Proyecto Individual - Aplicación para Interpolación de Imágenes

Juan Rodríguez Montero

juan.rodriguez@estudiantec.cr

Área Académica Ingeniería en Computadores

Instituto Tecnológico de Costa Rica

2

I. Listado de Requerimientos del Sistema

El sistema desarrollado tiene como objetivo realizar interpolación de imágenes utilizando algoritmos escritos en lenguaje ensamblador x86. A continuación, se listan los principales requerimientos:

- Manipulación y procesamiento de imágenes binarias en formato RAW.
- Interpolación de un bloque de imagen extraído de un cuadrante específico.

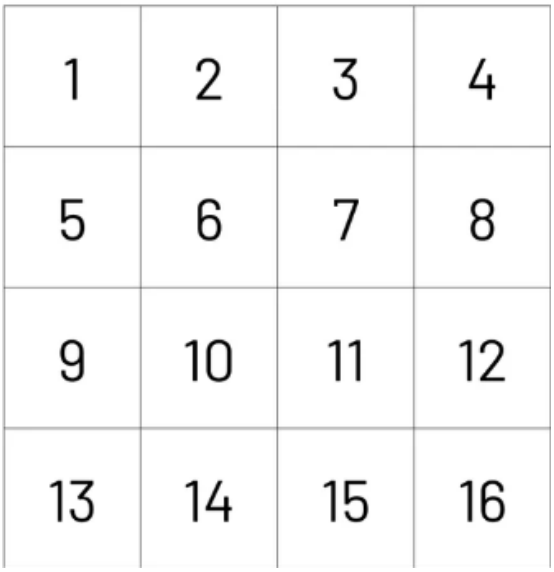

Figura 2. División de la imagen original en 16 cuadrantes.

- Generación de archivos binarios intermedios con la información transformada.
- Visualización de la imagen original, el cuadrante seleccionado y la imagen interpolada.
- Cumplimiento con la arquitectura x86 de 32 bits.
- Uso de herramientas libres y multiplataforma para desarrollo y pruebas.

II. Elaboración de Opciones de Solución

II-A. Opción 1: Simulación con DOSBox

La primera alternativa contemplada fue realizar el desarrollo usando **DOSBox**, un emulador de entorno MS-DOS, que permite compilar y ejecutar código NASM de 16 bits en plataformas modernas. Se propuso integrar este entorno con scripts en Python en Windows, montando un directorio compartido entre el sistema anfitrión y DOSBox para el manejo de archivos de imagen.

Esta solución requiere convertir los algoritmos a código de 16 bits, lo cual complica el manejo de imágenes grandes, por las restricciones de segmentación de memoria. Además, el entorno no ofrece herramientas modernas de debugging ni una integración fluida con sistemas de visualización.

II-B. Opción 2: Uso de entorno Linux con NASM de 32 bits (Implementada)

La solución elegida fue utilizar un entorno Linux con el ensamblador NASM en modo protegido (32 bits), lo cual permitió un manejo directo de archivos binarios y acceso completo a la memoria lineal.

El entorno de desarrollo consistió en:

- NASM para compilación del código ensamblador.
- Python para la visualización de imágenes y automatización del flujo.
- Scripts Bash para ejecución secuencial de pruebas.

Esta opción simplificó la implementación, aumentó la eficiencia del desarrollo y permitió un control más preciso sobre los archivos y el flujo de datos.

III. Comparación de Opciones de Solución

- **Compatibilidad:** DOSBox emula una arquitectura más antigua (16 bits), lo cual reduce la compatibilidad con herramientas actuales. La solución en Linux permite usar software moderno sin restricciones.
- **Memoria:** DOSBox impone límites estrictos debido a la segmentación de memoria. En Linux 32-bit se tiene acceso a un espacio más amplio y lineal.
- **Facilidad de Desarrollo:** La integración entre Python y NASM es mucho más directa en Linux, mientras que en DOSBox requiere pasos adicionales y complejos.

- **Escalabilidad:** La solución implementada puede adaptarse fácilmente para trabajar con imágenes más grandes o mayor cantidad de cuadrantes.

IV. Selección de la Propuesta Final

La solución elegida fue la opción 2, utilizando Linux con NASM de 32 bits. Esta elección se basó en criterios técnicos que facilitaron el desarrollo modular del sistema, permitieron una integración sencilla con herramientas externas y redujeron los problemas asociados al manejo de memoria y compatibilidad.

Consideración sobre otras arquitecturas (ARM y RISC-V)

Durante la etapa de planificación también se consideraron otras arquitecturas como ARM y RISC-V, ya que existen simuladores y entornos de desarrollo disponibles para ambas. Sin embargo, se descartaron por las siguientes razones:

- **Limitaciones en visualización:** Las herramientas de visualización y procesamiento de imágenes en simuladores de ARM y RISC-V son menos accesibles y presentan una curva de configuración más alta.
- **Complejidad de integración:** Integrar simuladores de ARM o RISC-V con scripts externos (Python, Bash) requiere una configuración avanzada y complica el flujo de desarrollo iterativo.
- **Requerimientos del curso:** El curso exige un trabajo en lenguaje ensamblador, sin restricción explícita a una arquitectura, pero el entorno x86 fue más natural para implementar flujos automatizados y depuración local.
- **Madurez del ecosistema:** x86 cuenta con herramientas robustas, documentación extensa y una comunidad activa, lo que facilitó el avance ágil del proyecto.

V. Opciones de Desarrollo Lógico del Software

V-A. Opción 1: Algoritmo secuencial por bloques (Implementado)

El enfoque implementado fue plantear el problema como un procesamiento secuencial por bloques: primero se selecciona un cuadrante de la imagen, se extraen los píxeles base, se almacena un bloque intermedio de 97x97 y luego se genera un bloque expandido de 193x193 mediante interpolación bilineal por pasos.

Este método fue elegido por su simplicidad y claridad para separar etapas de lectura, interpolación horizontal y vertical, lo cual facilitó el debugging y validación de resultados.

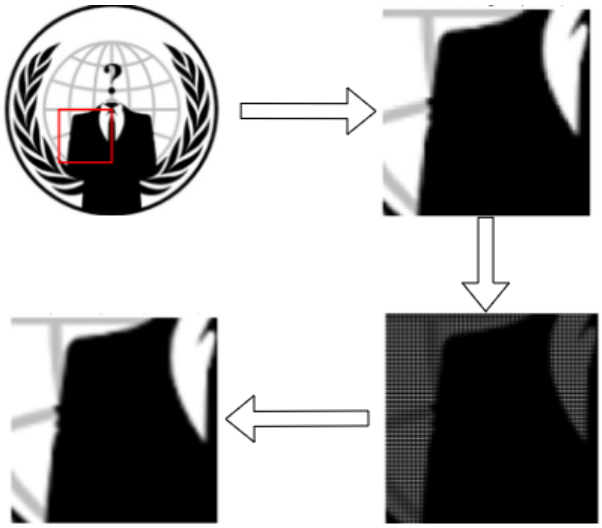

Figura 1. Diagrama de flujo general del sistema de interpolación de imágenes.

V-B. Opción 2: Procesamiento de imagen completo con máscara

Otra opción considerada fue procesar toda la imagen en un solo paso, aplicando una máscara que definiera qué región (cuadrante) debía ser interpolada, y descartando el resto. Este enfoque prometía ser más eficiente en términos de ejecución, pero introducía una mayor complejidad en el control de flujo y dificultaba la verificación de cada etapa intermedia.

Finalmente se optó por un enfoque modular y más transparente, especialmente adecuado para entornos de bajo nivel como ensamblador.

VI. Descripción del Funcionamiento por Etapas

El programa opera en una secuencia clara de etapas:

1. Se selecciona uno de los 16 cuadrantes de la imagen de entrada (formato RAW, resolución 390x390).
2. Se extrae una grilla base de 97x97 píxeles desde el cuadrante, tomando únicamente los píxeles de coordenadas pares $(2i, 2j)$.
3. Se realiza la interpolación horizontal, generando un bloque de 193x97 donde se insertan valores interpolados entre píxeles consecutivos en cada fila.
4. Se aplica interpolación vertical sobre el resultado anterior, produciendo un bloque final de 193x193 con valores interpolados entre filas.
5. El bloque final se guarda en un nuevo archivo `output.img`, que puede visualizarse mediante herramientas de Python.