Univesidad de Buenos Aires - FIUBA
66:20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
2^o cuatrimestre de 2016

`$Date: 2016/08/30 04:13:03 $`

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa y su correspondiente documentación que resuelvan el problema descripto más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

Durante la primera clase del curso presentaremos brevemente los pasos necesarios para la instalación y configuración del entorno de desarrollo.

5. Programa

Se trata de diseñar un programa que permita dibujar el conjunto de Julia [3] y sus vecindades, en lenguaje C, correspondiente a un polinomio cuadrático.

El mismo recibirá, por línea de comando, una serie de parámetros describiendo la región del plano complejo, las características del archivo imagen a generar, y el parámetro c.

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* [4], un formato simple para describir imágenes digitales monocromáticas.

5.1. Algoritmo

El algoritmo básico es simple: para algunos puntos z de la región del plano que estamos procesando haremos un cálculo repetitivo. Terminado el cálculo, asignamos el nivel de intensidad del pixel en base a la condición de corte de ese cálculo.

El color de cada punto representa la "velocidad de escape" asociada con ese número complejo: blanco para aquellos puntos que pertenecen al conjunto (y por ende la "cuenta" permanece acotada), y tonos gradualmente más oscuros para los puntos divergentes, que no pertenezcan al conjunto.

Más específicamente: para cada pixel de la pantalla, tomaremos su punto medio, expresado en coordenadas complejas, $z = z_{re} + z_{im}i$. A continuación, iteramos sobre $z_{n+1} = z_n^2 + c$, con $z_0 = z$. Cortamos la iteración cuando $|z_n| > 2$, o después de `N` iteraciones.

En pseudo código:

```
para cada pixel $p {
        $z = complejo asociado a $p;
        for ($i = 0; $i < $N - 1; ++$i) {
                if (abs($z) > 2)
                        break;
                $z = $z * $z + $c;
        }
        dibujar el punto p con brillo $i;
}
```

Notar que c es un parámetro del programa.

Así tendremos, al finalizar, una representación visual de la cantidad de ciclos de cómputo realizados hasta alcanzar la condición de escape (ver figura 1).

5.2. Interfaz

A fin de facilitar el intercambio de código *ad-hoc*, normalizaremos algunas de las opciones que deberán ser provistas por el programa:

- `-r`: permite cambiar la resolución de la imagen generada. El valor por defecto será de 640x480 puntos.
- `-c`: para especificar el centro de la imagen, el punto central de la porción del plano complejo dibujada, expresado en forma binómica (i.e. $a + bi$). Por defecto usaremos $0 + 0i$.
- `-C`: determina el parámetro c, también expresado en forma binómica. El valor por defecto será $0{,}285 - 0{,}01i$.
- `-w`: especifica el ancho del rectángulo que contiene la región del plano complejo que estamos por dibujar. Valor por defecto: 4.

- `-H`: sirve, en forma similar, para especificar el alto del rectángulo a dibujar. Valor por defecto: 4.
- `-o`: permite colocar la imagen de salida, (en formato PGM [4]) en el archivo pasado como argumento; o por salida estándar `-stdout-` si el argumento es "-".

5.3. Casos de prueba

Es necesario que el informe trabajo práctico incluya una sección dedicada a verificar el funcionamiento del código implementado.

En el caso del TP 0, será necesario escribir pruebas orientadas a probar el programa completo, ejercitando los casos más comunes de funcionamiento, los casos de borde, y también casos de error.

Incluimos en el apéndice A algunos ejemplos de casos de interés, orientados a ejercitar algunos errores y condiciones de borde.

5.4. Ejemplos

Generamos un dibujo usando los valores por defecto, barriendo la región rectangular del plano comprendida entre los vértices $-2 + 2i$ y $+2 - 2i$.

```
$ tp0 -o uno.pgm
```

La figura 1 muestra la imagen `uno.pgm`.

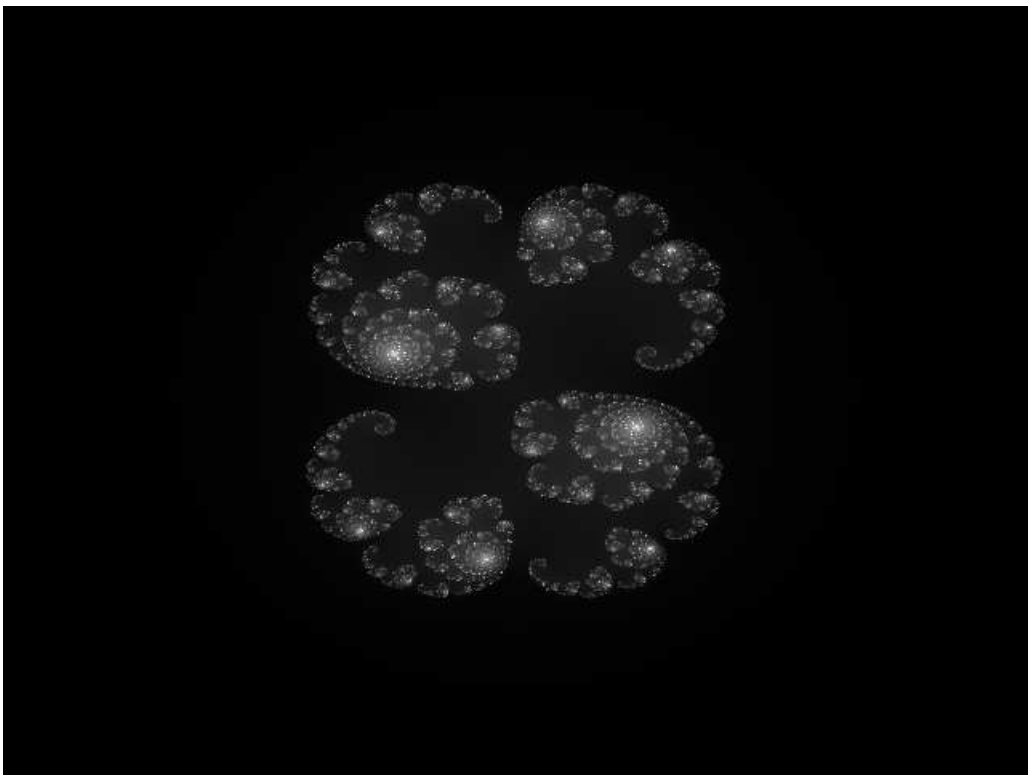

Figura 1: Región barrida por defecto.

A continuación, hacemos *zoom* sobre la región centrada en $+0{,}282 - 0{,}01i$, usando un rectángulo de 0,005 unidades de lado. El resultado podemos observarlo en la figura 2.

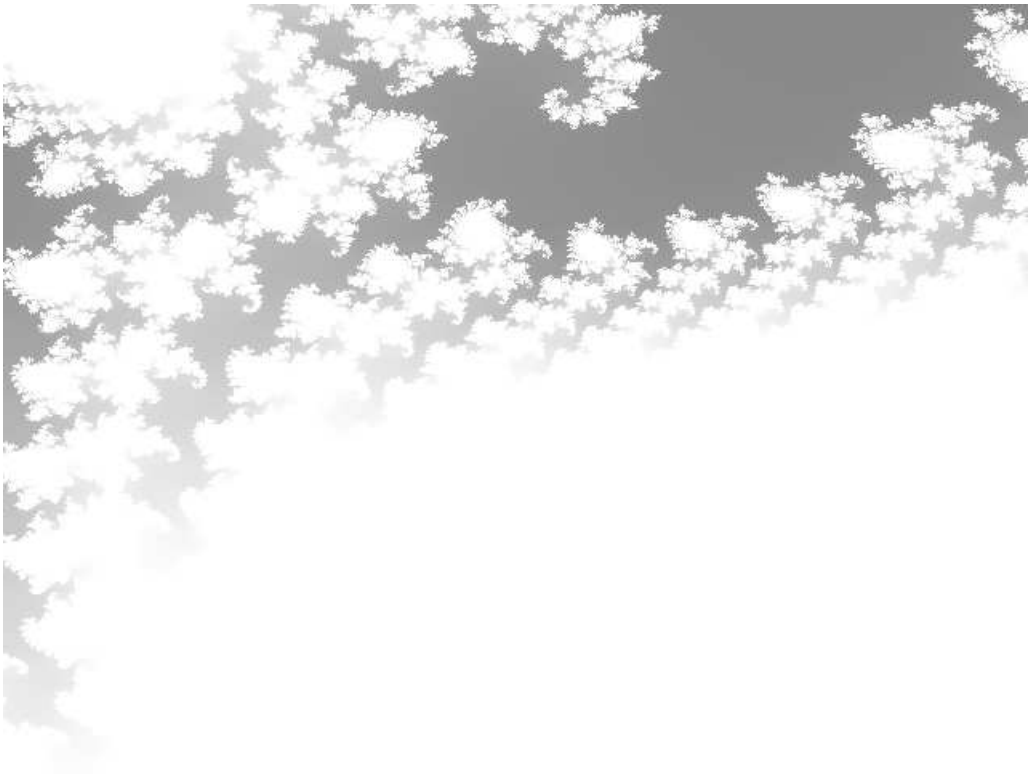

Figura 2: Región comprendida entre $0{,}2795 - 0{,}0075i$ y $0{,}2845 - 0{,}0125i$.

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

6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C.
- Este enunciado.

7. Fechas

Fecha de vencimiento: Martes 26/9.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

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

[3] http://en.wikipedia.org/wiki/Julia_set (Wikipedia).

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

A. Algunos casos de prueba

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

```
$ tp0 -c 0.01+0i -r 1x1 -o -
P2
1
1
255
255
```

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

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 no pertenece al conjunto de Julia.

3. Imagen imposible:

```
$ tp0 -c 0+0i -r 0x1 -o -
Usage:
  tp0 -h
  tp0 -V
...
```

4. Archivo de salida imposible:

```
$ tp0 -o /tmp
fatal: cannot open output file.
```

5. Coordenadas complejas imposibles:

```
$ tp0 -c 1+3 -o -
fatal: invalid center specification.
```

6. Argumentos de línea de comando vacíos,

```
$ tp0 -c "" -o -
fatal: invalid center specification.
```