UNIVERSIDAD DE ANTIOQUIA

1 8 0 3

Nombre de la institución: Universidad de Antioquia

Facultad o Departamento: Ingeniería

Título del informe: Reconstrucción de Imágenes BMP Transformadas en C++

Autores: Julian David Sanchez Garcia

Asignatura: Informática II

Facilitador o asesor: Augusto Salazar

Lugar y fecha: 25/04/2025

1. Análisis del problema

El problema a resolver consiste en revertir transformaciones a nivel de bits aplicadas a una imagen BMP para obtener su versión original. Estas transformaciones incluyen:

- Operaciones de bit como XOR, rotaciones y desplazamientos de bits aplicadas a los píxeles de la imagen.
- Enmascaramiento basado en una semilla y datos de rastreo que combinan la imagen transformada con valores RGB específicos, almacenados en archivos de texto (por ejemplo, M1.txt y M2.txt).
- Las principales dificultades incluyen:
    - Identificación del orden de las transformaciones: Las operaciones no se realizan de forma lineal; deben deducirse utilizando la semilla y los valores de enmascaramiento.
    - Implementación de algoritmos sin dependencias externas como STL: Las restricciones del desafío demandan uso de punteros y arreglos dinámicos en lugar de contenedores estándar.

2. Esquema de tareas en el desarrollo de los algoritmos

El desarrollo se organizó en las siguientes tareas principales:

Carga y procesamiento de la imagen BMP

- Cargar los píxeles de la imagen en un arreglo dinámico de tipo unsigned-char.
- Convertir la imagen al formato RGB888 (3 canales de color).

Implementación de operaciones a nivel de bits

- Crear funciones para realizar XOR, rotaciones (a izquierda y derecha) y desplazamientos (a izquierda y derecha) en bytes.

- Diseñar funciones reversibles, por ejemplo, rotar en la dirección opuesta para revertir una rotación inicial.
- **Procesamiento de los archivos de rastreo**
- Leer las semillas y valores enmascarados desde archivos de texto.
- Interpretar los valores para deducir el comportamiento de las transformaciones.
- **Integración de transformaciones en el flujo del programa**
- Aplicar las transformaciones a los píxeles de la imagen.
- Diseñar ejemplos de reversión basados en los valores de los archivos de rastreo.
- **Validación**
- Verificar que las operaciones (y sus inversas) producen los resultados esperados.
- Implementar mensajes de control para confirmar la consistencia del procesamiento.

3.Algoritmos implementados

- **Algoritmo para cargar y exportar imágenes:**
  - **loadPixels**: Carga la imagen, convierte sus datos al formato RGB888 y almacena los píxeles en un arreglo dinámico.
  - **exportImage**: Copia los datos del arreglo dinámico de píxeles a un objeto QImage y guarda la imagen resultante en disco.
- **Algoritmos de transformaciones bit a bit:**
  - **bitwiseXOR y applyXOR:** Realizan la operación XOR entre un byte o un conjunto de bytes y una clave dada.
  - **rotateRight y rotateLeft:** Implementan la rotación de bits hacia la derecha o izquierda respectivamente.
  - **shiftLeftOp y shiftRightOp:** Realizan desplazamientos de bits hacia la izquierda o derecha.
- **Algoritmo para procesar archivos de rastreo**
  - **loadSeedMasking:** Abre un archivo de texto, lee la semilla y los valores RGB, y los almacena en un arreglo dinámico de unsigned int.

4. Problemas de desarrollo que afrontó

- **Gestión de memoria**
    - El uso de arreglos dinámicos requiere liberar explícitamente la memoria asignada para evitar fugas. Durante las pruebas iniciales se detectaron errores al olvidar liberar memoria, lo cual se corrigió implementando bloques delete[].
- **Restricciones del uso de bibliotecas**
    - El uso prohibido de STL obligó a evitar funciones estándar como std::vector o std::array, optando por un diseño basado en punteros y memoria dinámica.
- **3. Dependencia de memcpy**
    - Inicialmente, se utilizó memcpy para copiar datos de líneas de píxeles, pero dado que se debía evitar el uso de este enfoque fue reemplazado por bucles manuales que copian byte por byte.
- **Validación de transformaciones**
    - Durante el desarrollo se realizaron pruebas individuales para cada operación (XOR, rotaciones, desplazamientos) y sus inversas. Al combinar varias operaciones, surgió la necesidad de un control más estricto para verificar los valores finales.

5. Evolución de la solución y consideraciones para la implementación

Evolución:

- **Versión inicial:**
    - Se implementaron funciones básicas para cargar y exportar imágenes, además de ejemplos simples de operaciones bit a bit.
- **Incorporación de archivos de rastreo:**
    - Se añadieron funciones para procesar las máscaras (M1.txt, M2.txt) y utilizar sus valores en transformaciones inversas.
- **Refinamiento y eliminación de dependencias:**
    - Se sustituyó memcpy por bucles manuales en loadPixels y exportImage.

Consideraciones para la implementación

- **Gestión de memoria:** Es crucial liberar correctamente todos los arreglos dinámicos para evitar fugas. Cualquier variable que use new[] debe tener un bloque correspondiente de delete[].
- **Validación exhaustiva:** Al procesar datos de enmascaramiento, se deben implementar verificaciones para asegurar que las operaciones inversas produzcan resultados coherentes.
- **Escalabilidad:** La solución actual está diseñada para funcionar con imágenes pequeñas. Para procesar imágenes de gran tamaño, podrían necesitar optimizaciones adicionales, especialmente en los bucles de transformación.