Universidad Nacional de La Plata

Facultad de informática

Seminario de Lenguajes C

Trabajo Final Seminario C
PPM image processor

Estudiante:

Castro, Lautaro Germán; 20000/3

1. Introducción

Este trabajo consiste en la implementación de la funcionalidad faltante partiendo del programa proporcionado en el siguiente repositorio: https://github.com/chrodriguez/ppm-ejercicio-c.

El programa consiste en un procesador de imágenes en formato "PPM tipo P6". La operatoria básica con este formato de imagen (como creación, liberación, lectura desde archivo, etc) ya es proporcionada en el repositorio base (vease "ppm.h" y "ppm.c"), de esta manera la tarea a realizar consiste en un programa de línea de comandos que proporcione las siguientes operaciones/procesamiento (estas operaciones están descritas en "ppm-operations.h") sobre un archivo ppm:

- -i entrada.ppm: archivo origen. Opción requerida
- -o salida.ppm: archivo salida. Opción requerida
- -n: calcula el negativo
- -r: rota a 90 grados
- -h: espejo horizontal
- -v: espejo vertical
- -b NUM: desenfoque con radio NUM
- -?: ayuda del programa

2. Diseño del programa

El trabajo a realizar presenta dos aspectos principales a resolver:

- **Interfaz de línea de comandos (CLI):** Para resolver este aspecto se utiliza getopt que es una función que forma parte de las librerías estándar de C. Esta herramienta es un parseador de argumentos que de manera simple permite proporcionar una CLI.
- **Operatoria PPM:** En el repositorio base se especifican los detalles de implementación. De esta manera su resolución se vuelve directa siguiendo las especificaciones. Las siguientes especificaciones son extraidas del repositorio base:
  - **Negativo**: Cada píxel de la imagen resultante se calcula como el valor de profundidad de color (*depth*) correspondiente a la imagen original, menos el valor actual del píxel. Cuando el resultado es negativo, al tratarse de valores sin signo (*unsigned*), funciona correctamente.
  - **Rotar a 90 grados**: Una imagen de ancho x y alto y se rota 90 grados, obteniendo una nueva imagen de ancho y y alto x. Cada píxel en la posición original $[i, j]$ se coloca en la posición $[x - j - 1, i]$ en la imagen de destino.
  - **Espejado horizontal**: Cada píxel en la posición $[i, j]$ de la imagen resultante se obtiene del píxel en la posición $[i, \text{width} - j - 1]$ de la imagen original.
  - **Espejado vertical**: Cada píxel en la posición $[i, j]$ de la imagen resultante se obtiene del píxel en la posición $[\text{height} - i - 1, j]$ de la imagen original.
  - **Desenfoque**: Cada píxel en la posición $[i, j]$ con un radio r especificado como argumento se calcula como el promedio de los píxeles en las posiciones que van desde $[\text{máx}(i - r, 0), \text{máx}(j - r, 0)]$ hasta $[\text{mín}(i + r, \text{height} - 1), \text{mín}(j + r, \text{width} - 1)]$.

3. Pruebas del programa (testing)

A nivel del programa se decide realizar pruebas en dos aspectos principales: **operatoria PPM** y **memory leaks**.

Para la operatoria PPM se decide la realización de test unitarios, probando cada operatoria individualmente. Para llevarse a cabo se utiliza cmoka un framework de testing unitario para C.

Para los memory leaks se utiliza valgrind, una herramienta que permite realizar debug y pruebas a programas en Linux, este es independiente del lenguaje ya que trabaja con archivos binarios. En este caso para la prueba se crean varias ejecuciones del programa y se analizan si existen o no fugas de memoria.

4. Propuestas de evolución del trabajo

Finalizada la implementación del trabajo se analizaron varios aspectos a mejorar:

- **Procesamiento matricial de los píxeles:** El procesamiento de una imagen PPM generalmente consiste en recorrer la matriz de píxeles que lo representa y procesar cada celda. De esta manera para la implementación de cada operación la lógica es siempre la misma:

```
for row in input_ppm.height
    for col in input_ppm.width
        output_ppm[row][col] = /* operation to apply */
```

Código 1: Pseudo-código procesamiento típico de imagen PPM

Entonces se podría pensar en una función de alto orden que recorra la matriz de píxeles y aplique una función de mapeo que procesa el valor de la posición [x,y] de la matriz original y lo mapea en el nuevo ppm a generar:

```
fun process_ppm(
    input_ppm: ppm,
    output_ppm: ppm,
    map: fun (ppm in, ppm out, int row, int col)
)
    for row in input_ppm.height
        for col in input_ppm.width
            map(input_ppm, output_ppm, row, col)
```

Código 2: Pseudo-código función de alto orden para procesamiento de imagen PPM

Así por ejemplo para el cálculo del negativo map podría ser:

```
fun neg_map(ppm in, ppm out, int row, int col)
      out.pixels[row][col].red   = 255 - in.pixels[row][col].red;
      out.pixels[row][col].green = 255 - in.pixels[row][col].green;
      out.pixels[row][col].blue = 255 - in.pixels[row][col].blue;
```

Código 3: Pseudo-código map para cálculo del negativo

- **Paralelización del procesamiento:** Dada la naturaleza de las operaciones de los procesamientos realizados (operatoria matricial), podría articularse una descomposición de datos, distribuyendo a diferentes hilos de ejecución un subconjunto del conjunto de datos original a modo que cada uno realizarían el mismo cómputo (cómputo regular) sobre un conjunto diferente de datos. Librerías como pthreads u openMP son alternativas víables para llevarse a cabo esta idea.