages arguments and the respective errors. In `arg_pos`, each value represents its position in a sixteen-bit unsigned binary integer, increasing in binary exponents.

```
/* parse_args_error: takes an arg_status value and returns a human-readable
 * string to be displayed to the user */

static const char * parse_args_error ( enum arg_status status )
{
        switch ( status ) {
                case ARG_S_ALONE:   return "Expected succeeding argument";
                case ARG_S_INVALID: return "Argument operand was invalid";
                case ARG_S_RANGE:   return "Argument operand was out-of-range";
                case ARG_S_UNKNOWN: return "Unrecognised argument";
                case ARG_S_DDEFINE: return "Argument was doubly-defined";
                case ARG_S_TOOLONG: return "Path exceeds maximum length";
                case ARG_S_ER_ACFG: return "--assumeconfig must succeed a " \
                                           "valid --config argument";
                default:            return "Unknown error";
        }
}

```

Source-Code Listing 26: The function for parsing a `arg_status` code to a string.

6.5.5 Shared Auxiliary Operand-Parsing Wrapper Functions

The following set of four functions are used throughout *furakutaru* to parse various data-types originating from an untrusted source. The functions, `set_float`, `set_int`, and `set_path` parse operands in the forms of floating-point numbers, integers, render degrees, and file-paths respectively. These functions are listed in Source-Code Listing 29.

The basic functions `set_int` and `set_float`, both return a success/failure to indicate their success. The `set_path` function is slightly more complex, however, as the function can fail at multiple points, for multiple reasons. Because of this, the latter returns an `arg_status` code.

```
/* [ exposed ] process_args: process the arguments in argv, and populated the
 * 'rs' structure appropriately. This function prints its own error messages.
 * Zero is returned on success, otherwise -1. If the latter is returned, the
 * caller should request an exit, and 'rs' is incomplete. If the function
 * succeeded but the user requested that the --config file be assumed as the
 * system defaults, 1 is returned. -2 is returned if the help information has
 * been requested; the program should usually quit with a success code if this
 * is the case. */

int process_args ( int argc, char ** argv, struct render_state * rs,
                char ** cfg )
{
        char * arg = NULL;
        enum arg_status status = ARG_S_OK;
        enum arg_positions apos = ARG_UNKNOWN;
        int assume_config = 0;

        static const char * arg_str_full [ ] = {
                "--verbose", "--lock", "--px", "--py", "--cx", "--cy", "--type",
                "--iteration", "--scale", "--degree", "--colour", "--outdir",
                "--config", "--assumeconfig", "--rotate", "--help"
        }, * arg_str_short [ ] = {
                "-v", "-l", NULL, NULL, NULL, NULL, "-t", "-i", "-s", "-d",
                "-c", "-o", "-f", NULL, "-r", "-h"
        };

        const int full_arg_count = sizeof ( arg_str_full ) /
                sizeof ( *arg_str_full );

        for ( int i = 1; i < argc; i++ ) {
                arg = argv [ i ];
                apos = ARG_UNKNOWN;

                /* if the corresponding arg_str_short value is NULL, there is no
                 * shortened counterpart against which to check */
                for ( int j = 0; j < full_arg_count; j++ ) {
                        if ( strcmp ( arg, arg_str_full [ j ] ) == 0 ||
                                        ( ( arg_str_short [ j ] ) ?
                                          ( strcmp ( arg,
                                                     arg_str_short [ j ] )
                                             == 0 ) : 0 ) ) {
                                apos = 1 << j;
                                break;
                        }
                }

                if ( apos == ARG_UNKNOWN ) {
                        fprintf ( stderr, "Argument error: \"%s\", %s\n",
                                        arg,
                                        parse_args_error ( ARG_S_UNKNOWN ) );
                        return -1;
                }

                switch ( ( status = arg_subprocessor (
                                        ( argc - i == 1 ) ? NULL :
                                            argv [ i + 1 ],
                                        apos, rs, cfg ) ) ) {
                        case ARG_S_OK:
                                continue;
                        case ARG_S_OK_ADV:
                                i++;
                                continue;
                        case ARG_S_OK_ACFG:
                                assume_config = 1;
                                continue;
                        case ARG_S_HELP:
                                return -2;
                        default:
                                fprintf ( stderr, "Argument error: \"%s\", " \
                                                "%s ( %d )\n", arg,
                                                parse_args_error ( status ),
                                                status );
                                return -1;
                        }
                }

        degree_sanitise ( & ( rs->degree ) );
        return assume_config;
```

Source-Code Listing 27: The topmost function used in argument-processing, tasked with performing the string-matching and calling of the sub-processor. This function produces its own error messages, and merely returns a binary indicator to its caller, informing the System whether execution should halt or continue. **This file is truncated due to space limitations. For the full listing, see the Appendices.**

```c
/* arg_subprocessor: sets/reports the appropriate values given in the argument
 * list and returns the status. ARG_S_OK_ADV indicates that the succeeding
 * argument has already been processed as an operand, and should be ignored by
 * the caller. */

static enum arg_status arg_subprocessor ( char * next, enum arg_positions arg,
                struct render_state * rs, char ** cfg )
{
        static uint16_t flags;
        void * val_ptr = NULL;
        int ( *setter ) ( void *, const char * ) = NULL;

        if ( CHK_BIT ( flags, arg ) )
                return ARG_S_DDEFINE;
        SET_BIT ( flags, arg );

        switch ( arg ) {
                case ARG_SHOWHELP:
                        print_help ( );
                        return ARG_S_HELP;
                case ARG_VERBOSITY:
                        rs->verbosity = 1;
                        break;
                case ARG_LOCK:
                        rs->locking = 1;
                        break;
                case ARG_TYPE:
                        if ( !next )
                                return ARG_S_ALONE;

                        if ( strcmp ( next, "mandelbrot" ) == 0 ) {
                                rs->type = FRAC_MANDELBROT;
                        } else if ( strcmp ( next, "julia" ) == 0 ) {
                                rs->type = FRAC_JULIA;
                        } else {
                                return ARG_S_INVALID;
                        }

                        return ARG_S_OK_ADV;
                case ARG_SCALE:
                        val_ptr = & ( rs->scale );
                        setter = set_float;
                        break;
                case ARG_CENTRE_X:
                        val_ptr = & ( rs->centre_x );
                        setter = set_float;
                        break;
                case ARG_CENTRE_Y:
                        val_ptr = & ( rs->centre_y );
                        setter = set_float;
                        break;
                case ARG_SEED_X:
                        val_ptr = & ( rs->seed_x );
                        setter = set_float;
                        break;
                case ARG_SEED_Y:
                        val_ptr = & ( rs->seed_y );
                        setter = set_float;
                        break;
                case ARG_ITERATION:
                        val_ptr = & ( rs->max_iterations );
                        setter = set_int;
                        break;
                case ARG_DEGREE:
                        val_ptr = & ( rs->degree );
                        setter = set_int;
                        break;
                case ARG_ROTATE:
                        val_ptr = & ( rs->angle );
                        setter = set_int;
                        break;
                case ARG_COLOUR:
                        return set_path ( & ( rs->colour_path ), next );
                case ARG_OUTDIR:
                        return set_path ( & ( rs->out_dir ), next );
                case ARG_CONFIG:
                        return set_path ( cfg, next );
                case ARG_ASSUMECFG:
```

Source-Code Listing 28: The sub-processor, used for dealing with arguments on an individual basis and calling the appropriate subsequent functions for operand-verification. This function performs no string-matching, aside from the sole exception of the processing of the `ARG_TYPE` argument. **This file is truncated due to space limitations. For the full listing, see the Appendices.**

```c
/* [ exposed ] set_float: parses a GLfloat and stores its value in the value
 * referred to by 'val'. Zero returned on success, -1 if 'str' is NULL, and '-2'
 * if the parsing failed. */

int set_float ( void * val, const char * str )
{
        char * end_ptr = ( char * ) str;
        GLfloat tmp;

        errno = 0;

        if ( !str )
                return -1;

        tmp = strtof ( str, &end_ptr );

        if ( errno || *end_ptr != '\0' )
                return -2;

        * ( ( GLfloat * ) val ) = tmp;
        return 0;
}

/* [ exposed ] set_int: see comments for set_float */

int set_int ( void * val, const char * str )
{
        char * end_ptr = ( char * ) str;
        GLint tmp;

        errno = 0;

        if ( !str )
                return -1;

        tmp = strtol ( str, &end_ptr, 10 );

        if ( errno || *end_ptr != '\0' )
                return -2;

        * ( ( GLint * ) val ) = tmp;
        return 0;
}

/* set_path: sets the next string to the location pointed to by 'val'.
 * ARG_S_ALONE is returned is there is no next argument, or ARG_S_OK_ADV on
 * success. ARG_S_TOOLONG is returned if the 'str' is too long for a UNIX
 * path. */

static enum arg_status set_path ( char ** val, char * str )
{
        if ( !str )
                return ARG_S_ALONE;

        if ( strlen ( str ) > PATH_MAX )
                return ARG_S_TOOLONG;

        *val = str;
        return ARG_S_OK_ADV;
}


/* [ exposed ] degree_sanitise: ensure that the degree is in a correct range,
 * otherwise set it to the default */

void degree_sanitise ( GLint * val )
{
        switch ( *val ) {
                case -1:
                case  0:
                case  1:
                        *val = DEGREE_DEFAULT;
        }
}

```

Source-Code Listing 29: Various helper functions for the parsing of untrusted data.

6.5.6 User-Engagement: Greeting

If verbosity is requested, then a loving, time-appropriate message is printed to the standard output device. This function uses the system time to select a time-appropriate, English greeting from a list of contenders. The function, `print_welcome`, is listed in Source-Code Listing 30.

```
#ifdef UNICODE_SUPPORTED
#define PROGRAM_TITLE        "ãM–^CM–^UãM–^C©ãM–^B˜ãM–^B¿ãM–^C«"
#else
#define PROGRAM_TITLE        "FURAKUTARU"
#endif /* UNICODE_SUPPORTED */

#define PROGRAM_VERSION      "1.0"
#define PROGRAM_AUTHOR       "Oliver Dixon"
#define PROGRAM_LICENCE      "M.I.T. Licence"
#define PROGRAM_LICENCE_URL  "https://mit–license.org/"

/* [ exposed ] print_welcome: makes the user feel special
 * https://www.youtube.com/watch?v=3ymwOvzhwHs */

void print_welcome ( )
{
        time_t raw_time = -1;
        struct tm * time_inf = NULL;

        int idx = 0;
        char * greeting [ ] = { "Hello", "Good morning", "Good afternoon",
                "Good evening" };

        if ( time ( &raw_time ) != ( time_t ) -1 &&
                        ( time_inf = localtime ( &raw_time ) ) != NULL ) {
                if ( time_inf->tm_hour < 12 ) {
                        idx = 1;
                } else if ( time_inf->tm_hour < 18 ) {
                        idx = 2;
                } else {
                        idx = 3;
                }
        }

        printf ( "%s. This is %s version %s by %s.\n" \
                "This software is licensed under the %s, a copy of\nwhich " \
                "can be found at %s or in the\nprogram directory.\n\n",
                greeting [ idx ], PROGRAM_TITLE, PROGRAM_VERSION,
                PROGRAM_AUTHOR, PROGRAM_LICENCE, PROGRAM_LICENCE_URL );
}

```

Source-Code Listing 30: Welcomes the user to the *furakutaru* System

6.5.7 User-Engagement: Argument Listing

As indicated in the previous sections, it is integral that, due to *furakutaru*'s heavy reliance on command-line arguments, that the user is well-versed in the options that the System accepts. By passing the `-h` option to the program, it does nothing aside from print a list of each argument, its operand (if applicable), and a brief description. The output is aligned to sixteen-column sectors, allowing for a well-formatted and easy-to-read output; this is shown in the Testing section.

This allows the Clientele to use the Product effectively without having to search the source code, as is the case with many ill-developed applications. The help function, `print_help`, can be found in Source-Code Listing 31.

6.6 The General *furakutaru* Runtime

The majority of the *furakutaru* code-base, including the operating system entry-point, is contained within the Runtime. The following section will detail each Runtime function. Note that this section does not define its own Common Status Interface, as the chances of erroneous behaviour occurring once instantiation having completed is marginally low.

6.6.1 Default System Values: `set_system_defaults`

furakutaru does not require that a user specify any options, and thus has a responsibility to provide sensible defaults to the renderer, assuming the likely case that some, or all, options have not been explicitly specified by the user. These defaults, in addition to the helper function which copies them to the initial render state, are listed in Source-Code Listing 32.

These defaults can be overwritten at any time, as `set_system_defaults` is only called once, before any other function, such as the argument-processor or file-parser, has a chance to invalidate the defaults.

```
/* [ exposed ] print_help: prints the command-line arguments and a description
 * of each to the stdout device. */

void print_help ( )
{
        time_t raw_time = -1;
        struct tm * time_inf = NULL;

        if ( time ( &raw_time ) != ( time_t ) -1 &&
                        ( time_inf = localtime ( &raw_time ) ) != NULL &&
                        ( time_inf->tm_hour == 0 && time_inf->tm_min == 30 ) )
                puts ( "\tgimme gimme gimme! ( a manual page after " \
                                "midnight )\n" );

        /* oh my goodness
         * https://www.youtube.com/watch?v=Qc23ryXcmME */
        printf ( PROGRAM_TITLE " v." PROGRAM_VERSION ", by " PROGRAM_AUTHOR \
                 "\nCommand-Line Arguments Summary\n\n" \
                 "%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n" \
                 "%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n" \
                 "%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n" \
                 "%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n%-16s %-16s\n",
                        "--verbose, -v", "Be overly-verbose.",
                        "--lock, -l", "Ignore interactive input, aside from " \
                                "the 'quit' and 'report' commands.",
                        "--px", "Specify the Real co-ordinate of the centre.",
                        "--py", "Specify the Imaginary co-ordinate of the " \
                                "centre.",
                        "--cx", "[Julia] Specify the Real co-ordinate of the" \
                                " seed.",
                        "--cy", "[Julia] Specify the Imaginary co-ordinate" \
                                " of the seed.",
                        "--type, -t", "Specify the type of Set: 'mandelbrot'" \
                                " or 'julia' are permitted.",
                        "--iteration, -i", "Specify the maximum iterations.",
                        "--scale, -s", "Specify the scale to which the Set " \
                                "should be zoomed.",
                        "--degree, -d", "Specify the degree to which the " \
                                "generating-polynomial should be " \
                                "raised.",
                        "--colour, -c", "Loads the file containing the colour" \
                                " palette.",
                        "--outdir, -o", "The directory to place output files.",
                        "--config, -f", "A file from which to load the " \
                                "initial render state.",
                        "--assumeconfig", "Assume the file specified by '-f'" \
                                " as the default state.",
                        "--rotate, -r", "Specify the angle by which to rotate" \
                                " the initial render.",
                        "--help, -h", "Display this help message and quit.");
}

```

Source-Code Listing 31: Provides a simple, comprehensive, and authoritative source of information to the user.

6.6.2 Respecting the Input Configuration File: `setup_init_vals`

This is largely a wrapper function to avoid the main entry-point from becoming convoluted and unmanageable on an twenty-four-line terminal. Upon the successful completion of the argument-processor, should it occur, the `setup_init_vals` function is called, to determine the precedence of defaults. If the `--assumeconfig` option was passed on the command-line, this function causes the default render state to become the configuration specified in the file. The function is specified in Source-Code Listing 33.

In practice, this implies that the user's request to reset the renderer to its default state, should revert not to the system defaults described in Source-Code Listing 32, but those interpreted from the configuration file.

6.6.3 The OS Entry-Point: `int main ( int argc, char ** argv )`

The entry function, listed in Source-Code Listing 34, is the parent of all other functions discussed in this document. Returning the UNIX standard `EXIT_SUCCESS` or `EXIT_FAILURE` on success or failure respectively, this function takes the number of command line arguments, plus one[7], for its first argument, and the argument array as its second.

This function also maintains the majority of original copies of the variables used throughout the

[7] On the majority of systems, the program-invocation path is considered as `argv [ 0 ]`.

```
#define MAX_ITERATION_DEFAULT ( 1000 )
#define SCALE_DEFAULT         ( 2.2  )
#define SEED_X_DEFAULT        ( 0.0  )
#define SEED_Y_DEFAULT        ( 0.0  )
#define DEGREE_DEFAULT        ( 2    )
#define CENTRE_X_DEFAULT      ( 0.0  )
#define CENTRE_Y_DEFAULT      ( 0.0  )
#define VERBOSITY_DEFAULT     ( 0    )
#define LOCKING_DEFAULT       ( 0    )
#define ROTATION_DEFAULT      ( 0    )

#ifndef AUX_DIR
#define AUX_DIR "src/aux/"
#endif

/* [ exposed ] set_system_defaults */

void set_system_defaults ( struct render_state * rs )
{
        rs->max_iterations = MAX_ITERATION_DEFAULT;
        rs->scale = SCALE_DEFAULT;
        rs->seed_x = SEED_X_DEFAULT;
        rs->seed_y = SEED_Y_DEFAULT;
        rs->degree = DEGREE_DEFAULT;
        rs->centre_x = CENTRE_X_DEFAULT;
        rs->centre_y = CENTRE_Y_DEFAULT;
        rs->type = FRAC_MANDELBROT;
        rs->verbosity = VERBOSITY_DEFAULT;
        rs->locking = LOCKING_DEFAULT;
        rs->angle = ROTATION_DEFAULT;
        rs->colour_path = NULL;
        rs->out_dir = NULL;
}


```

Source-Code Listing 32: Defines, and sets, appropriate default values. Assuming these values, the renderer draws a typical Mandelbrot Set of the second degree.

```
/* setup_init_vals: initiates the appropriate initial and current render
 * structures. Also reports the status/failure of importing a file. On failure,
 * this function returns -1, and zero on success. */

static int setup_init_vals ( struct render_state * rs,
                struct render_state * is, char * cfg, int assume_cfg )
{
        enum export_status status = EXPORT_S_IMPORT_OK;

        /* If --assumeconfig was set, use the config file as the default render
         * state. Otherwise, only use it as the initial state, and allow the
         * initial state to remain as the command-line arguments mixed with
         * the system defaults. */

        if ( cfg ) {
                fputs ( "Importing ", stdout );
                puts ( cfg );

                if ( ( status = csv_import ( cfg, rs ) )
                                != EXPORT_S_IMPORT_OK ) {
                        fputs ( parse_export_status ( status ), stderr );
                        fputc ( '\n', stderr );
                        return -1;
                }

                puts ( parse_export_status ( EXPORT_S_IMPORT_OK ) );
                if ( assume_cfg )
                        memcpy ( is, rs, sizeof ( struct render_state ) );
        }

        return 0;
}

```

Source-Code Listing 33: Assume the file configuration to be the true default, overriding the system defaults.

entire code-base, such as the current renderer state, `rs`, and the initial/default renderer state, `is`. Information regarding the *X* connection is also maintained in `x_disp_inf disp_inf` and `x_fb_inf fb_inf`.

Upon attaining all the argument values and initialising *OpenGL*, the entry-point function sets the default program and instantiates an identity matrix to be used in the Vertex Shader.

```
/* [ entry ] main: main entry point, taking arguments. This function returns
 * EXIT_SUCCESS on success, or EXIT_FAILURE on failure. */

int main ( int argc, char ** argv )
{
        struct x_disp_inf disp_inf;
        struct x_fb_inf fb_inf;
        GLuint program_list [ FRACTAL_COUNT ] = { 0 };
        struct render_state rs, is;
        int assume_cfg = 0;
        char * cfg = NULL;

        set_system_defaults ( &is );

        if ( ( assume_cfg = process_args ( argc, argv, &is, &cfg ) ) == -1 )
                return EXIT_FAILURE;
        if ( assume_cfg == -2 ) /* help requested; nothing to do */
                return EXIT_SUCCESS;

        rs = is;
        if ( rs.verbosity )
                print_welcome ( );

        if ( setup_init_vals ( &rs, &is, cfg, assume_cfg ) == -1 ||
                        init_assistant ( &disp_inf, &fb_inf, program_list,
                                is.colour_path ) == -1 )
                return EXIT_FAILURE;

        rs.program = program_list [ rs.type ];
        glUseProgram ( rs.program );

        set_rotation_matrix ( &rs, rs.angle );
        reset_render_state ( &rs, ( cfg ) ? &rs : &is );
        glEnableVertexAttribArray ( fb_inf.a_pos_addr );

        while ( 1 ) {
                switch ( ev_wait ( &disp_inf, &rs, program_list ) ) {
                        case -1:
                                /* -1 = ( gracious ) quit */
                                clean_all ( &disp_inf, &fb_inf, program_list );
                                glDisableVertexAttribArray (
                                                fb_inf.a_pos_addr );
                                if ( rs.verbosity )
                                        puts ( "\nThank you for using " \
                                                        PROGRAM_TITLE );
                                return EXIT_SUCCESS;
                        case 1:
                                /* 1 = reset state and update */
                                reset_render_state ( &rs, &is );
                                /* fall through */
                        case 0:
                                update ( &disp_inf, &fb_inf );
                }
        }
}

```

Source-Code Listing 34: The main entry-point for the *furakutaru* System, returning and assuming the expected values for a standard *UNIX*-like system.

6.6.4 Controlling the Rotation: `set_rotation_matrix`

As specified in the Analysis and Design sections, it is integral that *furakutaru* contains the functionality to rotate the render. This is achieved using a Vertex Shader, which multiplies the viewport vertices by a rotation matrix before passing control to the Fragment Shader. This way, the Fragment Shaders do not have to have any acknowledgement or knowledge of the state, or even ability, of rotation.

The `set_rotation_matrix` function is shown in Source-Code Listing 35.

Once the appropriate matrix has been created, this function also makes a copy of the local matrix in the render state structure, `rs`. This is to appease the *OpenGL* subsystem, as it occasionally mandates unpredictable access to original copies of the data stored on the GPU This function uses the *OpenGL*-provided `GLUniformMatrix4fv` to pass the four-by-four matrix directly to the Vertex Shader.

6.6.5 The Blocking Event Loop: `ev_wait`

The event loop calls the *XCB* built-in `xcb_wait_for_event` function, which blocks until the window receives an event to which it is subscribed—this is explained fully in § 6.2.8. In the case of

```
#define DEGRAD_MULTIPLIER ( 0.01745329252 ) /* ~pi/180 */

/* [ exposed ] set_rotation_matrix: sets the angle, given in degrees, by which
 * to rotate the render. The matrix is created, locally and copied to the
 * rs->rot_matrix and the vertex shader, as it is a requirement that the
 * original matrix values are accessible throughout execution of a scene. */

void set_rotation_matrix ( struct render_state * rs, GLint new_angle )
{
        GLint addr = glGetUniformLocation ( rs->program, "rot_matrix" );
        GLfloat rad = DEGRAD_MULTIPLIER;

        if ( addr == -1 )
                return;

        new_angle %= 360; /* wrap on a full rotation */
        rad *= new_angle;

        GLfloat matrix [ 16 ] = {
                cos ( rad ), - sin ( rad ), 0.0, 0.0,
                sin ( rad ),   cos ( rad ), 0.0, 0.0,
                0.0, 0.0, 1.0, 0.0,
                0.0, 0.0, 0.0, 1.0
        };

        rs->angle = new_angle;
        memcpy ( rs->rot_matrix, matrix, sizeof ( GLfloat [ 16 ] ) );
        glUniformMatrix4fv ( addr, 1, GL_FALSE, matrix );
}

```

Source-Code Listing 35: Sets the rotation matrix based on an angle provided, in degrees, to be passed to the Vertex Shader.

furakutaru, a window-invalidation, such as a resize of move, or a key-press whilst focus is placed on the window, both cause this function to return the nature of the event in the `xcb_generic_event_t` structure.

This structure is then cast to the appropriate event structure, and its data is read. While some specialised key-codes are handled in this function, the majority of key-presses are handled in a separate handler, akin to the `arg_subprocessor` function listed in Source-Code Listing 28.

The function, listed in Source-Code Listing 36, returns zero as a signal to update the rendering area, -1 as a signal to gracefully quit, or +1 as a signal to reset the renderer to its initial state and update the screen.

6.6.6 Reset the Renderer to its Initial State: `reset_render_state`

This function, listed in Source-Code Listing 37, resets the render state back to its initial state. It is assumed that the first argument points to the currently active render, and the second argument points to the initial state. `set_rotation_matrix`, detailed in Source-Code Listing 35, is called as a consequence of this function, which does a deep copy of the rotation matrix from the initial state to the current.

6.6.7 Uniform-Setter Wrapper Functions: `1i`, `1f`, and `2f`

furakutaru defines functions, which it suggests the compiler in-lines, to provide a robust method of accessing `uniform` variables in the Vertex and Fragment Shaders. As the GLSL source code defines uniforms by name, it is necessary to perform a look-up when attempting to write data to their location on the GPU The four functions, used to set a single integer, single float, or point, are named `set_uniform1i`, `set_uniform1f`, and `set_uniform2f` respectively.

The function set is listed in Source-Code Listing 38.

6.6.8 The Generic Key-Code Handler: `keycode_handle_gen`

As shown in Source-Code Listing 36, an additional function is called to handle the key-codes on a key-by-key basis, setting appropriate auxiliary function handlers for each case. This function, `keycode_handle_gen`, is outlined in Source-Code Listing 39.

This function is designed to work with the `verbose_report` function for high-quality and precise reporting for the clients enabling verbosity.

```
/* ev_wait: *blocking* event-checking loop. Waits for events, deals with
 * them accordingly, and returns 0 on success, or -1 as a signal to exit. A
 * return value of 1 indicates that the renderer should be reset. */

static int ev_wait ( struct x_disp_inf * disp_inf, struct render_state * rs,
                GLuint program_list [ FRACTAL_COUNT ] )
{
        xcb_generic_event_t * ev = NULL;
        xcb_keycode_t key;
        int ret = 0;

        ev = xcb_wait_for_event ( disp_inf->xcb_conn );
        switch ( ev->response_type & SYN_MASK ) {
                /* XCB_EXPOSE: update the window width and height values
                 * when it is resized */
                case XCB_EXPOSE:
                        disp_inf->win_width = (( xcb_expose_event_t * )ev)
                                ->width;
                        disp_inf->win_height = (( xcb_expose_event_t * )ev)
                                ->height;
                        break;

                /* XCB_KEY_PRESS: process the key-code sent */
                case XCB_KEY_PRESS:
                        key = ( ( xcb_key_press_event_t * )ev )->detail;

                        switch ( key ) {
                                case KEYCODE_QUIT:
                                        ret = -1;
                                        break;
                                case KEYCODE_INFO:
                                        print_info ( rs );
                                        break;
                        }

                        if ( rs->locking )
                                break;

                        switch ( key ) {
                                case KEYCODE_DEFAULT:
                                        if ( rs->verbosity )
                                                puts ( "Reverted to " \
                                                        "defaults." );
                                        ret = 1;
                                        break;
                                case KEYCODE_SWITCH:
                                        rs->type = !rs->type;
                                        rs->program = program_list
                                                [ rs->type ];
                                        glUseProgram ( rs->program );
                                        if ( rs->verbosity ) {
                                                stdputs ( "Switched the " \
                                                        "fractal " \
                                                        "to a " );
                                                stdputs ( ( rs->type ==
                                                        FRAC_MANDELBROT )
                                                        ? "Mandelbrot"
                                                        : "Julia" );
                                                puts ( " Set." );
                                        }
                                        set_rotation_matrix ( rs, 0 );
                                        ret = 1;
                                        break;
                                case KEYCODE_SAVEIMG:
                                        puts ( parse_export_status (
                                               image_export (
                                                        disp_inf->win_width,
                                                        disp_inf->win_height,
                                                        rs->out_dir ) ) );
                                        break;
                                default:
                                        keycode_handle_gen ( key, rs );
                        }
        }

        free ( ev );
        return ret;
}
```

Source-Code Listing 36: The blocking event loop, used to await key-presses or window-invalidation as an indicator for the renderer being required to re-draw the screen with the new values. If this function cannot handle the incoming key-code itself, it is passed to the general `keycode_handle_gen` function, which may further pass the code to additional key-handling functions.

```
/* reset_render_state: sets the initial values when conferring with the
 * OpenGL shaders program */

static void reset_render_state ( struct render_state * rs,
                 struct render_state * is )
{
        if ( rs != is ) {
                rs->max_iterations = is->max_iterations;
                rs->centre_x = is->centre_x;
                rs->centre_y = is->centre_y;
                rs->seed_x = is->seed_x;
                rs->seed_y = is->seed_y;
                rs->scale = is->scale;
                rs->degree = is->degree;
                rs->angle = is->angle;
        }

        set_uniform1i ( rs->program, "max_iterations", rs->max_iterations );
        set_uniform1f ( rs->program, "scale", rs->scale );
        set_uniform2f ( rs->program, "centre", rs->centre_x, rs->centre_y );
        set_uniform1i ( rs->program, "degree", rs->degree );
        set_uniform2f ( rs->program, "seed", rs->seed_x, rs->seed_y );
        set_rotation_matrix ( rs, is->angle );
}

```

Source-Code Listing 37: Resetting a render state back to defaults, whether the defaults originate from the System Stock, or a custom configuration file.

```
/* set_uniform* wrapper functions: these functions act as a less horrendous
 * interface to the glUniform* functions. If they fail, it is silent, as such a
 * failure would not be fatal but would be very visually-apparent. */

static inline void set_uniform1f ( GLuint program, const GLchar * property,
                GLfloat val )
{
        GLint addr = glGetUniformLocation ( program, property );
        if ( addr != -1 )
                glUniform1f ( addr, val );
}

static inline void set_uniform1i ( GLuint program, const GLchar * property,
                GLfloat val )
{
        GLint addr = glGetUniformLocation ( program, property );
        if ( addr != -1 )
                glUniform1i ( addr, val );
}

static inline void set_uniform2f ( GLuint program, const GLchar * property,
                GLfloat val_1, GLfloat val_2 )
{
        GLint addr = glGetUniformLocation ( program, property );
        if ( addr != -1 )
                glUniform2f ( addr, val_1, val_2 );
}

```

Source-Code Listing 38: The `set_uniform` wrappers, used as a safe and robust interface to the GPU uniform variables.

6.6.9 Julia Set-Specific Key-Handling: `keycode_handle_julia`

To ensure conciseness of the general key-code handler, the `keycode_handle_julia` function, listed in Source-Code Listing 40, is called when the general handler does not recognise the action being passed; as this is not referring to user data, it can be assumed it is valid to some extent.

Though small, the function operates identically to the general key-code handler, sending its output to the verbose reporter if necessary.

6.6.10 Ensuring the Health of the Render State: `degree_change`

As previously mentioned in Source-Code Listing 29, the degree must not become a value which the renderer cannot handle; namely, this is -1, 0, or +1. Because of this, the `degree_change` function, listed in Source-Code Listing 41, manages direct access to the `degree` variable in the GLSL Shaders.

```
/* keycode_handle_gen: confers with the OpenGL program to handle certain
 * key-presses. Application controls, such as quitting or displaying
 * information, should already have been handled. */

static void keycode_handle_gen ( xcb_keycode_t key, struct render_state * rs )
{
        union verbose_data delta, new_val;

        switch ( key ) {
                case KEYCODE_INCITER:
                        delta.i = ITERATION_STEP;
                        rs->max_iterations += delta.i;
                        set_uniform1i ( rs->program, "max_iterations",
                                        rs->max_iterations );
                        new_val.i = rs->max_iterations;
                        break;
                case KEYCODE_DECITER:
                        delta.i = -ITERATION_STEP;
                        rs->max_iterations += delta.i;
                        set_uniform1i ( rs->program, "max_iterations",
                                        rs->max_iterations );
                        new_val.i = rs->max_iterations;
                        break;
                case KEYCODE_ZOOMIN:
                        delta.f = SCALE_STEP;
                        rs->scale *= delta.f;
                        set_uniform1f ( rs->program, "scale", rs->scale );
                        new_val.f = rs->scale;
                        break;
                case KEYCODE_ZOOMOUT:
                        delta.f = 1/SCALE_STEP;
                        rs->scale *= delta.f;
                        set_uniform1f ( rs->program, "scale", rs->scale );
                        new_val.f = rs->scale;
                        break;
                case KEYCODE_LEFT:
                        delta.f = rs->scale * MOVEMENT_STEP;
                        rs->centre_x += cos ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        rs->centre_y += sin ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        set_uniform2f ( rs->program, "centre", rs->centre_x,
                                        rs->centre_y );
                        new_val.f = rs->centre_x;
                        break;
                case KEYCODE_RIGHT:
                        delta.f = - ( rs->scale * MOVEMENT_STEP );
                        rs->centre_x += cos ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        rs->centre_y += sin ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        set_uniform2f ( rs->program, "centre", rs->centre_x,
                                        rs->centre_y );
                        new_val.f = rs->centre_x;
                        break;
                case KEYCODE_UP:
                        delta.f = - ( rs->scale * MOVEMENT_STEP );
                        rs->centre_x -= sin ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        rs->centre_y += cos ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        set_uniform2f ( rs->program, "centre", rs->centre_x,
                                        rs->centre_y );
                        new_val.f = rs->centre_y;
                        break;
                case KEYCODE_DOWN:
                        delta.f = rs->scale * MOVEMENT_STEP;
                        rs->centre_x -= sin ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        rs->centre_y += cos ( rs->angle * DEGRAD_MULTIPLIER ) *
                                delta.f;
                        set_uniform2f ( rs->program, "centre", rs->centre_x,
                                        rs->centre_y );
                        new_val.f = rs->centre_y;
                        break;
                case KEYCODE_INCDEG:
                        delta.i = degree_change ( & ( rs->degree ), 1 );
                        set_uniform1i ( rs->program, "degree", rs->degree );
```

Source-Code Listing 39: Handling key-codes on an individual basis, setting appropriate auxiliary functions to parse and report the interaction. Many of the modifications are made with respect to the current zoom level: this is to prevent the renderer making huge jumps when the user attempts to pan into a highly zoomed fractal. **This file is truncated due to space limitations. For the full listing, see the Appendices.**

```c
/* keycode_handle_julia: handles keycodes specific to the management of the
 * Julia Set. */

static void keycode_handle_julia ( struct render_state * rs, xcb_keycode_t key )
{
        union verbose_data delta, new_val;
        delta.f = 0.0; /* G.C.C., oh my goodness */

        switch ( key ) {
                case KEYCODE_SEED_L:
                        delta.f = rs->scale * MOVEMENT_STEP;
                        rs->seed_x += delta.f;
                        new_val.f = rs->seed_x;
                        break;
                case KEYCODE_SEED_R:
                        delta.f = - ( rs->scale * MOVEMENT_STEP );
                        rs->seed_x += delta.f;
                        new_val.f = rs->seed_x;
                        break;
                case KEYCODE_SEED_D:
                        delta.f = rs->scale * MOVEMENT_STEP;
                        rs->seed_y += delta.f;
                        new_val.f = rs->seed_y;
                        break;
                case KEYCODE_SEED_U:
                        delta.f = - ( rs->scale * MOVEMENT_STEP );
                        rs->seed_y += delta.f;
                        new_val.f = rs->seed_y;
                        break;
                case KEYCODE_DEFAULT:
                        rs->seed_x = SEED_X_DEFAULT;
                        rs->seed_y = SEED_Y_DEFAULT;
                        break;
        }

        set_uniform2f ( rs->program, "seed", rs->seed_x, rs->seed_y );
        if ( rs->verbosity )
                verbose_report ( key, delta, new_val );
}

```

Source-Code Listing 40: A Julia-specific key-code handler function, acting as a complement to `keycode_handle_gen`, detailed in Source-Code Listing 39.

```c
/* degree_change: ensures the degree change does not go to a disallowed value
 * which causes peculiar ( or a complete absence of ) rendering: -1, 0, 1. */

static GLint degree_change ( GLint * dest, int direction )
{
        GLint new_val = *dest + ( direction * DEGREE_STEP );

        if ( direction != -1 && direction != 1 ) {
                *dest = DEGREE_DEFAULT;
                return DEGREE_DEFAULT;
        }

        switch ( new_val ) {
                case -1:
                case  0:
                case  1:
                        new_val = direction * DEGREE_DEFAULT;
                        break;
        }

        *dest = new_val;
        return new_val;
}

```

Source-Code Listing 41: Manages the direct editing of the `degree` variable, ensuring it does not enter a state which disables the renderer.

6.6.11 Extra-Verbose Reporting: `verbose_report`

If the `-v` or `--verbose` options were passed as command-line arguments, every interactive action is logged to the standard output device. A union is utilised in order to pass the various data-types to a common reporting function: `verbose_report`, listed in Source-Code Listing 42. This allows the source code to remain concise while retaining its accuracy and clientele-appeasement factor.

```
union verbose_data {
        GLint i;
        GLfloat f;
};

/* verbose_report: in the event of requested verbosity, print every action to
 * stdout */

static void verbose_report ( enum keycodes action, union verbose_data delta,
                union verbose_data new_data )
{
        int is_int = 0;

        switch ( action ) {
                case KEYCODE_LEFT:     stdputs ( "Pan left of" ); break;
                case KEYCODE_DOWN:     stdputs ( "Pan downwards of" ); break;
                case KEYCODE_UP:       stdputs ( "Pan upwards of" ); break;
                case KEYCODE_RIGHT:    stdputs ( "Pan right of" ); break;
                case KEYCODE_ZOOMIN:   stdputs ( "Scale-up by x" ); break;
                case KEYCODE_ZOOMOUT:  stdputs ( "Scale-down by x" ); break;
                case KEYCODE_INCITER:  stdputs ( "Increased iterations by" );
                                       is_int = 1;
                                       break;
                case KEYCODE_DECITER:  stdputs ( "Decreased iterations by" );
                                       is_int = 1;
                                       break;
                case KEYCODE_INCDEG:   stdputs ( "Increased the degree by" );
                                       is_int = 1;
                                       break;
                case KEYCODE_DECDEG:   stdputs ( "Decreased the degree by" );
                                       is_int = 1;
                                       break;
                case KEYCODE_SEED_L:   stdputs ( "Moved the seed left by" );
                                       break;
                case KEYCODE_SEED_D:   stdputs ( "Moved the seed downwards " \
                                                 "by" );
                                       break;
                case KEYCODE_SEED_U:   stdputs ( "Moved the seed upwards by" );
                                       break;
                case KEYCODE_SEED_R:   stdputs ( "Moved the seed right by" );
                                       break;
                case KEYCODE_SWITCH:
                                       return;
                case KEYCODE_ROTLEFT:
                case KEYCODE_ROTRGHT:
                                       stdputs ( "Rotated by" );
                                       is_int = 1;
                                       break;
                default:
                        return;
        }

        if ( is_int ) {
                printf ( " %d. The new value is %d.\n", ( GLint ) delta.i,
                                ( GLint ) new_data.i );
        } else {
                printf ( " %f. The new value is %f.\n", ( GLfloat ) delta.f,
                                ( GLfloat ) new_data.f );
        }
}

```

Source-Code Listing 42: A union is being used to transport the data from the key-code handler: non-integer and non-float values are expected to self-report.

6.6.12 Printing Information On-Request: `print_info`

The `print_info` function, listed in Source-Code Listing 43, outputs all the information regarding the current render to the standard output device in a human-readable and formatted form. If the requested information regards a Julia Set, the seed co-ordinates are included in the report.

6.6.13 The Render Update: `update`

`update` confers with *OpenGL*, informing the subsystem that a new frame needs to be rendered. By this point, it is expected that all new values should have been placed onto the GPU using a `set_uniform` wrapper, or similar. This function is listed in Source-Code Listing 44.

The viewpoint is set, the arena cleared, and the Vertex and Fragment Shaders are invoked to write the new pixels to the buffer. As *furakutaru* uses double-buffering, it is then necessary to swap the virtual buffer with its hardware counterpart, usually representing a display.

```
/* print_info: prints the information out to stdout regarding the current render
 * state. */

static void print_info ( struct render_state * rs )
{
        printf ( "[ %s ]\nCentre Position: ( %f, %f )\nZoom: %f\nDegree: %d\n" \
                        "Maximum Iterations: %d\nAngle: %d degrees\n",
                        ( rs->type == FRAC_MANDELBROT ) ? "Mandelbrot Set" :
                        "Julia Set", rs->centre_x, rs->centre_y, rs->scale,
                        rs->degree, rs->max_iterations, rs->angle );

        if ( rs->type == FRAC_JULIA )
                printf ( "Seed Position: ( %f, %f )\n", rs->seed_x,
                                rs->seed_y );
}

```

Source-Code Listing 43: Prints the information regarding the current fractal to the standard output. Like everything in *furakutaru*, this exchange is done via a text channel.

```
/* update: updates the screen. This should only be called from a blocking loop;
 * fractals do not mandate constant updates! */

static void update ( struct x_disp_inf * disp_inf, struct x_fb_inf * fb_inf )
{
        glViewport ( 0, 0, disp_inf->win_width, disp_inf->win_height );
        glClearColor ( 0.0f, 0.0f, 0.0f, 1.0f );
        glClear ( GL_COLOR_BUFFER_BIT );
        glDrawArrays ( GL_TRIANGLE_STRIP, 0, 4 );
        glXSwapBuffers ( disp_inf->disp, fb_inf->glx_draw );
}

```

Source-Code Listing 44: The `update` function, despite its short length, invokes the *OpenGL* system to draw the arrays (the Fragment Shaders) to the screen.

6.7 Saving, Importing, and Exporting Fractals

It was stated during the Analysis section, that *furakutaru* must have the ability to save and import fractals, using a CSV file as the method of saving. It is also a requirement, specified by the client, that the System must have the ability to export a fractal to a non-*furakutaru*-specific file, for easy sharing and post-processing.

This section describes the functions within *furakutaru* which enable the various methods of long-term storage.

6.7.1 The Common Status Interface for Long-Term Storage Actions with Related Parsers: `export_status` and `parse_export_status`

As with the other sectors of *furakutaru*, a Common Status Interface, with a complementary parser, has been created to handle errors specifically regarding the save-import-export actions. The parser, `parse_export_status`, found in Source-Code Listing 45, returns a human-readable string for the various status codes in the `export_status` enumerator.

6.7.2 Exporting an image as a *furakutaru* CSV: `csv_export`

The `csv_export` function, listed in Source-Code Listing 46, is the primary export function, providing the user the ability to save an interactive fractal in a particular state. Using a *furakutaru*-specific syntax, this can then be re-loaded into the System at a later date to its exact state.

This function returns `EXPORT_S_EXPORT_OK` on success, or a `EXPORT_S` error-code on failure. The command-line argument `out-dir` is respected, such that the output file will reside in the specified directory. If the user does not specify an output directory, the location of the executable is assumed[8].

6.7.3 Constructing a Pseudo-Random Path: `construct_export_path`

In order to gain the greatest potential of finding a suitable file to which the CSV can be saved, *furakutaru* includes the `construct_export_path` function, used to create append a psuedo-random integer to a prefix, and add a suffix extension: `.tga` or `.csv`.

[8] On some systems, this is the current working directory, which is often distinct to the executable path.

```
enum export_status {
        EXPORT_S_IMGEXP_OK        =   2,
        EXPORT_S_IMPORT_OK        =   1,
        EXPORT_S_EXPORT_OK        =   0,
        EXPORT_S_FOPEN            =  -1,
        EXPORT_S_WRITE_NOTHING    =  -2,
        EXPORT_S_WRITE_INCOMPLETE =  -3,
        EXPORT_S_NO_CONTENDERS    =  -4,
        EXPORT_S_READ             =  -5,
        EXPORT_S_SYNTAX           =  -6,
        EXPORT_S_READ_TOOLONG     =  -7,
        EXPORT_S_READ_TOOSHORT    =  -8,
        EXPORT_S_IMG_OPENGL       = -10
};

/* [ exposed ] parse_export_status: returns a human-readable string of an
 * export_status code. */

const char * parse_export_status ( enum export_status status )
{
        switch ( status ) {
                case EXPORT_S_IMGEXP_OK:
                        return "Image exported successfully";
                case EXPORT_S_IMPORT_OK:
                        return "Imported successfully";
                case EXPORT_S_EXPORT_OK:
                        return "Exported successfully";
                case EXPORT_S_NO_CONTENDERS:
                        return "Could not find any plausible contenders to " \
                               "which to write the information.";
                case EXPORT_S_FOPEN:
                        return "Could not open the file; do you have write " \
                               "access to the current/specified directory?";
                case EXPORT_S_WRITE_NOTHING:
                        return "Nothing could be written.";
                case EXPORT_S_WRITE_INCOMPLETE:
                        return "Only some of the file could be written.";
                case EXPORT_S_READ:
                        return "Could not read the C.S.V. file.";
                case EXPORT_S_READ_TOOLONG:
                        return "The given file was too long to be a valid " \
                               "fractal configuration.";
                case EXPORT_S_SYNTAX:
                        return "A syntax error was found; the file was not " \
                               "loaded.";
                case EXPORT_S_READ_TOOSHORT:
                        return "The C.S.V. file ended unexpectedly; the " \
                               "file was not loaded.";
                case EXPORT_S_IMG_OPENGL:
                        return "OpenGL could not provide the pixels.";
                default:
                        return "Unknown status";
        }
}

```

Source-Code Listing 45: The Common Status Interface for the save-import-export functionality.

This function uses the lightweight `access` built-in *UNIX* function to check the existence of a file. In the extremely unlikely event that the random number has already been used, the function will continue to attempt to find a suitable path. If, after `ATTEMPT_LIMIT`, such a path has not been found, it is assumed there is a systemic issue with the file-system or base directory, and the function quits with a failure code.

`construct_export_path` is listed in Source-Code Listing 47.

6.7.4 Saving a Render to an Image File: `image_export`

The function `image_export`, detailed in Source-Code Listing 48, writes to a fully compliant TGA file—a universal and simple image format which renders itself as completely manageable without the use of extraneous third-party libraries.

`image_export` finds an appropriate file to which it can write, and uses the in-built `glReadPixels` function to take the pixels from the current rendering context and place them into a buffer, allocated on the stack. The appropriate header, followed by the pixel data-stream, is written into the file.

This function conforms to the Common Error Interface for the save-import-export functions, returning `EXPORT_S_IMGEXP_OK` on success, and another descriptive code on failure.

```c
/* [ exposed ] csv_export: puts the current render state into a C.S.V.\ file,
 * respecting the rs.out_dir attribute. Returns EXPORT_S_EXPORT_OK on success,
 * otherwise failure. */

enum export_status csv_export ( struct render_state * rs )
{
        char path [ PATH_MAX + 1 ], buf [ BUF_SZ ];
        FILE * fp = NULL;
        size_t len = 0;

        if ( construct_export_path ( path, rs->out_dir, "csv" ) == -1 )
                return EXPORT_S_NO_CONTENDERS;

        if ( ( fp = fopen ( path, "a" ) ) == NULL )
                return EXPORT_S_FOPEN;

        fputs ( "Writing the C.S.V. data to ", stdout );
        puts ( path );

        /* max iterations, scale, seed X, seed Y, degree, centre X, centre Y,
         * type ( 'm' or 'j' ), rotation */

        len = snprintf ( buf, BUF_SZ - 1, "%d,%f,%f,%f,%d,%f,%f,%c,%d",
                        rs->max_iterations, rs->scale, rs->seed_x, rs->seed_y,
                        rs->degree, rs->centre_x, rs->centre_y, ( rs->type ==
                                FRAC_MANDELBROT ) ? 'm' : 'j', rs->angle );
        buf [ len ] = '\0';

        if ( fwrite ( buf, sizeof ( char ), len, fp ) < len ) {
                fclose ( fp );
                return ( !len ) ? EXPORT_S_WRITE_NOTHING :
                        EXPORT_S_WRITE_INCOMPLETE;
        }

        fclose ( fp );
        return EXPORT_S_EXPORT_OK;
}

```

Source-Code Listing 46: The CSV export functionality, enabling a user to output a fractal to a format which can later be read and re-created by *furakutaru*. In rare cases, the entire file cannot be written, such a full file-system. In this case, `EXPORT_S_WRITE_INCOMPLETE` is output. If none of the file could be written out `EXPORT_S_WRITE_NOTHING` is returned; this is uniform across all export functions.

```c
#define ATTEMPT_LIMIT ( 64 )

/* construct_export_path: constructs a path, using the base and extension. A
 * pseudorandom number is appended to file name.  Returns -1 if a suitable
 * file-name could not be found in ATTEMPT_LIMIT attempts, or zero on success.
 * */

static int construct_export_path ( char output [ PATH_MAX + 1 ],
                const char * base, const char * ext )
{
        static int seeded;
        int attempts = 0;

        if ( !seeded ) {
                srand ( time ( NULL ) );
                seeded = 1;
        }

        do {
                /* find a filename which does not already exist */
                if ( attempts++ > ATTEMPT_LIMIT )
                        return -1;

                output [ snprintf ( output, PATH_MAX, "%s/fura_%d.%s",
                                ( base ) ? base : "./", rand ( ), ext ) ]
                        = '\0';
        } while ( access ( output, F_OK ) == 0 );

        return 0;
}

```

Source-Code Listing 47: A function used to find a suitable path to which an export file can be saved. Unless there is a systemic issue with the base directory, this function uses a random number, seeded with the current time, to find an acceptable file.

```
#define BIT_DEPTH ( 24 )

/* [ exposed ] image_export: writes a full T.G.A.\ image file out containing the
 * current OpenGL render state */

enum export_status image_export ( GLsizei width, GLsizei height,
                const char * out_dir )
{
        size_t img_sz = width * height * ( BIT_DEPTH >> 3 ), bytes = 0;
        GLubyte buf [ img_sz ];
        short hdr [ ] = { 0, 2, 0, 0, 0, 0, width, height, BIT_DEPTH };
        char path [ PATH_MAX + 1 ];
        FILE * fp = NULL;

        if ( construct_export_path ( path, out_dir, "tga" ) == -1 )
                return EXPORT_S_NO_CONTENDERS;

        fputs ( "Exporting image to ", stdout );
        puts ( path );

        if ( ( fp = fopen ( path, "a" ) ) == NULL )
                return EXPORT_S_FOPEN;

        /* request the pixels from OpenGL */
        glReadPixels ( 0, 0, width, height, GL_BGR, GL_UNSIGNED_BYTE, buf );
        if ( glGetError ( ) != GL_NO_ERROR ) {
                fclose ( fp );
                return EXPORT_S_IMG_OPENGL;
        }

        /* write out the header and pixel data */
        if ( ( bytes = fwrite ( &hdr, 1, sizeof ( hdr ), fp ) ) <
                        sizeof ( hdr ) ||
             ( bytes = fwrite ( buf, sizeof ( GLubyte ), img_sz, fp ) ) <
                        sizeof ( GLubyte ) * img_sz ) {
                fclose ( fp );
                return ( !bytes ) ? EXPORT_S_WRITE_NOTHING :
                        EXPORT_S_WRITE_INCOMPLETE;
        }

        fclose ( fp );
        return EXPORT_S_IMGEXP_OK;
}

```

Source-Code Listing 48: Saves the current *OpenGL* render to a TGA file using the built-in *OpenGL* commands to gain direct read-access to the hardware buffer.

6.7.5 Importing a Render from a *furakutaru*-Specific CSV: `import_csv`

Provided a user has a valid *furakutaru* CSV file, the client mandated that this could be imported into the System at any time with full interactive and normal function. The `import_csv`, listed in Source-Code Listing 49, is capable of loading, parsing, and setting the initial state to a pre-existing file.

This function loads the file at the specified path into a buffer, and uses the standard `strtok` function to split the buffer on the delimiting character: a comma. Like the other functions in this category, the Common Status Interface is used, returning `EXPORT_S_IMPORT_OK` on success, and another error code on failure.

6.7.6 Loading a File into the CSV Buffer: `load_csv_buffer`

`load_csv_buffer`, listed in Source-Code Listing 50, loads the CSV file into a fixed buffer, of size `BUF_SZ` characters. Although fixed, any *furakutaru* CSV file will fit comfortably into this range. If the CSV file does exceed this range, the System becomes aware it is not a valid file, and can bail out before performing any expensive read/write operations; if this is the case `EXPORT_S_READ_TOOLONG` is returned.

On success, this function returns `EXPORT_S_IMPORT_OK`, and an error-code on failure.

6.7.7 Parsing the CSV Buffer: `import_subprocessor`

Providing the buffer is successfully populated with the contents of a CSV file, the function `import_subprocessor`, found in Source-Code Listing 51, uses a method similar to the argument-processors, in which a `switch-case` statement and function pointers are used to deal with many distinct cases in a concise fashio many distinct cases in a concise fashion.

```c
#define BUF_SZ ( 128 )

/* [ exposed ] csv_import: imports the C.S.V.\ file into 'rs', reporting the
 * appropriate export_status on completion or error. */

enum export_status csv_import ( const char * path, struct render_state * rs )
{
        char buf [ BUF_SZ ], * token = NULL;
        const char delim [ ] = ",";
        enum export_status status = EXPORT_S_IMPORT_OK;
        enum parse_field field = FIELD_UNKNOWN;

        if ( ( status = load_csv_buffer ( path, buf ) )
                        != EXPORT_S_IMPORT_OK )
                return status;

        token = strtok ( buf, delim );

        while ( token ) {
                field++;
                if ( ( status = import_subprocessor ( token, rs, field ) )
                                != EXPORT_S_IMPORT_OK )
                        return status;

                token = strtok ( NULL, delim );
        }

        degree_sanitise ( & ( rs->degree ) );
        return ( field + 1 != FIELD_LAST ) ? EXPORT_S_READ_TOOSHORT :
                EXPORT_S_IMPORT_OK;
}

```

Source-Code Listing 49: Importing a *furakutaru* CSV file to the renderer.

```c
/* load_csv_buffer: loads the C.S.V.\ file, with a maximum size of BUF_SZ - 1,
 * into the buffer 'buf'. Returns EXPORT_S_IMPORT_OK on success, otherwise an
 * error. */

static enum export_status load_csv_buffer ( const char * path,
                char buf [ BUF_SZ ] )
{
        unsigned long len = 0, bytes = 0;
        FILE * fp = NULL;

        if ( ( fp = fopen ( path, "r" ) ) == NULL )
                return EXPORT_S_FOPEN;

        fseek ( fp, 0, SEEK_END );
        len = ftell ( fp );
        fseek ( fp, 0, SEEK_SET );

        if ( len >= BUF_SZ ) {
                fclose ( fp );
                return EXPORT_S_READ_TOOLONG;
        }

        if ( ( bytes = fread ( buf, sizeof ( char ), len, fp ) ) < len ) {
                fclose ( fp );
                return EXPORT_S_READ;
        }

        buf [ bytes ] = '\0';
        fclose ( fp );
        return EXPORT_S_IMPORT_OK;
}

```

Source-Code Listing 50: Loading a file into a fixed buffer of `BUF_SZ` characters.

Using the same parser-helper functions as seen in Source-Code Listing 29, the sub-processor ensures the syntax and values in the provided file are as-expected, according to the *furakutaru* CSV syntax. An enumerator is also used to group the expected types—the order appearing in the sequentially defined enumerator are the order which they expected to appear in the file; this allows a parent function to increment the action counter and await `FIELD_LAST`.

This function conforms to the Common Status Interface for the save-import-export functions, however does not differentiate between particular points-of-failure when serving the error-code. `EXPORT_S_SYNTAX` indicates a failure, and `EXPORT_S_IMPORT_OK` marks a success.

```
enum parse_field {
        FIELD_UNKNOWN    =  0,
        FIELD_ITERATIONS =  1,
        FIELD_SCALE      =  2,
        FIELD_SEED_X     =  3,
        FIELD_SEED_Y     =  4,
        FIELD_DEGREE     =  5,
        FIELD_CENTRE_X   =  6,
        FIELD_CENTRE_Y   =  7,
        FIELD_TYPE       =  8,
        FIELD_ROTATE     =  9,
        FIELD_LAST       = 10
};


/* import_subprocessor: deals with the C.S.V.-parsing on a per-field basis.
 * Returns the appropriate export_status code: EXPORT_S_IMPORT_OK on success. */

static enum export_status import_subprocessor ( const char * str,
                struct render_state * rs, enum parse_field field )
{
        int ( *setter ) ( void *, const char * ) = NULL;
        void * val_ptr = NULL;

        switch ( field ) {
                case FIELD_ITERATIONS:
                        val_ptr = & ( rs->max_iterations );
                        setter = set_int;
                        break;
                case FIELD_SCALE:
                        val_ptr = & ( rs->scale );
                        setter = set_float;
                        break;
                case FIELD_SEED_X:
                        val_ptr = & ( rs->seed_x );
                        setter = set_float;
                        break;
                case FIELD_SEED_Y:
                        val_ptr = & ( rs->seed_y );
                        setter = set_float;
                        break;
                case FIELD_DEGREE:
                        val_ptr = & ( rs->degree );
                        setter = set_int;
                        break;
                case FIELD_CENTRE_X:
                        val_ptr = & ( rs->centre_x );
                        setter = set_float;
                        break;
                case FIELD_CENTRE_Y:
                        val_ptr = & ( rs->centre_y );
                        setter = set_float;
                        break;
                case FIELD_ROTATE:
                        val_ptr = & ( rs->angle );
                        setter = set_int;
                        break;
                case FIELD_TYPE:
                        switch ( *str ) {
                                case 'm':
                                        rs->type = FRAC_MANDELBROT;
                                        return EXPORT_S_IMPORT_OK;
                                case 'j':
                                        rs->type = FRAC_JULIA;
                                        return EXPORT_S_IMPORT_OK;
                                default:
                                        return EXPORT_S_SYNTAX;
                        }
                default:
                        /* appease compilers; should never happen */
                        return EXPORT_S_SYNTAX;
        }

        return ( setter ( val_ptr, str ) == 0 ) ? EXPORT_S_IMPORT_OK :
                EXPORT_S_SYNTAX;
}

```

Source-Code Listing 51: Using a sub-processor model to parse the arguments on a case-by-case basis, assigning the correct helper function and reporting the success.

Appendix D

Source Code Listing

In the public version of this report, the comprehensive source code listing has been omitted. If you would like an archive of the sources, please contact me at od641@york.ac.uk. The same applies to the presentation video, as described in the following *Testing and Appraisal* section.

Stage IV

Testing and Appraisal

Section 7

System Testing under Normal Conditions

Outlined in the Analysis section and its respective appendices, *furakutaru* has a multitude of strictly defined goals, achieved using a client-led questionnaire. The following section iterates the goals, presenting proofs-of-completion in the form of screenshots and on-line video.

The on-line video can be found at the following F.T.P. address. The video presents all interactive elements of *furakutaru*, such as rotating, panning, and zooming.

> **[URL Redacted]**
>
> For an optimal viewing experience, it is recommended that this video is either streamed with VLC, using the invocation "`vlc [URL Redacted]`", or downloaded to the local machine. A version can also be requested by sending e-mail to **[E-Mail Redacted]**.

7.1 Rendering of the Mandelbrot Set to an Arbitrary Degree

The first goal concerned the rendering of the basic Mandelbrot Set, to an arbitrary integer degree. Figure 7.1 presents a command-line invocation of *furakutaru*, in which a Mandelbrot Set of the fifth degree is being rendered. As discussed in the Analysis, the specified degree p generates a render with a rotational symmetry of $p - 1$, as is confirmed here.

Figure 7.1: A command-line invocation of the *furakutaru* System with the following arguments: `--type mandelbrot --degree 5`. A shorter equivalent of the command-line arguments, also recognised by the System, is `-t mandelbrot -d 5`.

7.2 Rendering of the Julia Set to an Arbitrary Degree and Seed

It is also a requirement of the System to possess the functionality to render the Julia Set. Like the Mandelbrot Set, it must have a variable degree, but also a variable seed. Figure 7.2 presents a Julia Set of the third degree, with a seed of $(-0.22, 0.858)$, equivalent to $-\frac{11}{50} + \frac{22}{25}i$ in the notation of Complex Analysis.

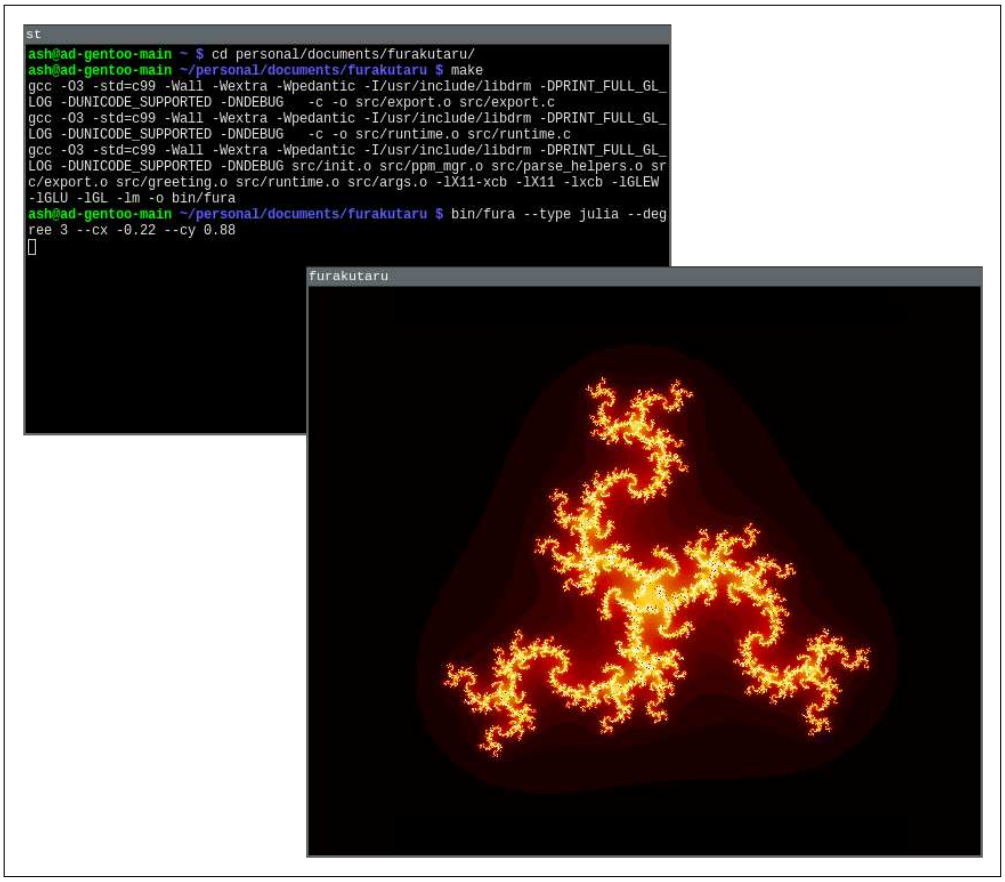


Figure 7.2: The Julia Set of the third degree, rendered with the command-line invocation `--type julia --degree 3 --cx -0.22 --cy 0.858`. Although shortened equivalents do not exist for the `--cx` and `--cy` options, the argument line could be abbreviated to `-t julia -d 3 --cx -0.22 --cy 0.858`. The usage of defensive programming throughout the *furakutaru* code-base disables ambiguities that could arise during the processing of arguments, such as throwing errors on double-definitions, e.g. if a user was the invoke the System with an argument line such as `-t mandelbrot -t julia`.

7.3 Exporting and Important a Fractal to Permanent Storage as Comma-Separated Values

As requested by the client, and shown in the Technical Solution, *furakutaru* should be able to export fractal configurations to a CSV file, which it can later re-load as normal fractals. Figure 7.3 shows a user interactively exporting the current render, and Figure 7.4 shows the user re-loading the fractal, from the generated file, using the `--config`/`-f` command-line arguments.

Should the `--assumeconfig` option have been passed on the command-line in Figure 7.4, the provided configuration would have been assumed as the defaults. The command-line arguments `--outdir`/`-o` could also be specific in Figure 7.3 in order to specify a base directory. This is useful in installations, in which the location executable of the is usually in a directory which is write-inaccessible to non-root users, such as `/usr/bin`.

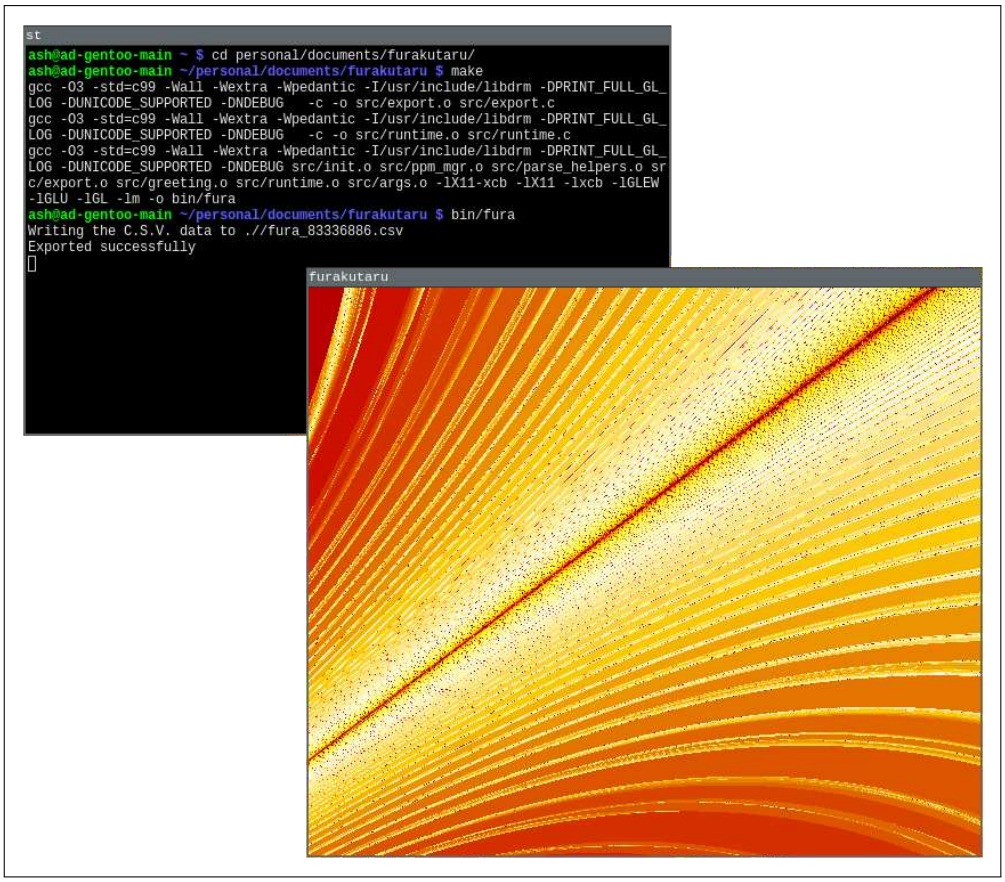


Figure 7.3: A user exporting a heavily zoomed Julia Set, using the interactive key-bind. The `construct_export_path` function, as shown in the Design and Technical Implementation sections, was used to generate a random path to which the file could be written.

7.4 Exporting a Fractal as a TGA Image

The exporting feature concerns creating a TGA image file from the current render. As an extremely simple, yet universal, format, *furakutaru* achieves this functionality by writing out a simple header, consisting largely of the dimensions and a file signature, followed by outputting the pixel data stream, using the built-in OpenGL function `glGetPixels`.

Figure 7.5 presents the user using the built-in key-bind to save the current render to a file in the current directory. This feature follows the same rationale as the CSV-exporter, such that the `--outdir`/`-o` command-line options are respected. The exported image is also shown.

7.5 Loading a Custom Colour Profile

furakutaru also offers the capability to import an externally defined colour profile. Stored in the PPM format, it is expected to be a one-dimensional image, optimally 256 pixels wide. If a pixel does not diverge, it is coloured as the first pixel to the left of the image. Because of this, it is important that users creating their own colour palettes do not select an obnoxiously annoying colour gradient to cover the first sector of the texture, as it will be the majority-colour on most initial renders.

Figure 7.6 displays this feature with a blank, pink, and rainbow-coloured texture, created using the *G.N.U. Image Manipulation Program*.

7.6 Providing an On-Demand Information-Dump

In the interests of user-friendliness, the user, at any time, can use a key-bind to dump information regarding the current render to the standard output device. For Julia Sets, this dump also includes the current seed. The behaviour is shown in Figure 7.7.

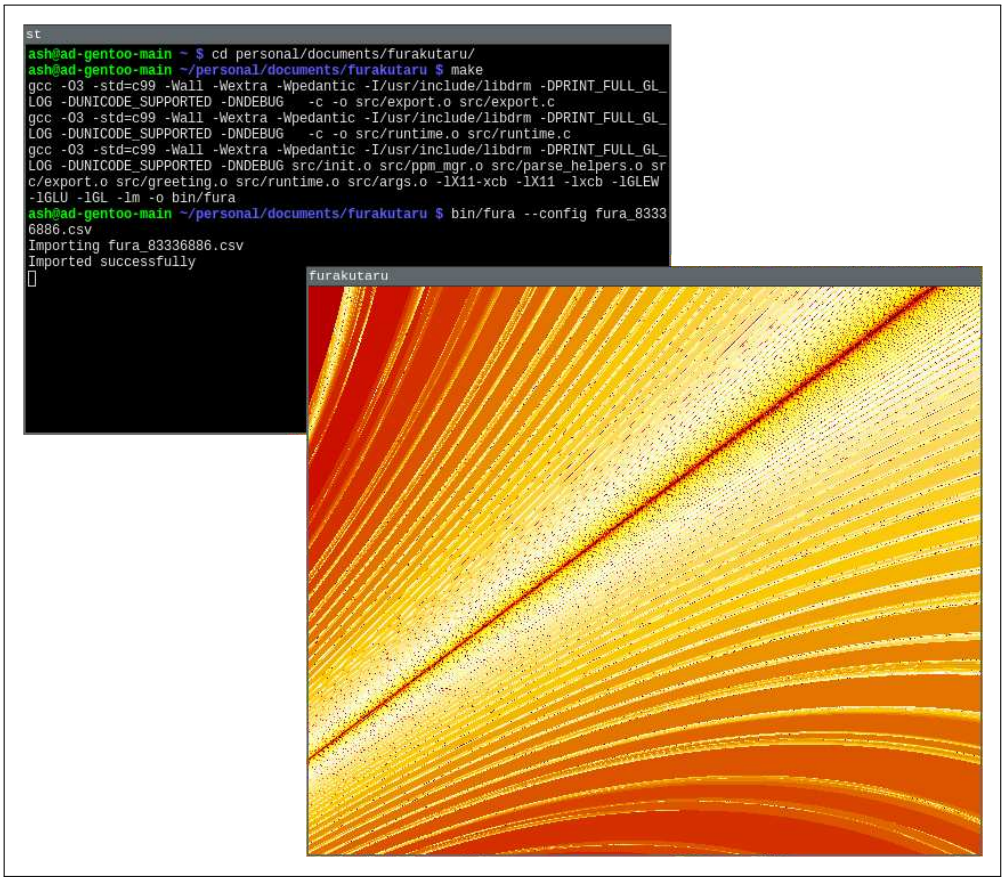


Figure 7.4: A user importing the fractal-configuration which was exported in Figure 7.3.

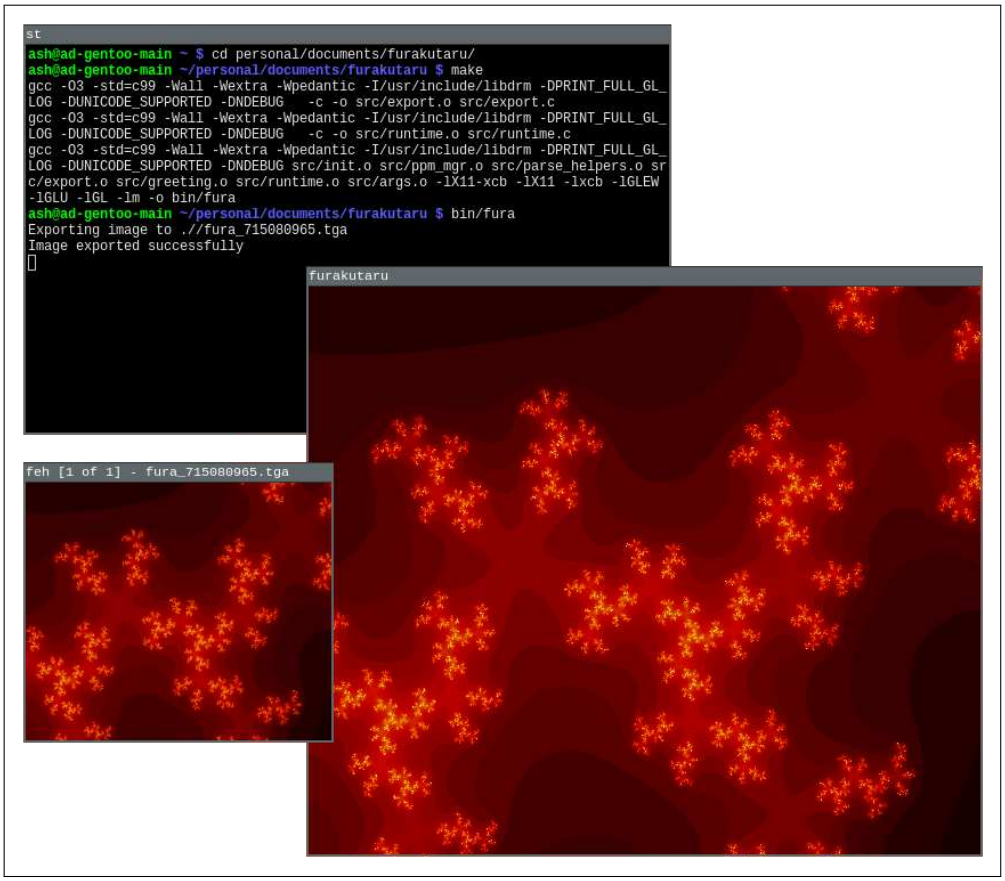


Figure 7.5: The System exhibiting its ability to export the current render as a raster TGA image file, using the same `construct_export_path` as shown in Figure 7.3.

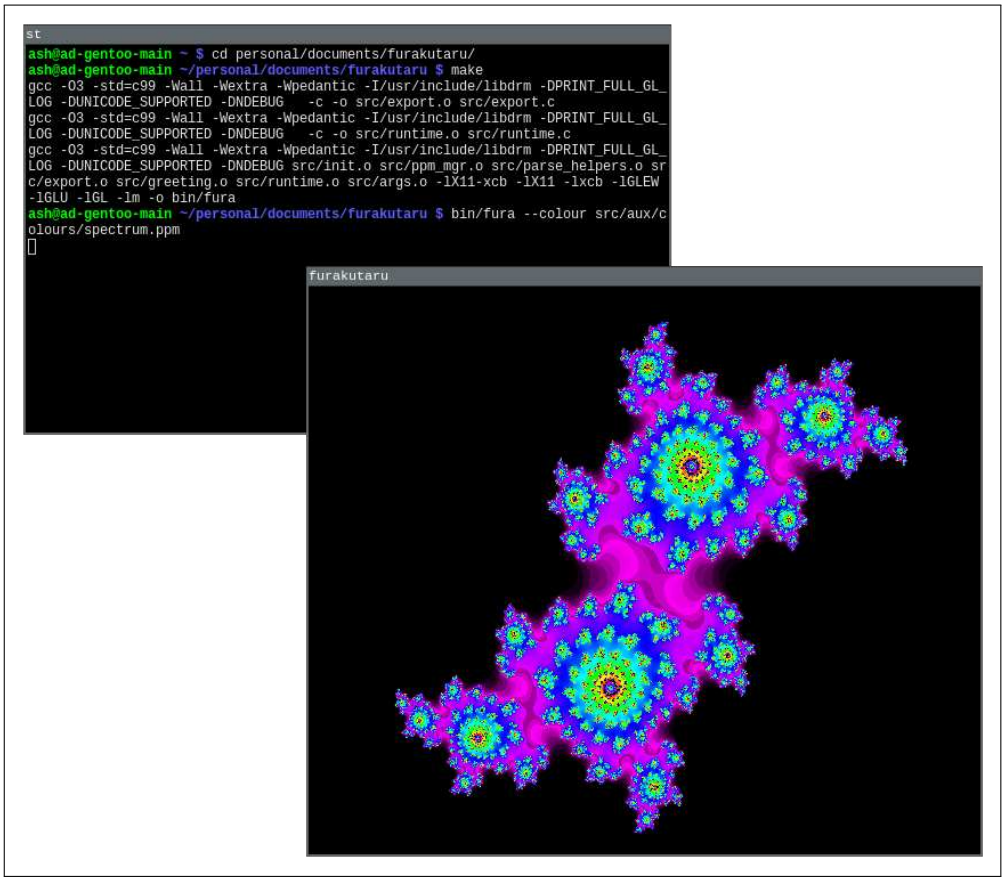


Figure 7.6: Displays the custom selection of a colour profile using the `--colour`/`-c` command-line argument. This particular render shows an unmodified Julia Set.

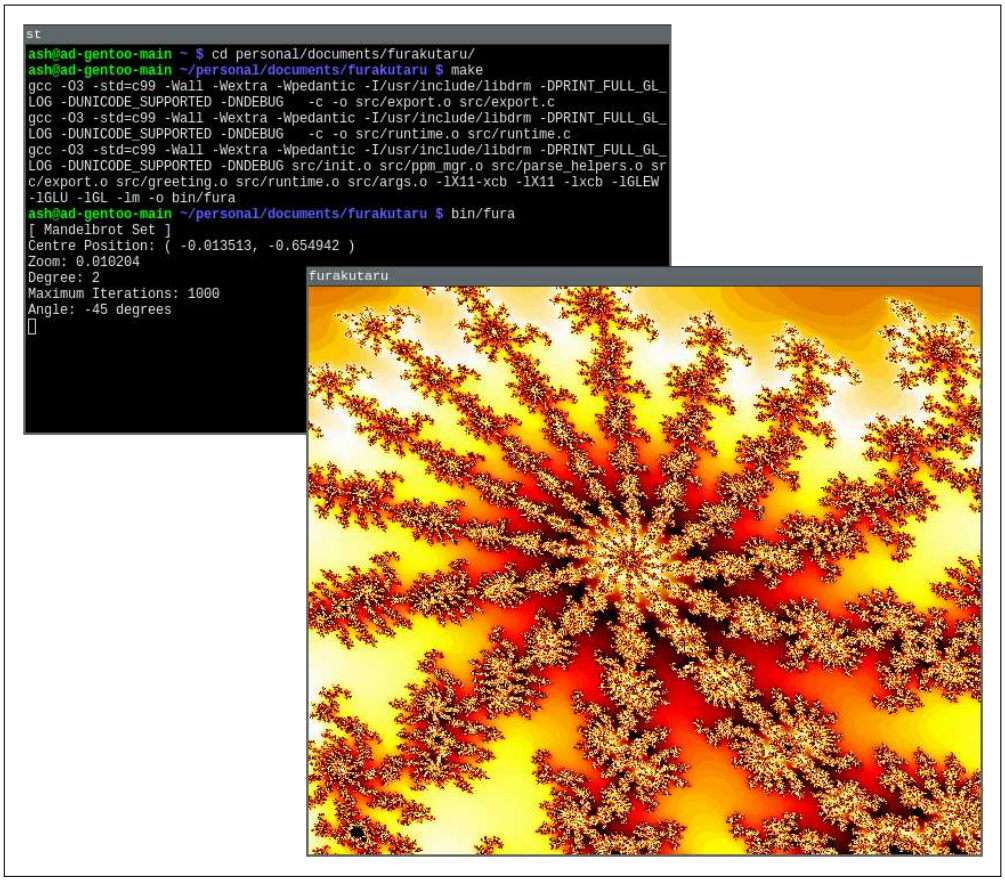


Figure 7.7: An information-dump for a zoomed Mandelbrot Set rotated -45 degrees.

Section 8

System Testing under Erroneous Conditions

As the *furakutaru* System primarily relies upon interactive, keyboard-based input, it is impossible for any user-invoked errors to occur during execution, as the System only responds to keys to which it has an appropriate reaction. The vast majority of errors are as follows:

- **Initialisation.** The initialisation stages, wherein the System established a connection with *X* and OpenGL through *XCB* and *glX* respectively, hold a reasonable margin of error, as they will fail on headless systems, or those without a somewhat-modern graphics card;
- **Parsing of the Configuration Files.** A corrupted CSV being loaded will cause the System to report an error and refuse to load. This is possible only if the storage medium becomes corrupted file-system-wide, or a user manually edits the file without proper knowledge of the internal format;
- **Loading of the Auxiliary Files.** A non-existent or otherwise-inaccessible GLSL shader or colour file causes *furakutaru* to report an error and quit. These are both critical pieces of data, for drawing and colouring the fractal respectively. The colour PPM loader also throws an error if the provided image is not one-dimensional, i.e. has a height greater than a single pixel.

This section explores the behaviour of *furakutaru* under the above conditions, ensuring it does not exhibit undefined or erroneous behaviour, and thus deals with each error graciously, reporting the appropriate error string. Only a handful of the potential errors are explored here, as the majority of errors are impossible to artificially reproduce on a modern system, or are very similar to other errors already tested.

8.1 Memory-Management

Although not an explicit requirement as defined by the client, the first test queries the ability of *furakutaru* to correctly manage memory, ensuring clean-up at all stages, including a non-gracious exit. Figure 8.1 displays the output from a simple run of the *Valgrind, Memcheck* memory-checker, using the options `--show-leak-kinds=all --leak-check=full --suppressions=valgrind.supp -s` to trace[1] potential memory leaks throughout a run of *furakutaru*.

Making use of the suppressions file, found in the Technical Implementation section, the Leak Summary of *Memcheck* indicates that *furakutaru* has **passed** this test.

8.2 Initialisation

8.2.1 No Valid *X* Display

For this test, a headless server is emulated by redefining the `DISPLAY` shell variable to an empty value[2]. On a non-robust system, the client application would report a segmentation fault upon attempting to access a display which does not exist.

Expected *furakutaru* behaviour uses the standard error-reporting system, in which the error-code `STATUS_XDISP` is raised by an initialisation function, and the appropriate human-readable string is printed to the standard error device by the respective code-parsing function.

Figure 8.2 presents a headless simulation. *furakutaru* has **passed** this test.

[1] Although *Valgrind* reports the excessive value of 221,189,510 bytes allocated, it should be recalled that this is an accumulated sum, taken throughout the entire execution of the System. The actual memory footprint of *furakutaru* is far less than 210MiB, and at its peak, uses approximately 0.06003189 MiB of R.A.M.

[2] The `DISPLAY` variable is used by the *X* server to inform clients of the preferred *X* display. On the vast majority of systems, this is set to `:0`, as most desktop users do not use more than one display.

```
==24217== Memcheck, a memory error detector
==24217== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24217== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==24217== Command: bin/fura
==24217==
==24217==
==24217== HEAP SUMMARY:
==24217==    in use at exit: 100,426 bytes in 483 blocks
==24217==  total heap usage: 8,520 allocs, 8,037 frees, 221,189,510 bytes allocated
==24217==
==24217== LEAK SUMMARY:
==24217==   definitely lost: 0 bytes in 0 blocks
==24217==   indirectly lost: 0 bytes in 0 blocks
==24217==     possibly lost: 0 bytes in 0 blocks
==24217==   still reachable: 0 bytes in 0 blocks
==24217==       suppressed: 100,426 bytes in 483 blocks
==24217==
==24217== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 5 from 5)
--24217--
--24217-- used_suppression:  23 <dlopen-fp> valgrind.supp:2 suppressed: 99,828 bytes in 43
    blocks
--24217-- used_suppression:  37 <nvidia-fp> valgrind.supp:27 suppressed: 60,136 bytes in
    189 blocks
--24217-- used_suppression:  29 <dlinit-fp> valgrind.supp:10 suppressed: 21,016 bytes in
    155 blocks
--24217-- used_suppression:  67 <nvidia-glx-fp> valgrind.supp:17 suppressed: 1,100 bytes in
     95 blocks
--24217-- used_suppression:   1 <xextcreate-fp> valgrind.supp:36 suppressed: 24 bytes in 1
    blocks
==24217==
==24217== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 5 from 5)
```

Figure 8.1: *furakutaru* running under a *Valgrind* virtual machine, used to track memory errors.

```
$ export DISPLAY=""
$ ./furakutaru
  XCB initialisation error -1:  could not open X display
  [ process exited with EXIT_FAILURE ]
```

Figure 8.2: A headless simulation, in which *furakutaru* reports the expected error message and quits graciously.

8.2.2 OpenGL is Unsupported by the *X* Server

In the case of an extremely old or minimalistic system, the *X* server may exist and be accessible via *XCB*, but lack OpenGL support. The OpenGL feature test, shown in Figure 8.3, shows the output of *furakutaru* on an *EeePC 1000* 32-bit mini-notebook running *Gentoo Linux*. Although its *Intel Atom N270* is capable of running a minimalistic *X* server, no aspects of hardware-acceleration, and thus OpenGL, are possible on this setup.

```
$ ./furakutaru
  OpenGL initialisation error -8:  OpenGL is not supported by the current X server
  [ process exited with EXIT_FAILURE ]
```

Figure 8.3: A test of the System on a configuration which has an active *X* instance with *XCB* bindings, but does not support the OpenGL subsystem and/or *glX*.

The *furakutaru* System has **passed** this test.

8.2.3 A GLSL Shader Could not be Compiled

If a Shader file is corrupted, *furakutaru* should refuse to load, while showing both a *furakutaru* error, in addition to a compilation-trace provided by OpenGL. The following test ensures that the System is able to graciously deal with erroneous Shader files, verifying that such a script does not cause the System to crash, leak memory, or invoke undefined behaviour.

Figure 8.4 inspects the behaviour of *furakutaru* when a variable declaration has been removed from a Shader.

```
$ ./furakutaru
  0(71) :  error C1503:  undefined variable "frag_colour"
  Shader initialisation error -14:  the shader could not be compiled
  [ process exited with EXIT_FAILURE ]
```

Figure 8.4: The behaviour of *furakutaru* when passed an erroneous Shader program.

As expected, the System correctly reports the error to the standard error device, and exits with the appropriate code. *furakutaru* has **passed** this test.

8.2.4 The GLSL Version is Unsupported

furakutaru mainly targets modern systems, as requested by the Client. Thus, some older video cards and processors may be incapable of meeting the requirements which are explicitly defined with the `#version` string, seen at the top of each Shader auxiliary file. If this is the case, *furakutaru* should exhibit similar behaviour to other OpenGL-initialisation errors, reporting a human-readable string to the standard error device, and quit graciously with a failure-indicating status code.

Figure 8.5 presents the behaviour of the System on hardware which does not meet the aforementioned standards.

```
$ ./furakutaru
  0(7) :  error C0201:  unsupported version 500
  [ process exited with EXIT_FAILURE ]
```

Figure 8.5: The System attempting to load a GLSL Shader which defines a requirement for a version which is not supported by the OpenGL subsystem.

furakutaru has **passed** this test, as the expected behaviour is followed, and the returned code is appropriate.

8.3 File System Restrictions

The following tests inspect the behaviour of *furakutaru* when the file-system disallows access to certain files to which the System is attempting to write/read. For these tests, two directories have been created on a UNIX system: `/noread` and `/nowrite`, with the permission strings `drw--w--w-` and `drwxr-xr-x`, both under the ownership of `root:root`.

8.3.1 Attempting to Write to a Forbidden Directory

For this test, *furakutaru* is invoked with the `--outdir/-o` option, instructing the base directory to be the forbidden `/nowrite` directory. Upon loading, the System was told to first save a CSV configuration, followed by a TGA export. The expected behaviour is that the System refuses to write to the directory, as files in `/nowrite` cannot be opened, although the directory itself can be read and explored. *furakutaru* should not quit, nor should there be any effect on the current render, however an error should be sent to the standard error device, and the operation halted. This behaviour is transcribed in Figure 8.6.

As the System presented the expected reaction, *furakutaru* has **passed** this test.

```
$ ./furakutaru --outdir /nowrite
  Writing the CSV data to /nowrite/fura_1076436437.csv
  Could not open the file; do you have write-access to the specified directory?
  Exporting image to /nowrite/fura_1270594253.tga
  Could not open the file; do you have write-access to the specified directory?
```

Figure 8.6: The *furakutaru* System attempting to output various files to a directory on which it does have write-access.

8.3.2 Attempting to Load a Configuration from a Forbidden Directory

The directory `/noread` is used here to observe *furakutaru*'s behaviour when provided with a fractal configuration at a path which cannot be accessed by the current user under which the System is being executed. Figure 8.7 shows the behaviour of the System when invoked with the `--config/-f` command-line argument, instructing a load from the `/noread` directory.

```
$ ./furakutaru --config /noread/cfg.csv
  Importing /noread/cfg.csv
  Could not open the file; do you have read-access to the specified directory
  [ process exited with EXIT_FAILURE ]
```

Figure 8.7: *furakutaru* attempting to load a fractal configuration from a path on which it does not have read-access.

As *furakutaru* correctly detected, reported, and exited graciously from this error, the System has **passed** this test.

8.4 Argument-Processing Errors

The wide range of arguments provided by *furakutaru* can be intimidating for an ill-informed first-time user of the System, especially if they lack a familiarity or basic understanding of the command-line. Although *furakutaru* offers a `--help/-h` option for outputting a comprehensive guide to the command-line options, it remains integral that arguments are verified for their validity, both in value and context, in the interests of ensuring and preserving a stable render-state.

The following sections explore the behaviour of *furakutaru* when provided with erroneous arguments by malicious or uneducated users. The expected behaviour for all these tests generally conforms to the process of reporting the error, performing any appropriate clean-up[3], and quitting immediately. *furakutaru* should always call a fatal error on erroneous arguments, and not ignore them while printing a mere 'warning'.

8.4.1 An Argument Double-Definition

In order to prevent argument-lines containing conflicting arguments, such as `--type mandelbrot --type julia`, arguments may only be defined once, even if they do not conflict. Figure 8.8 shows the response of *furakutaru* in this situation, in which `--type` is defined twice.

```
$ ./furakutaru --type julia --type mandelbrot --cx 0.2
  Argument error:  "--type", Argument was doubly defined ( -5 )
  [ process exited with EXIT_FAILURE ]
```

Figure 8.8: An invocation of the System with a doubly defined `--type` argument. Execution is halted immediately once the second `--type` is encountered by the argument-processor.

The *furakutaru* System has **passed** this test.

[3] By design, the argument-processor is called as early as possible. This enables the System to do as little clean-up as possible, should the arguments contain an error, as no OpenGL or *X Windows* connections have been established.

8.4.2 Missing Operands

The majority of arguments require a following operand, the notable exceptions being `--help/-h`, `--verbose/-v`, `--lock/-l`, and `--assumeconfig`. If any other argument is listed as the final argument in the string, *furakutaru* can immediately detect that there cannot be another operand. If the argument is followed by another argument, the latter is interpreted as an operand and will cause an error in the operand-parsing stage.

Figure 8.9 shows this behaviour, such that the only argument passed indicates the specification of the Centre X co-ordinate, despite it being the sole argument.

```
$ ./furakutaru --cx
  Argument error:  "--cx", Expected succeeding argument ( -1 )
  [ process exited with EXIT_FAILURE ]
```

Figure 8.9: The System responding to an erroneous invocation, in which the `--cx` argument is provided with no succeeding operand.

As the System exhibited the expected report-clean-exit process, *furakutaru* has **passed** this test.

8.4.3 Invalid Operands

In the case that an operand has been provided, the argument sub-processor must proceed to parse the operand into its expected form, as indicated by the nature of its preceding argument op-code. Figure 8.10 provides an example of such a situation, in which the `--iteration/-i`, usually intended to specify the maximum iterations, is provided a floating-point number. As the expected operand data-type is an unsigned integer, the System should refuse the argument-line and exit with the appropriate status-code.

```
$ ./furakutaru --iteration 10.5
  Argument error:  "--iteration", Argument operand was invalid ( -2 )
  [ process exited with EXIT_FAILURE ]
```

Figure 8.10: *furakutaru* reporting an erroneous operand, in which a floating-point was given in-place of an integer.

The System accurately reported the appropriate error, and has thus **passed** this test.

8.4.4 Unrecognised Argument/Op-Code

If the System encounters an unknown string in the argument-line and is not expecting an operand, it must raise the issue of an Unrecognised Argument. In this case, the System should not ignore and continue, but request a fatal exit from the caller. Figure 8.11 presents this behaviour, when the foreign `--ewb` argument is provided. *furakutaru* should also inform the user of which argument was undefined, as a long argument-line could cause confusion when a nondescript error is reported.

```
$ ./furakutaru --ewb
  Argument error:  "--ewb", Unrecognised argument
  [ process exited with EXIT_FAILURE ]
```

Figure 8.11: The *furakutaru* System handling an unknown argument, `--ewb`.

As *furakutaru* accurately described the error, the System has **passed** this test.

This concludes the manual error-testing of *furakutaru*, in which the System successfully **passed** each test to which it was subjected, correctly providing descriptive errors, comprehensive clean-up, and indicative status-codes.

Section 9

Appraisal and Client-Feedback

Referencing the goals outlined in the Analysis section, *furakutaru* has met its targets entirely. Whilst some of the nice-to-have features, such as an interactive colour editor, were omitted, their inclusion was not specified by the client, and their absence does not hinder the application or its defined objectives. The following verbatim text details the Client's response to the presented product, in which the feedback is aggregated via a response-led questionnaire.

Considering the overwhelmingly positive feedback provided by the Client, this project is considered a success.

```
(Q.) On the 11th July, 2019, the goals and objectives for the 'furakutaru'
       System were defined. Reviewing our e-mail exchange, do you believe that
       these objectives were met to an acceptable standard ?
(A.) "Yes. Through my usage of 'furakutaru' to date, I am incredibly satisfied
       with the Product. The defined goals were met in their entirety, in
       addition to some complementary features which enable easier usage of
       the System, such as a comprehensive and well-documented command-line
       interface."

(Q.) Do you find the System easy to use, and do you believe that your students
       will share your opinion?
(A.) "My extensive use-sessions of 'furakutaru' are delightfully easy to use,
       partly due to its simplicity, but also its well-documented nature with
       human-readable errors and detailed fractal-information reports. I am
       sure that the students which are familiar with the command-line, of
       whom are plentiful, will have no issues interacting with the System
       after a few minutes of training, however I feel as though some of the
       lesser students may have short-lived issues grasping the concept of
       command-line arguments. Thankfully, the majority of 'furakutaru' is
       interactive, and the skillful design mandates no knowledge of
       command-line arguments for basic operation."

(Q.) Are there any features which could be added in order to make the System
       easier to use or be of greater use?
(A.) "My only qualm with the System is its inability to export images as
       anything other than T.G.A. Although it is recognised as a common image
       format and is thus interpretable by the majority of image viewers,
       students may prefer their exports in more universal formats such as
       P.N.G. or J.P.E.G. However, I also realise that the inclusion of such
       functionality may compromise the lightweight-nature of System, as the
       complex compression methods of the P.N.G. and J.P.E.G. formats seems,
       from my experience, to be a complicated and timely operation."

(Q.) Finally, did you encounter any issues or difficulties whilst using the
       System?
(A.) "Aside from the aforementioned issue regarding the small range of image
       formats to which the System can export, I, or any of my
       student-volunteers, ran into no issues. Our uniform submission is that
       the System entirely satisfies our needs, and we are extraordinarily
       pleased with the result. The high-calibre students of [REDACTED] have
       expressed their personal gratitude."
```

Cited Works

The following works have served immensely throughout my research of fractals and subsequent creation of *furakutaru*. Texts which proved particularly integral are highlighted in **bold**.

[Man82] Benoit Mandelbrot. *The Fractal Geometry of Nature*. Times Books, 1982. ISBN: 978-0-71-671186-5.

[Hut81] John Hutchinson. "Fractals and Self-Similarity". In: *Indiana University Mathematics Journal* (30 1981), p. 2. DOI: 10.1512/iumj.1981.30.30055.

[Man80] Benoit Mandelbrot. "Fractal aspects of the iteration of $z \mapsto \lambda z(1-z)$ for complex λ and z". In: *Fractals and Chaos*. Vol. 357. New York, New York, United States: Springer, 1980, p. 40. DOI: 10.1007/978-1-4757-4017-2_3.

[Met94] Wolfgang Metzler. "The 'mystery' of the quadratic Mandelbrot set". In: *American Journal of Physics* 62.9 (1994), p. 814. DOI: 10.1119/1.17465.

[BM81] Robert Brooks and J. Matelski. "**The Dynamics of 2-Generator Subgroups of PSL(2, C)**". In: *Riemann Surfaces and Related Topics: Proceedings of the 1978 Stony Brook Conference*. Ed. by Irwin Kra and Bernard Maskit. [These images were darkened and had compression artefacts removed by the *G.N.U. Image Manipulation Program*.] Princeton University Press, 1981, pp. 70, 71. ISBN: 978-0-69-108267-7.

[Dou86] Adrien Douady. "Julia Sets and the Mandelbrot Set". In: *The Beauty of Fractals: Images of Complex Dynamical Systems*. Ed. by Heinz-Otto Peitgen and Peter H. Richter. Berlin, Germany: Springer, 1986, p. 161. ISBN: 978-3-642-61719-5. DOI: 10.1007/978-3-642-61717-1_13.

[Lei90] Tan Lei. "**Similarity between the Mandelbrot set and Julia sets**". In: *Communications in Mathematical Physics*. Vol. 134. Germany: Springer-Verlag, 1990, p. 587. DOI: 10.1007/BF02098448.

[Ava09] Stephanie Avalos-Bock. "Fractal Geometry: The Mandelbrot and Julia Sets". In: *University of Chicago, Department of Mathematics* (2009), p. 1. URL: https://www.math.uchicago.edu/~may/VIGRE/VIGRE2009/REUPapers/Avalos-Bock.pdf.

[Fel12] David P. Feldman. *Chaos and Fractals: An Elementary Introduction*. Oxford University Press, 2012, pp. 3, 4. ISBN: 978-0-19-956644-0.

[Jun02] Wolf Jung. "Homeomorphisms on Edges of the Mandelbrot Set". PhD thesis. RWTH Aachen University, 2002, p. 23.

[Sch99] Dierk Schleicher. "On Fibers and Renormalization of Julia Sets and Multibrot Sets". In: *arXiv Mathematics e-prints* (1999), p. 3. arXiv: math/9902156 [math.DS].

[Dev18] Ultra Fractal Developers. *Ultra Fractal 6 Manual*. [On-line; accessed 1st July 2019]. 2018. URL: https://www.ultrafractal.com/download/uf6-manual.pdf.

[FL02] Andrew Forward and Timothy Lethbridge. "The Relevance of Software Documentation, Tools and Technologies: A Survey". In: *Proceedings of the 2002 ACM Symposium on Document Engineering*. DocEng '02. New York, New York, United States: Association for Computing Machinery, 2002, p. 29. ISBN: 1-58113-594-7. DOI: 10.1145/585058.585065.

[Nor89] Alan Norton. "Julia sets in the quaternions". In: *Computers & Graphics* 13.2 (1989), p. 273. ISSN: 0097-8493. DOI: 10.1016/0097-8493(89)90071-X.

[War14] Brian Ward. *How Linux Works: What Every Superuser Should Know*. No Starch Press, 2014, p. 21. ISBN: 978-1-59-327088-9.

[Nur+09] Nurzhan Nurseitov et al. "Comparison of JSON and XML data interchange formats: a case study." In: *Caine* 9 (2009), p. 157.

[The16] The PPM Developers. *Netpbm Colour Image Format*. 2016. URL: http://netpbm.sourceforge.net/doc/ppm.html.

[Coc00] Alistair Cockburn. *Writing effective use cases*. Addison-Wesley Professional, 2000, p. 28. ISBN: 978-0-20-170225-5.

[HG00] Loes M. van Herten and Louise J. Gunning-Schepers. "Targets as a tool in health policy: Part I: lessons learned". In: *Health Policy* 53.1 (2000), p. 1. ISSN: 0168-8510. DOI: 10.1016/S0168-8510(00)00081-6.

[Bul+03] J. M. Bull et al. "Benchmarking Java against C and Fortran for scientific applications". In: *Concurrency and Computation: Practice and Experience* 15.3-5 (2003), p. 427. DOI: 10.1002/cpe.658.

[CLM05] Xing Cai, Hans Petter Langtangen, and Halvard Moe. "On the Performance of the Python Programming Language for Serial and Parallel Scientific Computations". In: *Scientific Programming* 13 (1 2005), p. 31. DOI: 10.1155/2005/619804.

[Loy+02] Marc Loy et al. *Java Swing*. O'Reilly Media, Inc., 2002, p. 3. ISBN: 978-1-56-592455-0.

[Lut01] Mark Lutz. *Programming Python*. O'Reilly Media, Inc., 2001, p. 251. ISBN: 978-0-59-600085-1.

[Mit13] Shaun Mitchell. *SDL Game Development*. Packt Publishing, 2013, p. 6. ISBN: 978-1-84-969682-1.

[JR93] Eric F. Johnson and Kevin Reichard. *Professional Graphics Programming in the X Window System*. 1st ed. M.I.S. Press, 1993, p. 2. ISBN: 1-55828-255-6.

[YHL11] Chao-Tung Yang, Chih-Lin Huang, and Cheng-Fang Lin. "Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters". In: *Computer Physics Communications* 182.1 (2011), pp. 266–269. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2010.06.035.

[Dro19] Michael Droettboom. *Understanding JSON Schema*. [On-line; accessed 4th July 2019]. 2019. URL: https://json-schema.org/understanding-json-schema/UnderstandingJSONSchema.pdf.

[Vas10] Tzvetomir Ivanov Vassilev. "Comparison of Several Parallel API for Cloth Modelling on Modern GPUs". In: *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*. CompSysTech '10. New York, N.Y., U.S.A.: Association for Computing Machinery, 2010, p. 133. ISBN: 9781450302432. DOI: 10.1145/1839379.1839403.

[ND10] John Nickolls and William J. Dally. "The GPU Computing Era". In: *IEEE Micro* 30.2 (2010), p. 56. ISSN: 0272-1732. DOI: 10.1109/MM.2010.41.

[LR98] Linda Lamb and Arnold Robbins. *Learning the vi Editor*. Sebastopol, California, United States: O'Reilly Media, Inc., 1998, p. 3. ISBN: 978-1-56-592426-0.

[MPT74] Doug McIlroy, E. N. Pinson, and B. A. Tague. "Unix Time-Sharing System: Foreword". In: *The Bell System Technical Journal* (1974), pp. 1902–1903.

[Sal94] Peter H. Salus. *A Quarter-Century of UNIX*. Reading, Massachusetts, United States: Addison-Wesley, 1994, p. 52. ISBN: 978-0-20-154777-1.

[Dhu+03] Dinakar Dhurjati et al. "Memory Safety without Runtime Checks or Garbage Collection". In: *Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems*. LCTES '03. New York, N.Y., U.S.A.: Association for Computing Machinery, 2003, pp. 69–80. ISBN: 1581136471. DOI: 10.1145/780732.780743.

[HH95] Daryl H. Hepting and John C. Hart. "The Escape Buffer: Efficient Computation of Escape Time for Linear Fractals". In: *Proceedings of Graphics Interface '95*. GI '95. Quebec, Quebec, Canada: Canadian Human-Computer Communications Society, 1995, pp. 204–214. ISBN: 0-96-953384-5.

[Liu+15] Miao Liu et al. "Distributional Escape-Time Algorithm Based on Generalized Fractal Sets in a Cloud Environment". In: *Chinese Journal of Electronics* 24.1 (2015), pp. 124–127. ISSN: 1022-4653. DOI: 10.1049/cje.2015.01.020.

[HPS91] Daryl Hepting, Przemyslaw Prusinkiewicz, and Dietmar Saupe. "Rendering Methods for Iterated Function Systems". In: *Fractals in the Fundamental and Applied Sciences*. Ed. by Jose Marques Henriques, Luis Filipe Penedo, and Heinz-Otto Peitgen. Amsterdam, The Netherlands: North-Holland, 1991, pp. 1–39. ISBN: 0444887571.

[YH04] Nergiz Yaz and H. Hilmi Hacısalihoğlu. "On Fractal Colouring Algorithms". In: *Dynamical Systems and Applications* 5.10 (2004), pp. 706–711.

[Gar+00] Francisco Garcıa et al. "Coloring Dynamical Systems in the Complex Plane". In: *Unnamed Publisher: The University of the Basque Country* (0). n.d.

[Kim15] Theodore Kim. "Quaternion Julia Set Shape Optimization". In: *Computer Graphics Forum* 34.5 (2015), pp. 167–176. DOI: 10.1111/cgf.12705.

[Bri88] Mark Bridger. "Looking at the Mandelbrot Set". In: *The College Mathematics Journal* 19.4 (1988), pp. 353–363. DOI: 10.1080/07468342.1988.11973139.

[Fou09] The *XOrg* Foundation. ***Basic Graphics Programming with the XCB Library***. [*X11* version R. 7.5]. 2009. URL: https://www.x.org/releases/X11R7.5/doc/libxcb/tutorial/#colormap.

All images of fractals, aside from [BM81], were generated by myself using Wolfram *Mathematica 11* or *furakutaru*. All post-processing was performed in the *GNU Image Manipulation Program* (raster) and *Inkscape* (vector) on *Gentoo Linux*.

The CPU, GPU, and Server/Build icons were created by *Linector* of www.flaticon.com.