#include "fractol.h"

int set\_colour\_julia(double c\_real, double c\_imag, double z\_real, double z\_imag)

{

int k;

double temp\_complex;

k = -1;

temp\_complex = 0;

while (++k < ITERATIONS)

{

temp\_complex = z\_real \* z\_real - z\_imag \* z\_imag + c\_real;

z\_imag = 2 \* z\_real \* z\_imag + c\_imag;

z\_real = temp\_complex;

if (MODULUS(z\_real, z\_imag) > 4)

return (k);

}

return (0);

}

void juliaset(t\_fractal \*julia)

{

int i\_window;

int j\_window;

int iterations;

double z\_real;

double z\_imag;

j\_window = -1;

while (++j\_window < (WIDTH / RATIO))

{

i\_window = -1;

iterations = 0;

while (++i\_window < WIDTH)

{

z\_real = julia->zoom \* (i\_window - WIDTH / 2 ) \* CARTESIAN\_DIST / (2 \* WIDTH) + julia->zoom\_real;

z\_imag = julia->zoom \* (- j\_window + WIDTH / (2 \* RATIO) ) / (2 \* WIDTH) \* CARTESIAN\_DIST - julia->zoom\_imag;

iterations = set\_colour\_julia(julia->c\_real, julia->c\_imag, z\_real, z\_imag);

if (iterations == 0)

my\_mlx\_pixel\_put(julia, i\_window, j\_window, 0x000000);

else

my\_mlx\_pixel\_put(julia, i\_window, j\_window, iterations \* julia->colour \* (MAXCOLOURS / ITERATIONS));

}

}

}

int set\_colour\_mandelbrot(double c\_real, double c\_imag)

{

int k;

double t\_real;

double t\_imag;

double temp\_complex;

k = -1;

temp\_complex = 0;

t\_real = c\_real;

t\_imag = c\_imag;

while (++k < ITERATIONS)

{

temp\_complex = t\_real \* t\_real - t\_imag \* t\_imag + c\_real;

t\_imag = 2 \* t\_real \* t\_imag + c\_imag;

t\_real = temp\_complex;

if (MODULUS(t\_real, t\_imag) > 4)

return (k);

}

return (0);

}

void mandelbrotset(t\_fractal \*mandel)

{

int i\_window;

int j\_window;

int iterations;

double c\_real;

double c\_imag;

j\_window = -1;

while (++j\_window < (WIDTH / RATIO))

{

i\_window = -1;

iterations = 0;

while (++i\_window < WIDTH)

{

c\_real = mandel->zoom \* (i\_window - WIDTH / 2 ) \* CARTESIAN\_DIST / (2 \* WIDTH) + mandel->zoom\_real;

c\_imag = mandel->zoom \* (- j\_window + WIDTH / (2 \* RATIO) ) / (2 \* WIDTH) \* CARTESIAN\_DIST - mandel->zoom\_imag;

iterations = set\_colour\_mandelbrot(c\_real, c\_imag);

if (iterations == 0)

my\_mlx\_pixel\_put(mandel, i\_window, j\_window, 0x000000);

else

my\_mlx\_pixel\_put(mandel, i\_window, j\_window, iterations \* mandel->colour \* (MAXCOLOURS / ITERATIONS));

}

}

}

#include "fractol.h"

void fractalsetup(t\_fractal \*fractal)

{

mlx\_destroy\_image(fractal->mlx, fractal->img);

fractal->img = mlx\_new\_image(fractal->mlx, WIDTH, (WIDTH / RATIO));

fractal->addr = mlx\_get\_data\_addr(fractal->img, &fractal->bits\_per\_pixel,

&fractal->line\_length, &fractal->endian);

if (!ft\_strncmp(fractal->name, "julia", 5))

juliaset(fractal);

else if (!ft\_strncmp(fractal->name, "mandelbrot", 10))

mandelbrotset(fractal);

mlx\_put\_image\_to\_window(fractal->mlx, fractal->win, fractal->img, 0, 0);

}

int my\_mlx\_pixel\_put(t\_fractal \*fractal, int x, int y, int color)

{

char \*dst;

dst = fractal->addr + (y \* fractal->line\_length + x

\* (fractal->bits\_per\_pixel / 8));

\*(unsigned int \*)dst = color;

return (0);

}

void show\_options(void)

{

write(1,"Please enter the fractal that you want: mandelbrot or julia\n", 60);

exit(1);

}

int freeall(t\_fractal \*fractal)

{

mlx\_destroy\_image(fractal->mlx, fractal->img);

mlx\_destroy\_window(fractal->mlx, fractal->win);

mlx\_destroy\_display(fractal->mlx);

free(fractal->mlx);

exit(0);

}

int main(int argc, char \*\*argv)

{

t\_fractal fractal;

if (argc == 2)

{

if (!ft\_strncmp(argv[1], "mandelbrot", 10) && ft\_strlen(argv[1]) == 10)

mandelbrot\_param(&fractal, argv[1]);

else if (!ft\_strncmp(argv[1], "julia", 5) && ft\_strlen(argv[1]) == 5)

julia\_param(&fractal, argv[1]);

else

show\_options();

fractal.mlx = mlx\_init();

fractal.win = mlx\_new\_window(fractal.mlx, WIDTH, (WIDTH / RATIO), fractal.name);

fractal.img = mlx\_new\_image(fractal.mlx, WIDTH, (WIDTH / RATIO));

fractalsetup(&fractal);

mlx\_key\_hook(fractal.win, key\_hook, &fractal);

mlx\_mouse\_hook(fractal.win, mouse\_hook, &fractal);

mlx\_hook(fractal.win, 17, 0, close\_window, &fractal);

mlx\_loop(fractal.mlx);

}

else

show\_options();

return (0);

}

#ifndef FRACTOL\_H

# define FRACTOL\_H

# include "mlx/mlx.h"

# include <math.h>

# include <stdio.h>

# include <stdlib.h>

# include <unistd.h>

# include <math.h>

#define WIDTH 1280

#define RATIO (1.777778)

#define MODULUS(real,imag) (real \* real + imag \* imag)

#define ITERATIONS 250

#define MAXCOLOURS 0xffffff

#define CARTESIAN\_DIST 8

# define ESC 65307

# define LEFT 65361

# define UP 65362

# define RIGHT 65363

# define DOWN 65364

# define R 114

# define Z 122

# define PLUS 65451

# define MINUS 65453

typedef struct s\_fractal {

char \*name;

void \*img;

void \*addr;

int bits\_per\_pixel;

int line\_length;

int endian;

void \*mlx;

void \*win;

double zoom;

double z\_real;

double z\_imag;

double c\_real;

double c\_imag;

int iterations;

int colour;

double carte\_distance;

double zoom\_real;

double zoom\_imag;

} t\_fractal;

void mandelbrot\_param(t\_fractal \*fractal, char \*name);

void julia\_param(t\_fractal \*fractal, char \*name);

void juliaset(t\_fractal \*julia);

void mandelbrotset(t\_fractal \*mandel);

int set\_colour\_mandelbrot(double c\_real, double c\_imag);

int set\_colour\_julia(double c\_real, double c\_imag, double z\_real, double z\_imag);

void fractalsetup(t\_fractal \*fractal);

void show\_options(void);

int my\_mlx\_pixel\_put(t\_fractal \*fractal, int x, int y, int color);

int key\_hook(int keycode, t\_fractal \*fractal);

int mouse\_hook(int key\_code, int x, int y, t\_fractal \*fractal);

int close\_window(t\_fractal \*fractal);

int freeall(t\_fractal \*fractal);

size\_t ft\_strlen(const char \*str);

int ft\_strncmp(const char \*s1, const char \*s2, size\_t n);

#endif

#include "fractol.h"

int close\_window(t\_fractal \*fractal)

{

freeall(fractal);

exit(EXIT\_SUCCESS);

return (EXIT\_SUCCESS);

}

int key\_hook(int keycode, t\_fractal \*fractal)

{

if (keycode == ESC)

close\_window(fractal);

else if (keycode == LEFT)

fractal->zoom\_real = fractal->zoom\_real - 0.24 \* fractal->zoom;

else if (keycode == DOWN)

fractal->zoom\_imag = fractal->zoom\_imag + 0.24 \* fractal->zoom;

else if (keycode == RIGHT)

fractal->zoom\_real = fractal->zoom\_real + 0.24 \* fractal->zoom;

else if (keycode == UP)

fractal->zoom\_imag = fractal->zoom\_imag - 0.24 \* fractal->zoom;

/\*

else if (keycode == PLUS)

fractal->colour += 100;

else if (keycode == MINUS)

fractal->colour -= 100;

else if (keycode == R)

{

write(1, "Reset time!\n", 12);

julia\_param(fractal, fractal->name);

}

\*/

else if (keycode == Z)

fractal->zoom = 1;

if (keycode != ESC)

fractalsetup(fractal);

return (0);

}

int mouse\_hook(int key\_code, int x, int y, t\_fractal \*fractal)

{

if (key\_code == 1 && !ft\_strncmp(fractal->name, "julia", 5))

{

fractal->c\_real = ((double) x / WIDTH \* CARTESIAN\_DIST - (CARTESIAN\_DIST/2));

fractal->c\_imag = ((double) y / (WIDTH / RATIO) \* (CARTESIAN\_DIST/RATIO) - (CARTESIAN\_DIST/(RATIO\*2)));

fractal->zoom = 1;

}

else if (key\_code == 5)

fractal->zoom \*= 1.42;

else if (key\_code == 4)

fractal->zoom /= 1.42;

fractalsetup(fractal);

return (0);

}

NAME = fractol

CC = cc

CFLAGS = -Wall -Werror -Wextra -g

MLXFLAGS = -L ./mlx -lmlx -Ilmlx -lXext -lX11

SRC = fractol.c \

fractals.c \

hooks.c \

params.c \

all: $(NAME)

$(NAME): $(SRC:.c=.o) fractol.h

$(CC) $(CFLAGS) -lm $(SRC) $(MLXFLAGS) -o $(NAME)

clean:

rm -rf $(SRC:.c=.o)

fclean: clean

rm -rf $(NAME)

re: fclean all

.PHONY: all, clean, fclean, re, bonus

#include "fractol.h"

void mandelbrot\_param(t\_fractal \*fractal, char \*name)

{

fractal->zoom = 1;

fractal->colour = 0xff00ff;

fractal->name = name;

fractal->zoom\_real = -0.75;

fractal->zoom\_imag = 0;

}

void julia\_param(t\_fractal \*fractal, char \*name)

{

fractal->zoom = 1;

fractal->colour = 0xffff00;

fractal->name = name;

fractal->c\_real = 0;

fractal->c\_imag = 0;

fractal->zoom\_real = 0;

fractal->zoom\_imag = 0;

}

size\_t ft\_strlen(const char \*str)

{

size\_t count;

count = 0;

while (str[count] != '\0')

count++;

return (count);

}

int ft\_strncmp(const char \*s1, const char \*s2, size\_t n)

{

size\_t count;

count = 0;

if (n == 0)

return (0);

while (s1[count] == s2[count] && count < n - 1 && s1[count] && s2[count])

count++;

return ((unsigned char)s1[count] - (unsigned char)s2[count]);

}