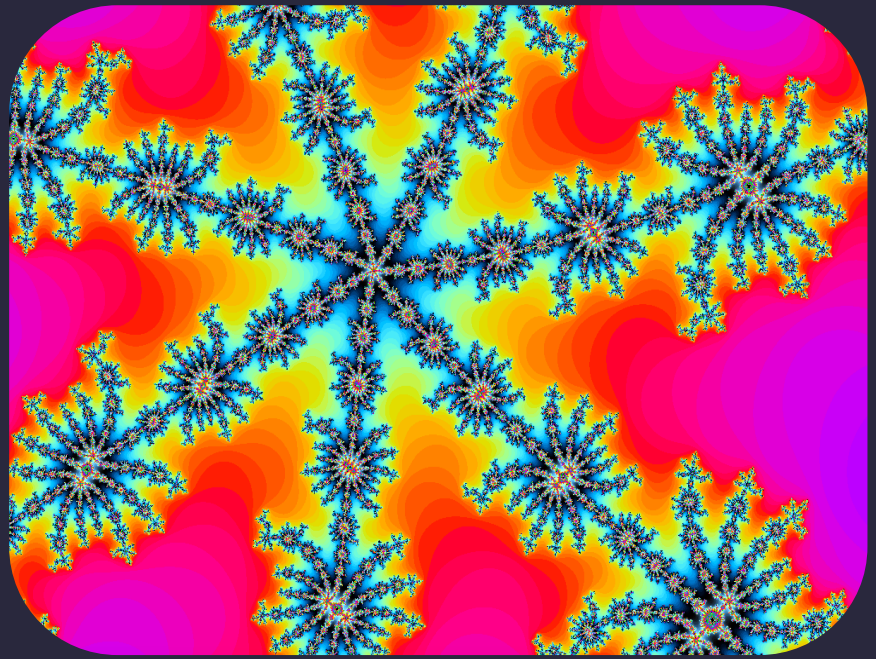
FRACT’OL

by Saskia Mischnick



What is a fractal?

A fractal is an abstract mathematical object, like a curve or a surface, which pattern remains the same at every scale.

No matter how far we zoom in, the pattern remains the same / repeats itself!

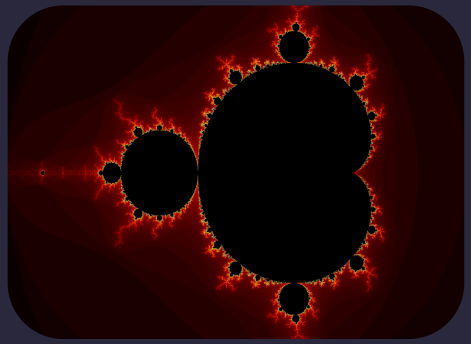
Fractals in nature

ROMANESCO

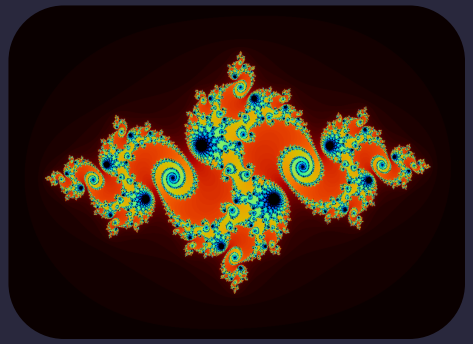
NAUTILUS SHELL

SUCCULENTS

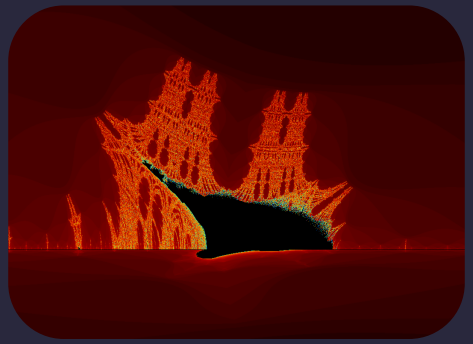
Mathematical fractals



MANDELBROT



JULIA

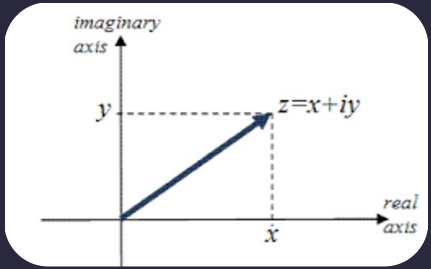


BURNING SHIP

Mathematical concepts behind fractals

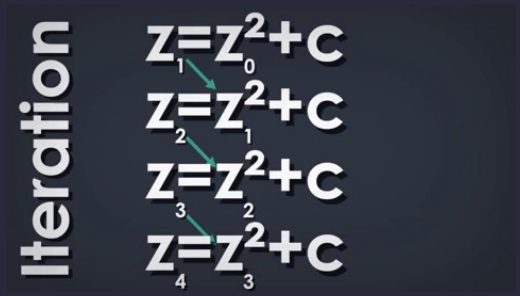
THE COMPLEX PLANE

- Displays complex numbers on a plane / coordinate system
- COMPLEX NUMBERS:
  x + y*i
  x = real number
  y*i = imaginary number (i = $\sqrt{-1}$)



RECURSION

- repeated application of a procedure or formula
- EXAMPLE: Mandelbrot formula
  $z_n = z_{n-1}^2 + c$
  $z_0 = 0$
  c = point on the complex plane



Requirements of the fract'ol project

DIFFERENT FRACTALS

Display at least the Mandelbrot and the Julia set

ZOOM

Allow to zoom in and explore the details of the fractal

DIFFERENT COLORS

Offer different color-schemes to show the depth of the fractal

Challenge: Calculation of every pixel

```c
/**
 * First clears the image. Then iterates through each pixel of the image and
 * determines if it lies within the fractal set or outside of it by calling the
 * fractals rendering function. In case it lies outside, calls the respective
 * function to color the pixel. In the end, the image is pushed to the window.
 * @param vars [t_vars *] Pointer to the struct containing important variables.
 */
int render(t_vars *vars)
{
    t_point p;

    ft_bzero(vars->img.addr, WIDTH * HEIGHT * sizeof(int));
    p.x = 0;
    p.y = 0;
    while (p.y < HEIGHT)
    {
        p.x = 0;
        while (p.x < WIDTH)
        {
            vars->f_render(&p, vars);
            if (vars->z.res >= 4)
                vars->f_col(&vars->img, p.x, p.y, vars->z);
            p.x++;
        }
        p.y++;
    }
    mlx_put_image_to_window(vars->mlx_ptr, vars->win_ptr, vars->img.img, 0, 0);
    return (0);
}
```

Starting with Pixel (0/0)

Loop iterating through all rows

Loop iterating through all columns

Function to carry out the calculation for every pixel

Function to color every pixel according to outcome

Challenge: Calculation of every pixel

```c
/**
 * Applies the mandelbrot formula to a single pixel, determining its
 * coordinates on the complex plane and respectively checking whether
 * it lies within the mandelbrot-set or outside of it and how many
 * iterations are necessary to break out of the mandelbrot-area.
 * @param p [t_point *] Pointer to variables describing a pixel on the display.
 * @param vars [t_vars *] Pointer to the struct containing important variables.
 */
void    render_mandelbrot(t_point *p, t_vars *vars)
{
    t_point tmp;

    vars->z.n = 0;
    vars->z.r = 0;
    vars->z.i = 0;
    vars->z.res = 0;
    get_r_and_i(p, vars);
    if (optimize_mandelbrot(p) == 0)
        return ;
    while (vars->z.n < 255 && vars->z.res < 4)
    {
        tmp.r = vars->z.r;
        tmp.i = vars->z.i;
        vars->z.r = (tmp.r * tmp.r) - (tmp.i * tmp.i) + p->r;
        vars->z.i = (2 * tmp.r * tmp.i) + p->i;
        vars->z.n++;
        vars->z.res = check_z(vars);
    }
}
```

Starting with $z_0 = 0$

Getting the real and imaginary value of the pixel

While loop recursively calculating for that pixel until

- the result is bigger than 4
- 255 iterations have been reached

Challenge: Handling events

```c
/**
 * Handles right and left key events and carries out the respective actions to
 * move the view.
 * @param key [int] Defines the key that has been pressed.
 * @param vars [t_vars *] Pointer to the struct containing important variables.
*/
void    key_right_left(int key, t_vars *vars)
{
    double  x;

    x = (vars->x_max - vars->x_min) * 0.05;
    if (key == KEY_RIGHT)
    {
        vars->x_zero -= (0.05 * WIDTH);
        vars->x_max += x;
        vars->x_min += x;
    }
    else if (key == KEY_LEFT)
    {
        vars->x_zero += (0.05 * WIDTH);
        vars->x_max -= x;
        vars->x_min -= x;
    }
}
```

Determine value by which to shift (5% of whole axis)

Shifting the origin of the plane

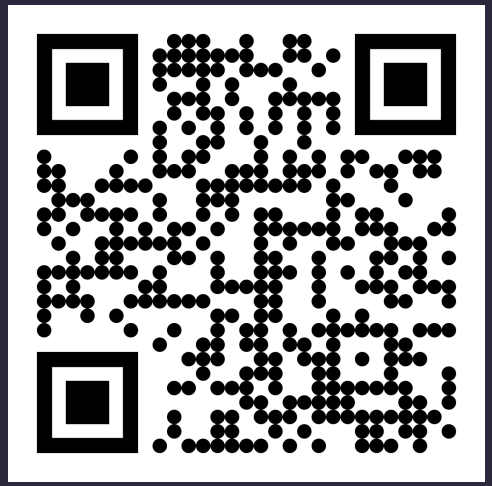
Adjust the scale values of my image section

<LET’S EXPLORE!>

 1,000 lines of code

 ca. 70 hours of work

TEST IT YOURSELF:



You can find my fract‘ol and an explanation how to install and use it on my github profile