6620. Organización de Computadoras
Trabajo Práctico 0:
Infraestructura Básica

Riesgo, Daniela, *Padrón Nro. 95557*
danielap.riesgo@gmail.com

Martin, Débora, *Padrón Nro. 90934*
debbie1mes.world@gmail.com

Constantino, Guillermo, *Padrón Nro. 89776*
guilleconstantino@gmail.com

2do. Cuatrimestre de 2014
66.20 Organización de Computadoras – Prática Martes
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

El presente trabajo tiene como objetivo familiarizarse con las herramientas de software que serán usadas en los siguientes trabajos, implementando un programa en lenguaje C, y su correspondiente documentación, que resuelva el problema planteado, que permita dibujar el conjunto de Mandelbrot y sus vecindades

1. Introducción

Al comenzar a utilizar nuevas herramientas, en cualquier ámbito, es necesaria una breve introducción al funcionamiento de las mismas: tener una noción de las prestaciones que ofrecen, asi también como de sus limitaciones.
Como primer objetivo en la materia, nos proponemos adentrarnos en el funcionamiento del emulador GXemul. Nuestra meta será emular una plataforma MIPS (ejecutando un sistema operativo NetBSD), para poder desde allí desarrollar programas en lenguaje C. Estos serán compilados y ejecutados haciendo uso de la herramienta GCC (GNU Compiler Collection), mediante el cual también será posible obtener, a posteriori, el código MIPS32 del programa.
Una vez cumplido este objetivo, aprenderemos los rudimientos de LaTeX para generar la documentación relevante al trabajo práctico.

2. Programa a implementar

Se trata de un disenar un programa que permita dibujar el conjunto de Mandelbrot y sus vecindades, en lenguaje C. El mismo recibirá por línea de comando, una serie de parámetros describiendo la región del plano complejo y las características del archivo imagen a generar. No deberá interactuar con el usuario, ya que no se trata de un programa interactivo, sino más bien de una herramienta de procesamiento batch. Al finalizar la ejecución, y volver al sistema operativo, el programa habrá dibujado el fractal en el archivo de salida. El formato gráfico a usar es PGM o portable gray map, un formato simple para describir imágenes a digitales monocromáticas.

3. Explicación de la Implementación

Las principales funciones y estructuras en nuestra implementación son las siguientes:

3.1. main.c

3.1.1. main

Esta función se encarga de procesar las opciones ingresadas por línea de comando e invocar a las funciones según corresponda.

3.1.2. generatePGM

Genera la imágen PGM a partir de los valores de escape, que son insertados en la matriz que utiliza la función writePGM para generar dicha imágen.

3.2. pgm.h

Contiene la estructura básica para generar una imagen del tipo PGM.

3.2.1. writePGM

Toma como parámetros el nombre de archivo, y una estructura PGM, la cual contiene: cantidad de filas y columnas, color de gris máximo, y la matriz de pixeles.

3.3. velocidad_escape.h

Este archivo maneja la estructura de los pixeles y operaciones entre números complejos.

3.3.1. velocidad_de_escape

Devuelve velocidad de escape según el pixel y complejo que represente. Este valor define una intensidad según la condición de corte.

4. Generación de ejecutables y código assembly

Para generar el ejecutable del programa, debe correrse la siguiente sentencia en una terminal:

```
$ gcc -Wall -pedantic --std=c99 -c velocidad_escape.c
$ gcc -Wall -pedantic --std=c99 -c pgm.c
$ gcc -Wall -pedantic --std=c99 velocidad_escape.o pgm.o main.c -o tp0
```

Para generar el código MIPS32, debe ejecutarse lo siguiente:

```
$ gcc -Wall -S -pedantic --std=c99 -c velocidad_escape.c
$ gcc -Wall -S -pedantic --std=c99 -c pgm.c
$ gcc -Wall -S -pedantic --std=c99 velocidad_escape.o pgm.o main.c -o tp0
```

Nótese que para ambos casos se han activado todos los mensajes de 'Warning' (-Wall). Además, para el caso de MIPS, se ha habilitado '-S', que detiene al compilador luego de generar el assembly.

5. Corridas de prueba

En esta sección se presentan algunas de las distintas corridas que se realizaron para probar el funcionamiento del trabajo práctico.

1. Generamos una imagen de 1 punto de lado, centrada en el orígen del plano complejo:

```
> ./tp0 -c 0+0i -r 1x1 -o -
P2
1
1
255
255
```

Notar que el resultado es correcto, ya que este punto pertenece al conjunto de Mandelbrot.

2. Repetimos el experimento, pero nos centramos ahora en un punto que seguro no pertenece al conjunto:

```
> ./tp0 -c 10+0i -r 1x1 -o -
P2
1
1
255
0
```

Notar que el resultado es correcto, ya que este punto pertenece al conjunto de Mandelbrot.
3. Imagen imposible:

```
> ./tp0 -c 0+0i -r 0x1 -o -
Usage:
  tp0 -h
  tp0 -V
: Undefined error: 0
```

4. Archivo de salida imposible:

```
> ./tp0 -o /tmp
fatal: cannot open output file: is a directory
```

5. Coordenadas complejas imposibles:

```
> ./tp0 -c 1+3 -o -
fatal: invalid center specification: Undefined error: 0
```

6. Argumentos de línea de comando vacíos,

```
> ./tp0 -c "" -o -
fatal: invalid center specification: Undefined error: 0
```

7. Imagen PGM

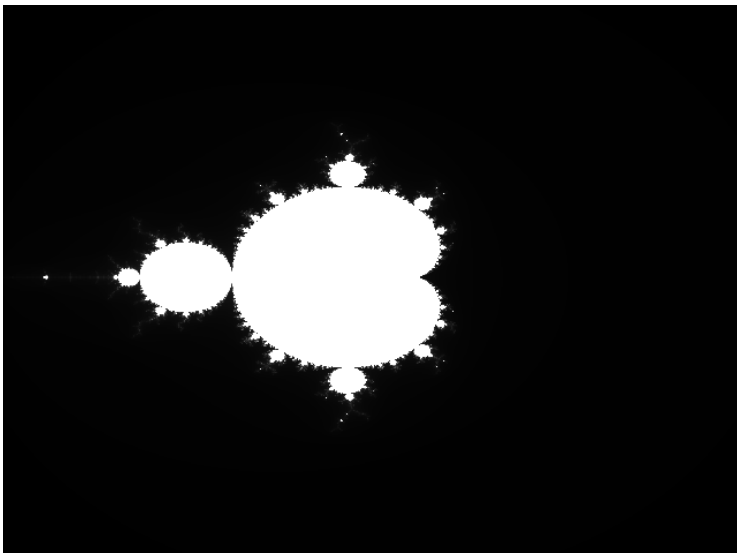

Figura 1:

```
> ./tp0 -o uno.pgm
```

Genera la siguiente imagen:

8. Imagen PGM con región no centrada y un rectángulo de 0,005 unidades de lado.

```
> ./tp0 -c +0.282-0.01i -w 0.005 -H 0.005 -o dos.pgm
```

Genera la siguiente imagen:

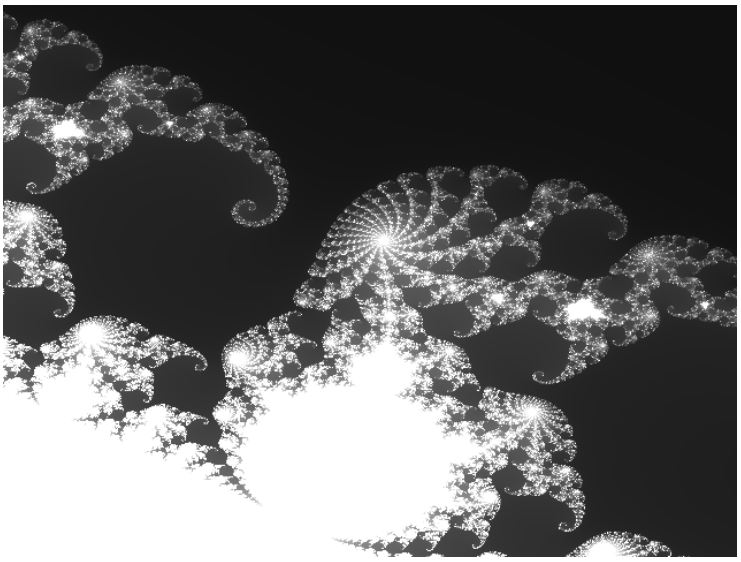

Figura 2:

6. Código fuente C

6.1. main.c

```
#include <stdlib.h>
#include <getopt.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

#include "pgm.h"
#include "velocidad_escape.h"

#define DEFAULT_RESOLUTION_WIDTH 640
#define DEFAULT_RESOLUTION_HEIGHT 480
#define DEFAULT_CENTER_REAL 0
#define DEFAULT_CENTER_IMAG 0
#define DEFAULT_PLANE_WIDTH 4
#define DEFAULT_PLANE_HEIGHT 4
#define ARG_DEFAULT_OUT "-"


typedef struct _OutputData{
    int resolution[2];
    float center[2];
    float plane[2];
    FILE* output;
} OutputData;
```

```
void generatePGM(OutputData* data){
    PGMData pgm_image;
    pgm_image.row = data->resolution[1];
    pgm_image.col = data->resolution[0];
    pgm_image.max_gray = 255;
    pgm_image.matrix = allocate_dynamic_matrix(pgm_image.row, pgm_image.col);

    double first_real_value = data->center[0] - data->plane[0]/2;
    double first_imaginary_value = data->center[1] + data->plane[1]/2;
    double width_scale = (data->plane[0] / data->resolution[0]);
    double height_scale =  - (data->plane[1] / data->resolution[1]);
    first_real_value += width_scale/2;
    first_imaginary_value += height_scale/2;

    for(int i = 0; i < pgm_image.row; i++){
        for(int j = 0; j < pgm_image.col; j++){
            pixel_t* pixel = crear_pixel(first_real_value + j * width_scale, first_imagina
            pgm_image.matrix[i][j] = velocidad_de_escape(pixel);
            destruir_pixel(pixel);
        }
    }

    writePGM(data->output, &pgm_image);
    deallocate_dynamic_matrix(pgm_image.matrix, pgm_image.row);
}

void OutputDataInitialize(OutputData* data){
    data->resolution[0] = DEFAULT_RESOLUTION_WIDTH;
    data->resolution[1] = DEFAULT_RESOLUTION_HEIGHT;
    data->center[0] = DEFAULT_CENTER_REAL;
    data->center[1] =  DEFAULT_CENTER_IMAG;
    data->plane[0] = DEFAULT_PLANE_WIDTH;
    data->plane[1] = DEFAULT_PLANE_HEIGHT;
    data->output = stdout;
}

int terminateError(char* errorMessaje){
    perror(errorMessaje);
    return 1;
}

int main(int argc, char* argv[]){
    static struct option long_options[] =
    {
        {"resolution", required_argument, 0, 'r'},
        {"center", required_argument, 0, 'c'},
        {"width", required_argument, 0, 'w'},
        {"height", required_argument, 0, 'H'},
        {"output", required_argument, 0, 'o'},
```

```
    {0, 0, 0, 0}
};

OutputData data;
OutputDataInitialize(&data);
bool need_close = false;
char option, i;
int option_index;

while ((option = getopt_long(argc, argv, "o:r:c:w:H:", long_options, &option_index)) !
    switch (option){
        case 'r':
            if (sscanf(optarg, "%d%*c%d", &data.resolution[0], &data.resolution[1]) !=
                return terminateError("fatal: invalid resolution specification");
            }
            if (data.resolution[0] == 0 || data.resolution[1] == 0){
                return terminateError("Usage:\n\ttp0 -h\n\ttp0 -V\n");
            }
            break;
        case 'c':
            if (sscanf(optarg, "%f%*c%f%c", &data.center[0], &data.center[1], &i) != 3
                return terminateError("fatal: invalid center specification");
            }
            break;
        case 'w':
            data.plane[0] = atof(optarg);
            if (data.plane[0] == 0){
                return terminateError("fatal: invalid width specification");
            }
            break;
        case 'H':
            data.plane[1] = atof(optarg);
            if (data.plane[1] == 0){
                return terminateError("fatal: invalid height specification");
            }
            break;
        case 'o':
            if (strcmp(ARG_DEFAULT_OUT, optarg) != 0){
                data.output = fopen(optarg, "w");
                if (! data.output){
                    return terminateError("fatal: cannot open output file");
                }
                need_close = true;
            }
            break;
        default:
            return terminateError("fatal: invalid arguments");
    }
}
generatePGM(&data);
```

```
    if (need_close) fclose(data.output);
    return 0;
}
```

6.2. pgm.c

```
#include "pgm.h"

#define HI(num) (((num) & 0x0000FF00) >> 8)
#define LO(num) ((num) & 0x000000FF)

int **allocate_dynamic_matrix(int row, int col) {

int **ret_val;
int i;

ret_val = (int **) malloc(sizeof(int *) * row);
if (ret_val == NULL) {
perror("memory allocation failure");
exit(EXIT_FAILURE);
}

for (i = 0; i < row; ++i) {
ret_val[i] = (int *) malloc(sizeof(int) * col);
if (ret_val[i] == NULL) {
perror("memory allocation failure");
exit(EXIT_FAILURE);
}
}

return ret_val;
}

void deallocate_dynamic_matrix(int **matrix, int row) {

int i;

for (i = 0; i < row; ++i)
free(matrix[i]);
free(matrix);
}

void SkipComments(FILE *fp) {

int ch;
char line[100];

while ((ch = fgetc(fp)) != EOF && isspace(ch))
;
if (ch == '#') {
fgets(line, sizeof(line), fp);
SkipComments(fp);
} else
fseek(fp, -1, SEEK_CUR);
```

```
}

void writePGM(FILE *pgmFile, const PGMData *data) {
int i, j;
int lo;

fprintf(pgmFile, "P2\n");
fprintf(pgmFile, "%d\n%d\n", data->col, data->row);
fprintf(pgmFile, "%d\n", data->max_gray);

for (i = 0; i < data->row; ++i){
for (j = 0; j < data->col; ++j) {
            fprintf(pgmFile, "%d ", data->matrix[data->row - i-1][j]);
}
fprintf(pgmFile, "\n");
}
}
```

6.3. velocidad_escape.c

```
#include <stdlib.h>
#include <stddef.h>
#include <math.h>
#include <complex.h>

#include "velocidad_escape.h"

#define N 256

typedef struct num_complejo {
double real;
double imaginario;
} complejo;

struct pixel {
complejo* numero_complejo;
};

complejo* crear_complejo(double real, double imaginaria) {
complejo* num = malloc(sizeof(complejo));
if (num == NULL)
return NULL;
num->real = real;
num->imaginario = imaginaria;
return num;
}

void destruir_complejo(complejo* num) {
free(num);
}

pixel_t* crear_pixel(double real, double imaginaria) {
pixel_t* pixel = malloc(sizeof(pixel));
if (pixel == NULL)
return NULL;
complejo* num = crear_complejo(real, imaginaria);
if (num == NULL) {
free(pixel);
return NULL;
}
pixel->numero_complejo = num;
return pixel;
}

void destruir_pixel(pixel_t* pixel) {
destruir_complejo(pixel->numero_complejo);
free(pixel);
}
```

```
unsigned int modulo_al_2(complejo* num) {
return (num->real * num->real) + (num->imaginario * num->imaginario);
}

complejo* sumar(complejo* num1, complejo* num2) {
double real = num1->real + num2->real;
double imaginario = num1->imaginario + num2->imaginario;
return crear_complejo(real, imaginario);
}

complejo* al_2(complejo* num) {
double real = num->real * num->real - num->imaginario * num->imaginario;
double imaginario = 2 * num->real * num->imaginario;
return crear_complejo(real, imaginario);
}

int velocidad_de_escape(pixel_t* pixel) {
complejo* num = pixel->numero_complejo;
complejo* num2 = crear_complejo(num->real, num->imaginario);
int i;
for (i = 0; i < (N - 1); ++i) {
if (modulo_al_2(num2) > 4)
break;
num2 = sumar(al_2(num2), num);
}
return i;
}
```

7. Conclusiones

El trabajo práctivo motivo de este informe ha presentado al equipo diversos desafíos. En primer lugar cabe mencionar la adaptación a un ambiente basado en GNU/Linux, no dominado por todos sus integrantes en igual manera. También es destacable la dificultad inicial que trajo la correcta configuración del ambiente virtual utilizado para emular la máquina MIPS. Finalmente, también fue invertida una cantidad considerable de tiempo en aprender a utilizar e investigar sobre LaTeX, ya que si bien permite obtener muy buenos resultados, se requiere de mucha lectura para poder aprovechar todo su potencial. Por las razones expuestas, consideramos muy necesario un trabajo práctico introductorio de esta naturaleza,para nivelar e introducir los elementos a utilizar en las demás actividades prácticas que se desarrollarán en el curso.
Como conclusión final, podemos considerar que hemos logrado obtener un buen manejo de las herramientas introducidas en este primer proyecto.

Referencias

[1] GXemul, http://gxemul.sourceforge.net/

[2] The NetBSD Project, http://www.netbsd.org/

[3] Mandelbrot, http://en.wikipedia.org/wiki/Mandelbrot_set/

[4] Introduction to the Mandelbrot Set, http://www.olympus.net/personal/dewey/mandelbrot.html.

[5] Smooth shading for the Mandelbrot exterior. http://linas.org/art-gallery/escape/smooth.html. Linas Vepstas. October, 1997.

[6] PGM format specification. http://netpbm.sourceforge.net/doc/pgm.html.

[7] Oetiker, Tobias, "The Not So Short Introduction To LaTeX2", http://www.physics.udel.edu/∼dubois/lshort2e/