Informe de desarrollo del desafío 1

Informática II

Teoría

Keiner Marcelo Torres Villeros

Nikolas Geovanny Ortega Suarez

Docentes: Augusto Salazar Y Aníbal Guerra

2025-1

Introducción:

Este informe contiene el desarrollo del desafío 1 del curso de informática II. El objetivo principal era revertir transformaciones bit a bit las cuales fueron aplicadas sobre una imagen en formato BMP, en este programa no se utilizaron estructuras ni la STL de C++. Para el desarrollo de este se utilizó las librerías de Qt para la manipulación de imágenes, además del uso de arrays y punteros.

Desarrollo del desafío

El objetivo principal fue revertir una cadena de transformaciones aplicadas sobre una imagen BMP. Se partía de una imagen enmascarada (P2.bmp) y una máscara (I_M.bmp) y se debía recuperar la imagen original (I_O.bmp). Las operaciones incluyeron rotación de bits y operaciones XOR. Además, debía verificarse que una sección específica de la imagen coincidiera con datos almacenados en M1.txt, y se exigía generar archivos M1.txt y M2.txt coherentes a partir de los datos procesados.

Para el desarrollo de este desafío como primero se dio inicio con la carga de imágenes BMP las cuales se hizo el llamado con loadPixels(), en esta se usa Qimage para convertir las imágenes a formatos RGB888; al terminar este proceso se aplicó XOR entre la imagen original I_O y la máscara I_M generando así a P1.bmp.

El siguiente paso para encontrar la imagen original fue aplicar rotación de bits a la derecha sobre P1.bmp para así generar P2.bmp, el siguiente paso fue generar la imagen distorsionada I_D la cual venia incrementada en el código proporcionado por el docente, esta fue creada desde I_O para validar escritura BMP.

Para la lectura y verificación de M1.txt se utilizó loadSeedMasking() la cual cargo una semilla (offset) y un conjunto de valores RGB para poder verificar su relación con P2.bmp; al terminar este proceso llena la generación de archivos en esta se creó la función generarM1DesdeP2(), esta función se creó para generar M1.txt y generarM2DesdeP1() generó M2.txt para comprobar su coherencia con la máscara.

Para el proceso inverso se utilizó rotación a la izquierda sobre P2.bmp y luego a este se le hizo XOR con I_M para poder así generar P3.bmp la cual fue la imagen recuperada, como último paso se utilizó la función compararImagenes() la cual validó que P3.bmp sea igual a I_O , con esta también se compararon archivos de textos generados con las versiones originales (M1.txt y M2.txt).

Uno de los problemas más importantes que se presentaron en el desarrollo de este desafío fueron los valores fuera de rango en M1.txt: Inicialmente se detectaron valores RGB mayores a 255. Esto se debió a una mala generación del archivo. Se solucionó generando M1.txt con generarM1DesdeP2() de forma coherente.

Otro de los errores comunes pero muy importantes fue la re declaración de variables, ocurrió una doble definición de n_pixels. Se resolvió reorganizando su declaración.

Entre estos problemas también presenté un desajuste en archivos M1.txt y M2.txt: Al comparar con versiones originales, hubo diferencias. Estas se justificaron debido a la falta de información sobre las transformaciones exactas de generación utilizados al inicio del desafío.

Las funciones que se utilizaron para el desarrollo de este desafío fueron:

- loadPixels(): Carga una imagen BMP y extrae sus datos RGB como arreglo lineal.

- exportImage(): Guarda una imagen BMP a partir de datos RGB en memoria.

- applyXOR(): Aplica operación XOR entre dos imágenes byte a byte.

- rotateBitsRight() / rotateBitsLeft(): Rotan bits hacia la derecha o izquierda a nivel de byte, usadas para enmascarar y deshacer enmascaramiento.

- loadSeedMasking(): Lee una semilla y valores RGB desde un archivo .txt, típicamente M1.txt.

- generarM1DesdeP2(): Genera el archivo M1.txt sumando P2.bmp y M.bmp, desde cierto offset.

- generarM2DesdeP1(): Similar a la anterior, pero genera M2.txt desde P1.bmp.

- compararImagenes(): Compara dos imágenes BMP para verificar si son idénticas bit a bit.

- compararArchivos(): Compara dos archivos de texto línea por línea para verificar integridad.

Conclusión:

El desarrollo del Desafío 1 nos permitió aplicar conceptos de manipulación de imágenes a nivel de bytes y bit a bit en C++, sin usar estructuras ni la STL. A través del uso de operaciones XOR, rotaciones de bits y una gestión cuidadosa de la memoria, logramos implementar un sistema capaz de revertir el enmascaramiento aplicado sobre una imagen.

Se resolvieron problemas como sobrepasar los rangos RGB, errores de definición múltiple y generación inconsistente de archivos, mediante diseño modular, pruebas y depuración.

El código final no solo cumple con todos los requisitos funcionales del desafío, sino que también es adaptable a diferentes imágenes y casos siempre y cuando sean una imagen BMP en formato RGB888.