CMPUT 229

Lab #3: Fractal

CMPUT 229

Background

Overview

Rendering the Mandelbrot Fractal:

In this lab you will write functions to render the Mandelbrot fractal.

The Mandelbrot fractal is definied using complex numbers.

To perform computations on complex numbers, we will have to use RISC-V's floating point capabilities.

Rendering will be performed using the GLIR library.

Complex Numbers

What are Complex Numbers:

Complex numbers have a real and imaginary part. For example, the number `2 + 3i` has real part `2` and imaginary part `3`.

The special value i statisfies `i² = -1`.

Algebraic operations on complex numbers work the same as on real numbers, if `i` is treated as a variable, and `i² = -1`.

Complex addition:

$$
\begin{aligned}
&(3+2i)+(4-i) \\
&=(3+4)+(2-1)i \\
&=7+i
\end{aligned}
$$

Complex Multiplication:

$$
\begin{aligned}
&(3+2i)\cdot(4-i) \\
&=(3\cdot 4)+(3\cdot -i)+(2i\cdot 4)+(2i\cdot -i) \\
&=12-3i+8i+-2i^2 \\
&=14+5i
\end{aligned}
$$

The Mandelbrot Set

What is the Mandelbrot Set

The Mandelbrot set is a set of complex numbers definied by a recursive function:

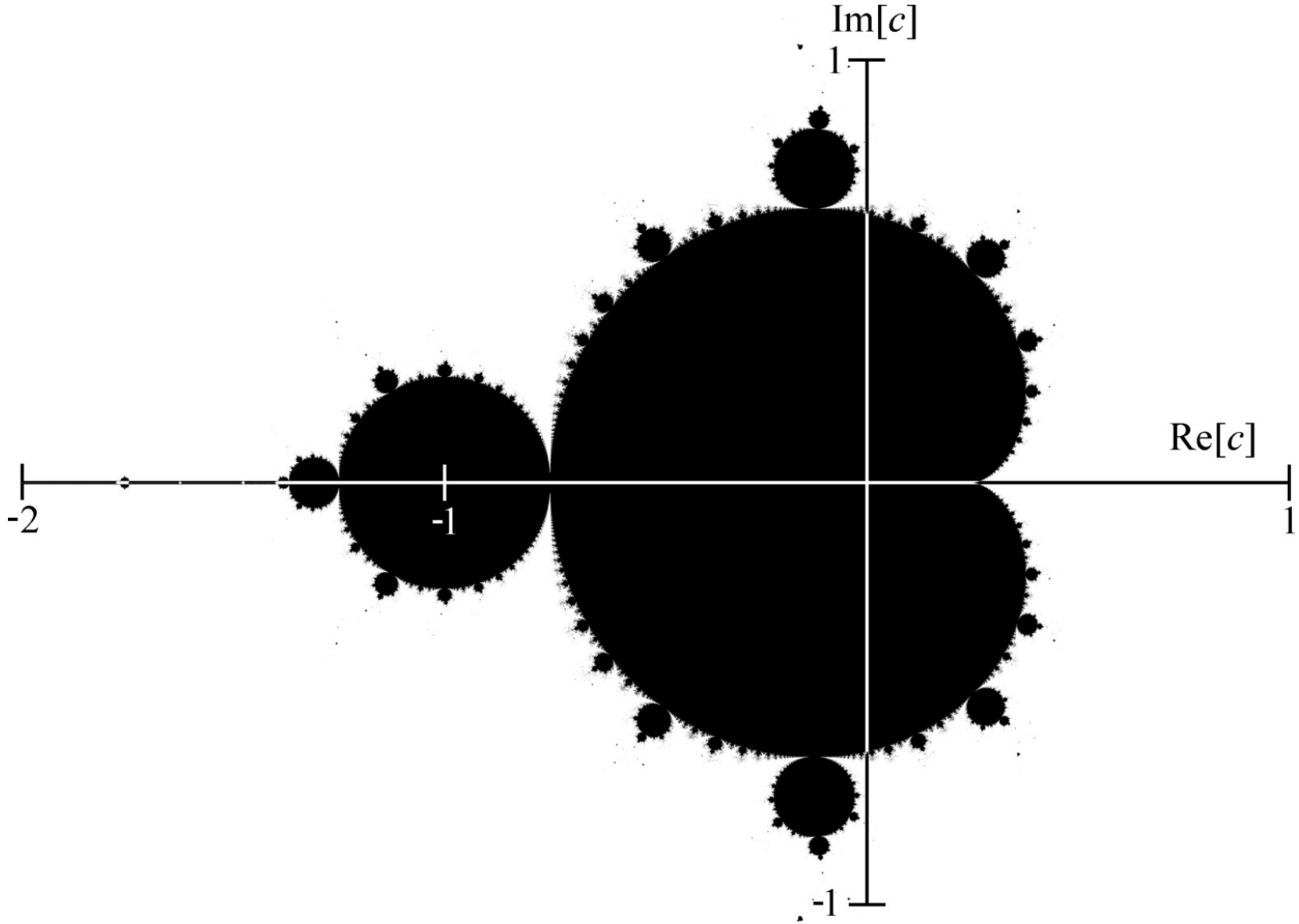
$z_0 = c$

$z_n = z^2_{n-1} + c$

A given complex number c belongs to the Mandelbrot set if z_n does not tend to infinity as n increases.

It is known that no complex number z such that $|z| \geq 2$ is in the Mandelbrot set. Thus only iterations where $|z_n| < 2$ need be computed.

The Mandelbrot Set

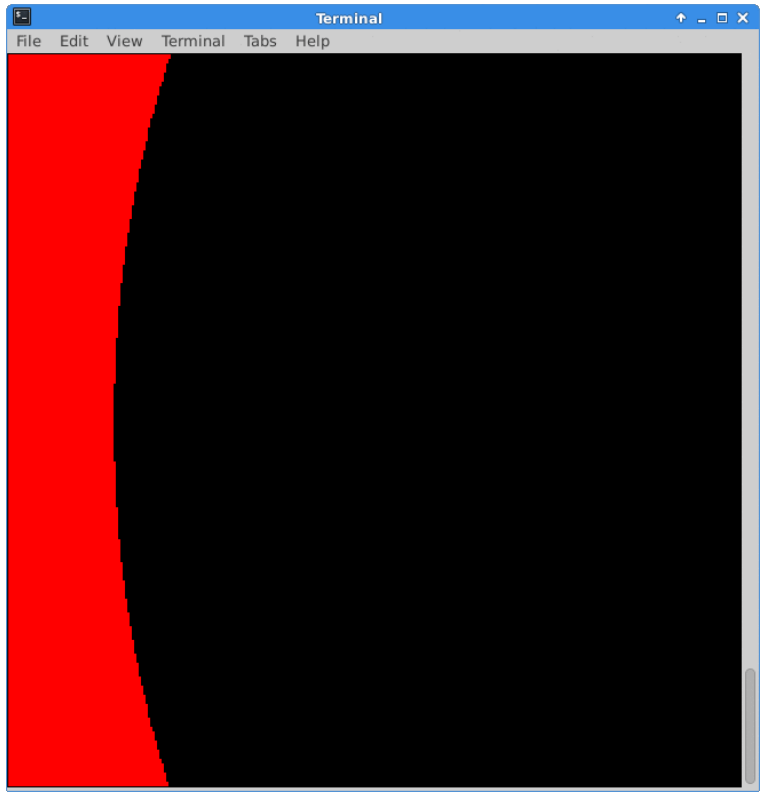


Mandelbrot Animations

Animating the Mandelbrot Set

The number of iterations until $|z_n| > 2$ is called the escape time.

For a given complex number, we can colour that point based on the escape time.



GLIR – Graphics Library for RISC-V

Graphical output will be handled with GLIR.

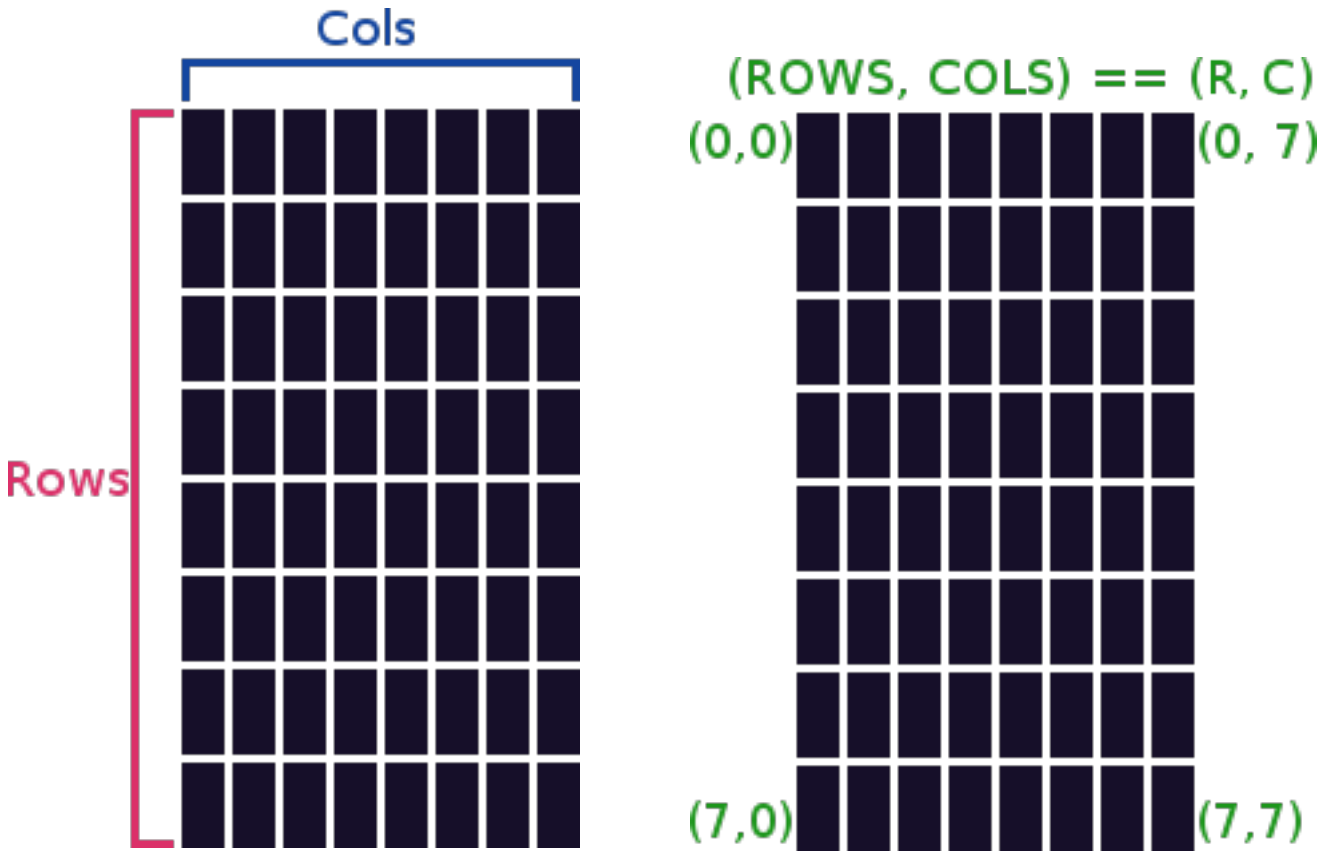
GLIR emulates graphics by printing cells onto the terminal window.

Each cell can have a character (symbol), a foreground colour, and a background colour.

The Terminal

Form of the Terminal

- Grid of cells, with rows and columns.
- Cells are rectangular
- Coordinates are of the form (row, column).



Coordinate Systems

There are two coordinates systems in this assignment:

Complex coordinates, used by the Mandelbrot fractal.

Terminal coordinates, used to draw to the terminal.

To render the fractal, we must convert between coordinates.

Mandelbrot Coordinates

Mandelbrot Coordinates -> Complex Plain:

The Mandelbrot set is defined on the complex plane.

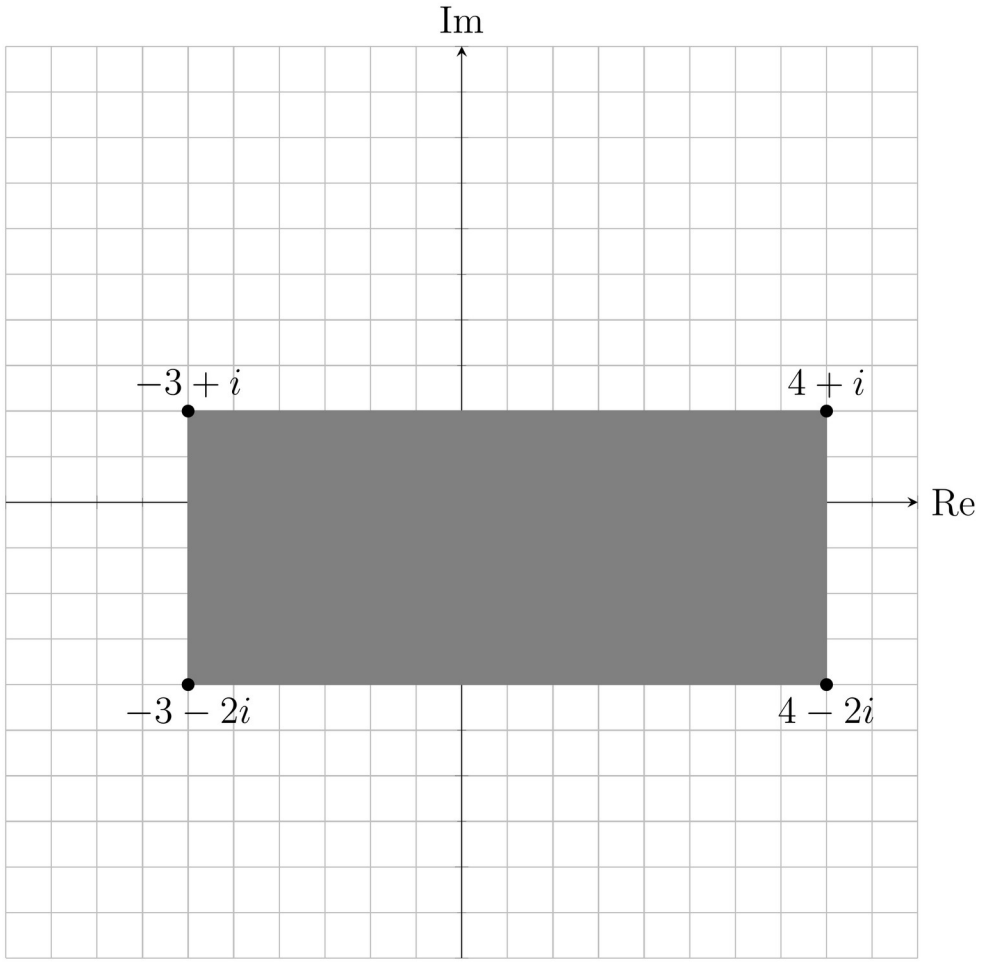
We can represent a rectangular region of the complex plain using two intervals: one for the range of real values, and another for the range of imaginary values.

We will use the notation `[min_r, max_r]` to refer to the range of real values, and `[min_i, max_i]` for the range of imaginary values.

The entire region can be described by: `([min_r, max_r], [min_i, max_i])`.

Mandelbrot Coordinates

Consider the region given by: `([-3, 4], [-2, 1])`.



Terminal Coordinates

Terminal Window:

The terminal has a number of rows and columns.

The row position represents the vertical distance from the top of the screen.

The column position represents the horizontal distance from the left side of the screen.

Translation:

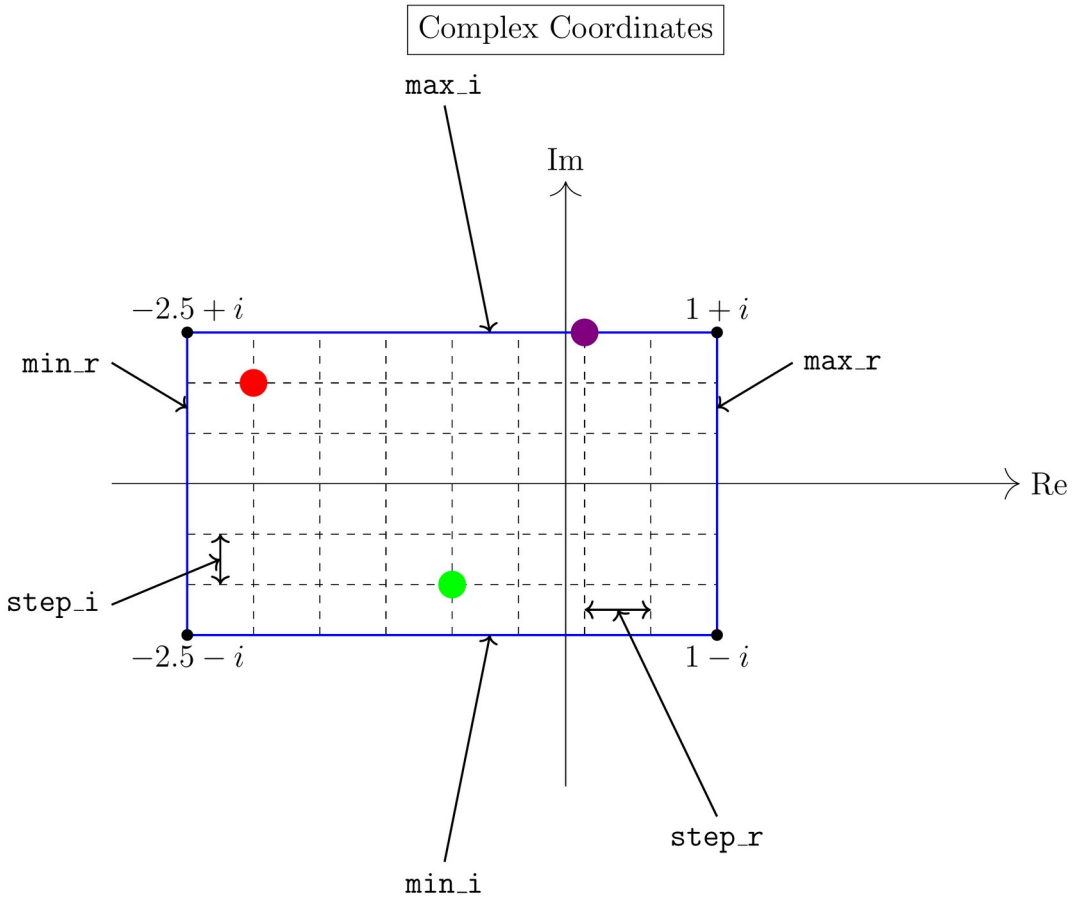
Each cell is represented by the point at its top left corner.

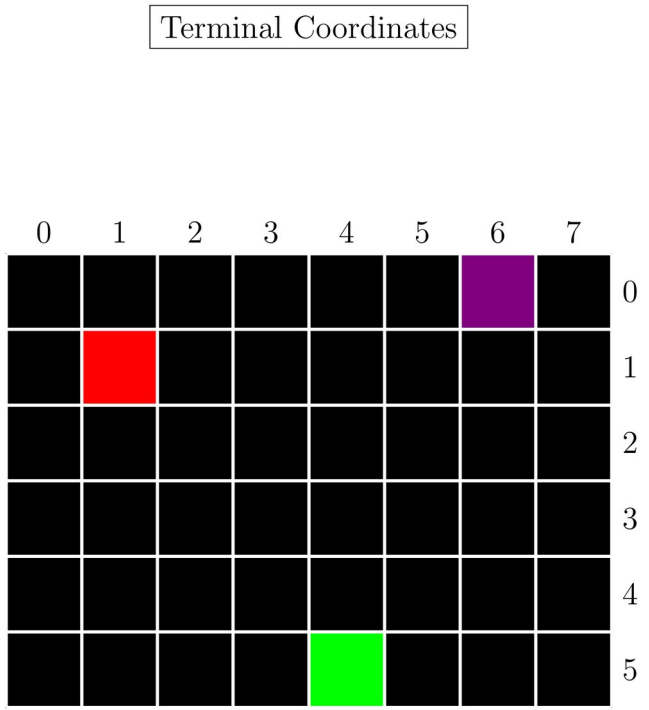
Terminal cells are mapped directly to the complex plain.

The function getStep computes the step size used in this mapping.

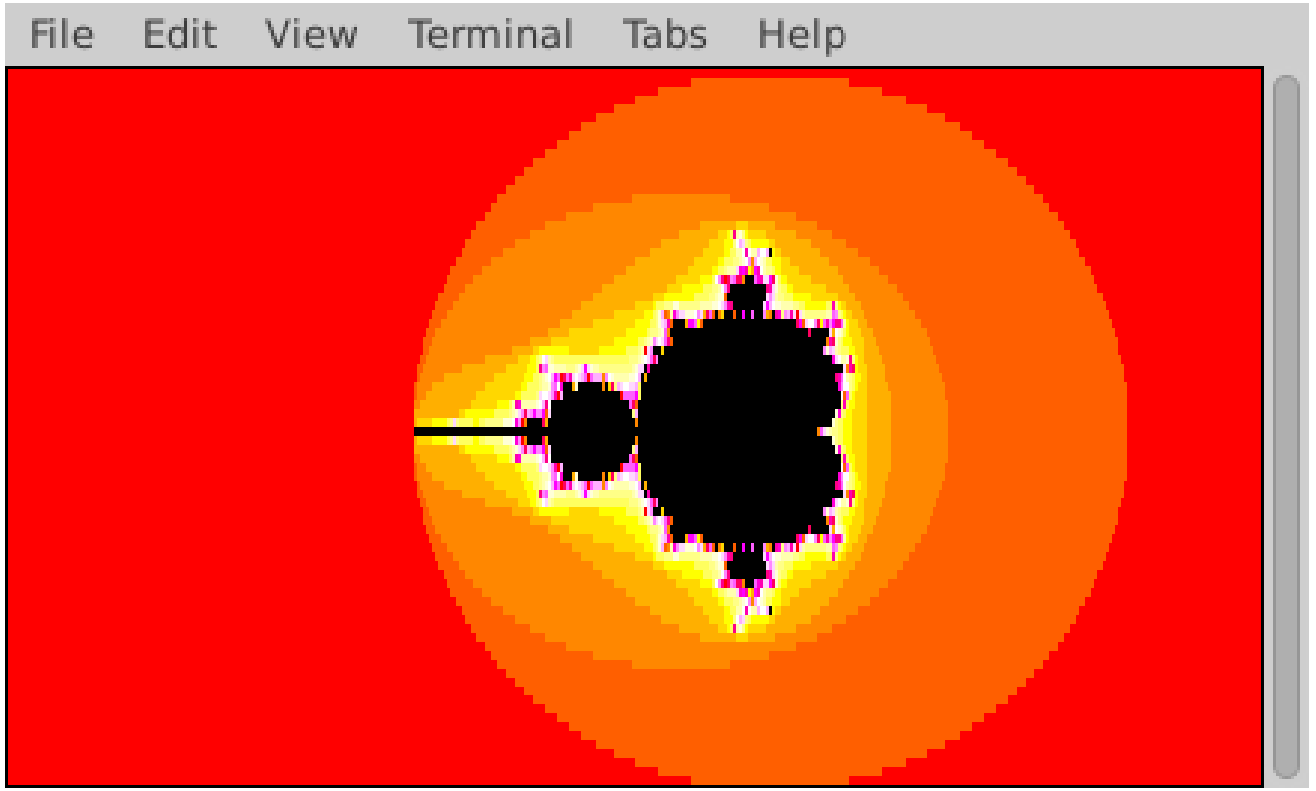
Example Mapping

Consider the mapping between the region `([-2.5, 1], [-1,1])` to a 6 by 8 terminal:





Changing The Region



Symbols & Colour

Colour Palette:

GLIR uses ANSI escape codes, which modify properties of the text in the terminal.

GLIR abstracts away the details of ANSI escape codes, and allows you to select colours from a lookup table:

The colours used for the fractal are stored at the `palette` variable. Points that don't escape are coloured using the value of the `in_color` variable.

Symbols

Characters in the Terminal:

GLIR can print strings to the terminal.

The characters used in this assigment are stored starting at the `symbols` label.

The number of symbols is in `symbol_size`.

Rendering

Using Colours and Symbols to Render the Mandelbrot Set:

In this lab, you will use both colours and symbols for rendering.

There are a finite number of colours and symbols, but infinite possible escape times.

Thus we find `escape_time` mod `palette_size` and use that to index into the tables.

This causes the colours and symbols to repeat in sequence.

Floating Point

RISC-V Support For Floating Point:

Floating point is supported in RISC-V through the “F”, “D”, and “Q” extensions.

In this lab we are concerned with the “D” extension, which adds support for 64-bit floating point (double precision).

We use double precision floats rather than single precision to allow us to render the fractal in greater detail, and to reduce numerical issues.

Floating point extension:

The “D” extension adds 32 additional floating point registers, each of which can hold a 64-bit float.

The extension adds instructions for floating point conversion, arithmatic, control, and data movement.

We will use `rd`, `rs1`, and `rs2` as the integer destination and source registers.

We will use `fd`, `fs1`, and `fs2` as the floating point destination and source registers.

Conversion Instructions

`fcvt.d.w fd, rs1`	Convert the value in `rs1` to a 64-bit float and store in `fd`.
`fcvt.w.d rd, fs1`	Convert the value in `fs1` to 32-bit signed integer and store the result in `rd`.

Arithmetic Instructions

`fadd.d fd, fs1, fs2`	Sum `fs1` and `fs2`, then store the result in `fd`.
`fsub.d fd, fs1, fs2`	Subtract `fs1` and `fs2`, then store the result in `fd`.
`fmul.d fd, fs1, fs2`	Multiply `fs1` and `fs2`, then store the result in `fd`.
`fdiv.d fd, fs1, fs2`	Divide `fs1` by `fs2`, then store the result in `fd`.

Control Instructions

We cannot branch directly on floating point conditions. Instead the following instructions store a `1` or `0` depending on the condition.

`feq.d rd, fs1, fs2`	Set `rd` to `1` if `fs1 = fs2`, set to `0` otherwise.
`flt.d rd, fs1, fs2`	Set `rd` to `1` if `fs1 < fs2`, set to `0` otherwise.
`fge.d rd, fs1, fs2`	Set `rd` to `1` if `fs1 ≥ fs2`, set to `0` otherwise.

Data Movement Instructions

`fmv.d fd, fs1`	Copy the value in `fs1` to `fd`.
`fld fd, X(rs1)`	Load into `fd` the value at `rs1 + X`.
`fsd fd, X(rs1)`	Store into memory at `rs1 + X` the value in `fd`.

Calling Conventions

RISC-V Floating Point Calling Convetions:

The RISC-V floating point registers are logically partitioned into temporary, saved, and argument groups.

These groups function in the same way as the integer registers.

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Example Floating Point Code

We have provided some sample programs in the `Code/Examples` directory.

CMPUT 229

Assignment

Overview of the lab

Overview:

In this lab, you must implement a series of functions to render the Mandelbrot fractal.

You are required to implement all of the following functions.

renderFractal

Description:

This function renders a portion of the the Mandelbrot fractal.

Parameters:

`a0`: Number of rows.

`a1`: Number of columns.

`a2`: Maximum iterations.

`fa0`: `max_i`

`fa1`: `min_i`

`fa2`: `max_r`

`fa3`: `min_r`

Return Value:

N/A

calculateEscape

Description:

Calculate the escape time up to the given maximum for a complex number `c`.

Parameters:

`a0`: Maximum number of iterations.

`fa0`: Real part of `c`.

`fa1`: Imaginary part of `c`.

Return Values:

`a0`: `1` if c escaped before max iterations, `0` otherwise.

`a1`: Number of iterations before escape. If the maximum number of iterations was reached, this should be the maximum number of iterations.

getStep

Description:

Compute the real and immaginary step sizes.

Parameters:

`a0`: Number of rows.

`a1`: Number of columns.

`fa0`: `max_i`

`fa1`: `min_i`

`fa2`: `max_r`

`fa3`: `min_r`

Return Value:

`fa0`: Real step size.

`fa1`: Immaginary step size.

CMPUT 229

Testing

Testing your Solution

Included tests:

We have provided some test inputs and outputs.

Tests are stored in the “Tests” directory. `*.txt` files correspond to the inputs, while `*.out` files are the expected outputs.

Format of the test file:

Make sure the test file has the correct format (refer to the website).

RARS may need the full path to the test file.