Rust Programming Assignment: Mandelbrot Set Visualization

University of Rust In Practice Randoms

Objective

The goal of this exercise is to compute and render the Mandelbrot set using Rust. Through this project, you will:

- Use complex numbers with the `num` crate.
- Understand the Mandelbrot escape-time algorithm.
- Map image space (pixels) to the complex plane.
- Build and render an ASCII-based visualization.

Setup

1. Create a new Rust project:

```
cargo new mandelbrot_rust
```

2. Add the dependency in `Cargo.toml`:

```
[dependencies]
num = "0.4"
```

3. Import the `Complex` type at the top of your `main.rs`:

```
use num::complex::Complex;
```

Part 1 — Complex Number Practice

1. Create two complex numbers and try operations like addition and multiplication.
2. Use `norm()` to get the distance from the origin.
3. Answer: What happens to the size of a complex number when you repeatedly square it?

Part 2 — The Mandelbrot Escape Rule

1. Write a function that accepts a point `(cx, cy)` and a maximum iteration count.
2. Start with $z = 0$ and iterate $z = z^2 + c$ until `norm(z) > 2`.
3. Return the number of iterations it took to escape, or the maximum if it didn't escape.
4. Test with different values of (cx, cy): some will escape quickly, others won't.

Part 3 — Map Pixels to Complex Coordinates

1. Choose output dimensions (e.g., 100x24 characters).
2. Write code that maps each pixel (x, y) to a complex point (cx, cy).
3. Use linear interpolation between bounds like:

$$\text{Real: } [-2.0, 1.0], \quad \text{Imaginary: } [-1.0, 1.0]$$

4. Confirm that the corners of your image map correctly to these values.

Part 4 — Build the Escape-Time Grid

1. Iterate through every pixel in your output space.
2. For each pixel, convert it to (cx, cy) and compute its escape time.
3. Store results in a 2D `Vec<Vec<usize>>` grid.

Part 5 — Render the Fractal

1. Write a function to convert escape-time values to ASCII characters:

```
match value {
    0..=2 => ' ',
    3..=5 => '.',
    6..=10 => '*',
    11..=30 => '+',
    31..=100 => '#',
    _ => '%'
}
```

2. Print each row line-by-line to display the fractal.
3. Tweak ranges and characters for better contrast.

Testing and Tuning

Try the following configuration:

- `max_iters = 1000`
- `x_min = -2.0`, `x_max = 1.0`
- `y_min = -1.0`, `y_max = 1.0`
- `width = 100`, `height = 24`

Bonus Challenges (Optional)

- Add zooming via CLI arguments or user input.
- Add ANSI color output.
- Render as an image instead of ASCII.

Deliverables

- Your complete `main.rs` file.
- A screenshot of your fractal in the terminal.
- A short paragraph explaining what the Mandelbrot set represents and what you learned.