Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Arquitectura de Computadores I (CE-4301)

Proyecto Individual:

Diseño de la implementación de interpolación bilineal

Profesores:
Jeferson Gonzales Gomez
Luis Alberto Chavarría Zamora

Estudiante:
Adriel Sebastián Chaves Salazar

I Semestre 2025

Tabla de Contenidos

1. Introducción

La interpolación bilineal representa un método fundamental en el procesamiento digital de imágenes para aumentar la resolución sin perder calidad significativa [1]. Este documento presenta el diseño e implementación de un sistema de interpolación bilineal desarrollado en lenguaje ensamblador, destacando la integración entre programación de bajo nivel y las interfaces modernas de alto nivel. La implementación en lenguaje ensamblador de algoritmos de procesamiento de imágenes constituye un desafío técnico significativo que permite explorar las capacidades fundamentales de los conjuntos de instrucciones de los procesadores modernos [2].

El presente proyecto aborda la implementación de un algoritmo de interpolación bilineal en lenguaje ensamblador x86 de 64 bits, utilizando Python como framework integrador. La elección de x86 como ISA (Instruction Set Architecture) responde a la naturaleza CISC (Complex Instruction Set Computing) de esta arquitectura, que ofrece instrucciones especializadas para operaciones matemáticas complejas, facilitando la implementación de los cálculos necesarios para la interpolación bilineal [3]. El sistema desarrollado permite procesar imágenes de 512×512 píxeles, seleccionar cuadrantes específicos de 128×128 píxeles, y aplicar interpolación bilineal para generar una imagen resultante de 256×256 píxeles.

Una característica distintiva de nuestra implementación es la organización de píxeles en bloques estructurados (2×2 para entrada y 4×4 para salida), permitiendo una ejecución más eficiente del algoritmo en código ensamblador. Esta aproximación simplifica significativamente la lógica de procesamiento en lenguaje de bajo nivel, delegando la interpretación espacial de los datos a Python, mientras que el cálculo matemático de interpolación se realiza exclusivamente en ensamblador.

El documento se estructura en cuatro secciones principales: primero, un listado detallado de requerimientos del sistema, considerando aspectos técnicos, estándares aplicables y el estado del arte en procesamiento de imágenes. Segundo, la elaboración de opciones de solución al problema planteado, cada una acompañada de diagramas explicativos y fundamentada en criterios técnicos. Tercero, una comparación objetiva de las opciones presentadas, evaluando aspectos como eficiencia computacional, complejidad de implementación y alineación con los requerimientos. Finalmente, la selección y descripción detallada de la propuesta final implementada, justificando las decisiones tomadas y demostrando el cumplimiento de los objetivos establecidos.

2. Requerimientos del sistema

Este apartado presenta un análisis detallado de los requerimientos de ingeniería identificados para el desarrollo del sistema de interpolación bilineal en lenguaje ensamblador. Se consideran aspectos técnicos, partes involucradas, estado del arte y estándares aplicables al problema.

2.1. Requerimientos de ingeniería

Los requerimientos técnicos establecidos para el sistema de interpolación bilineal implementado en lenguaje ensamblador son los siguientes:

1. **Procesamiento exclusivo en ensamblador:** El algoritmo de interpolación bilineal debe ser implementado completamente en lenguaje ensamblador, sin recurrir a funciones de procesamiento de imágenes de alto nivel para los cálculos matemáticos [3].

2. **Almacenamiento local:** Las imágenes de entrada y salida deben almacenarse en el mismo directorio de ejecución para facilitar el acceso y la verificación.

3. **Generación de archivos:** El programa en ensamblador debe ser capaz de generar un archivo de salida que contenga la imagen interpolada, respetando el formato binario establecido.

4. **Selección de cuadrantes:** El sistema debe permitir al usuario seleccionar uno de los 16 cuadrantes posibles de la imagen original (matriz 4×4 de cuadrantes) y mostrar el resultado de la interpolación aplicada solo a dicho cuadrante.

5. **Formato de imagen:** Todas las imágenes procesadas deben estar en escala de grises con valores de píxel en el rango [0, 255], facilitando los cálculos matemáticos en el procesador x86 [1].

6. **Integración con software de visualización:** La visualización de las imágenes debe realizarse mediante un software de alto nivel (Python en nuestro caso), mientras que todo el procesamiento algorítmico debe ejecutarse exclusivamente en ensamblador.

7. **Framework automatizado:** Todo el sistema debe integrarse en un framework que permita la ejecución automática del proceso completo, desde la lectura del archivo original hasta la visualización del resultado final.

8. **Entorno de desarrollo:** El sistema se ha desarrollado en Linux/Ubuntu para facilitar la integración con el compilador NASM y aprovechar la gestión de procesos del sistema operativo [2].

Estos requerimientos técnicos se establecieron considerando las limitaciones inherentes a la programación en lenguaje ensamblador y la necesidad de crear un sistema integrado que funcione de manera eficiente.

2.2. Partes involucradas

El desarrollo del sistema considera diversas partes involucradas que establecen requisitos específicos o se ven afectadas por el diseño:

- **Usuario final:** Debe poder seleccionar imágenes, visualizar los resultados y comprender el proceso de interpolación bilineal mediante una interfaz gráfica intuitiva.

- **Desarrollador:** Requiere un entorno que facilite la implementación del algoritmo en ensamblador y su integración con componentes de alto nivel.

- **Evaluador:** Necesita verificar que el procesamiento se realiza exclusivamente en ensamblador y que los resultados de la interpolación son correctos según los algoritmos matemáticos establecidos.

La consideración de estas partes ha influido en decisiones como la implementación de una interfaz gráfica utilizando Tkinter y el desarrollo de un sistema automatizado mediante scripts de Bash.

2.3. Estado del arte

En el campo del procesamiento de imágenes y la programación en bajo nivel, existen diversos enfoques y tecnologías relevantes para nuestro sistema:

1. **Arquitecturas de procesadores:** Actualmente, las arquitecturas CISC como x86-64 y las arquitecturas RISC como ARM y RISC-V compiten en el mercado, cada una con ventajas específicas para diferentes aplicaciones [4]. Nuestra elección de x86-64 se basa en su amplia disponibilidad y en la riqueza de instrucciones matemáticas que facilitan los cálculos de interpolación.

2. **Algoritmos de interpolación:** Existen diversos métodos para aumentar la resolución de imágenes, incluyendo interpolación por vecino más cercano, bilineal, bicúbica y basada en wavelets [1]. La interpolación bilineal representa un equilibrio entre calidad y complejidad computacional adecuado para implementación en ensamblador.

3. **Conjuntos de instrucciones específicos:** Las extensiones AVX y SSE de x86 permiten operaciones vectoriales que podrían optimizar el procesamiento de imágenes [5], aunque en nuestra implementación optamos por instrucciones estándar para mantener la claridad del código.

4. **Enfoques híbridos:** La tendencia actual favorece sistemas híbridos donde los componentes críticos se implementan en bajo nivel mientras que la interfaz y gestión de archivos se desarrollan en lenguajes de alto nivel [3], enfoque que adoptamos en nuestro diseño.

2.3.1. Interpolación bilineal y especificación

Interpolación bilineal

El proceso de interpolación bilineal es un poco más complejo pues trata de rellenar el espacio con información continua, no replicando un mismo valor. Por ejemplo, el proceso de interpolación bilineal para la imagen de la ecuación (1). Primero se comienza con los valores que ya se conocen.

Píxeles conocidos

Los píxeles que se usan de referencia son los marcados en rojo, los índices se muestran entre paréntesis en color azul

$$I' = \begin{bmatrix} 10(1) & a(2) & b(3) & 20(4) \\ c(5) & d(6) & e(7) & f(8) \\ g(9) & h(10) & i(11) & j(12) \\ 30(13) & k(14) & l(15) & 40(16) \end{bmatrix}. \tag{2}$$

El procedimiento para obtener los valores de la a a l. Se muestra a continuación.

Píxeles horizontales y verticales

Estos píxeles se calculan como si fuera una interpolación lineal. Se le da un mayor peso a los valores más cercanos al conocido. Los valores horizontales y verticales se muestran a continuación (son operaciones con enteros):

1. $a = \frac{4-2}{4-1} \times 10 + \frac{2-1}{4-1} \times 20 = \frac{2}{3} \times 10 + \frac{1}{3} \times 20 = 13$
2. $b = \frac{4-3}{4-1} \times 10 + \frac{3-1}{4-1} \times 20 = \frac{1}{3} \times 10 + \frac{2}{3} \times 20 = 17$
3. $c = \frac{13-5}{13-1} \times 10 + \frac{5-1}{13-1} \times 30 = \frac{2}{3} \times 10 + \frac{1}{3} \times 30 = 17$
4. $g = \frac{13-9}{13-1} \times 10 + \frac{9-1}{13-1} \times 30 = \frac{1}{3} \times 10 + \frac{2}{3} \times 30 = 23$
5. $k = \frac{16-14}{16-13} \times 30 + \frac{14-13}{16-13} \times 40 = \frac{2}{3} \times 30 + \frac{1}{3} \times 40 = 33$
6. $l = \frac{16-15}{16-13} \times 30 + \frac{15-13}{16-13} \times 40 = \frac{1}{3} \times 30 + \frac{2}{3} \times 40 = 37$
7. $f = \frac{16-8}{16-4} \times 20 + \frac{8-4}{16-4} \times 40 = \frac{2}{3} \times 20 + \frac{1}{3} \times 40 = 27$
8. $j = \frac{16-12}{16-4} \times 20 + \frac{12-4}{16-4} \times 40 = \frac{1}{3} \times 20 + \frac{2}{3} \times 40 = 33$

Para mayor guía se rellenan los valores faltantes en

$$I' = \begin{bmatrix} 10(1) & 13(2) & 17(3) & 20(4) \\ 17(5) & d(6) & e(7) & 27(8) \\ 23(9) & h(10) & i(11) & 33(12) \\ 30(13) & 33(14) & 37(15) & 40(16) \end{bmatrix}. \tag{3}$$

Píxeles intermedios

Estos píxeles se calculan usando los píxeles verticales y horizontales. Igual que el método anterior se le da mayor peso a los píxeles cercanos, en este caso los recién calculados en el paso anterior, como se observa en

$$I' = \begin{bmatrix} 10(1) & 13(2) & 17(3) & 20(4) \\ 17(5) & d(6) & e(7) & 27(8) \\ 23(9) & h(10) & i(11) & 33(12) \\ 30(13) & 33(14) & 37(15) & 40(16) \end{bmatrix}. \tag{4}$$

Para interpolar los píxeles d, e, h, i. Para estos valores se puede inferir ya sea horizontal o verticalmente, sin importar el eje se obtiene el mismo valor. En este caso se infiere horizontalmente:

1. $d = \frac{8-6}{8-5} \times 17 + \frac{6-5}{8-5} \times 27 = \frac{2}{3} \times 17 + \frac{1}{3} \times 27 = 20$
2. $e = \frac{8-7}{8-5} \times 17 + \frac{7-5}{8-5} \times 27 = \frac{1}{3} \times 17 + \frac{2}{3} \times 27 = 24$
3. $h = \frac{12-10}{12-9} \times 23 + \frac{10-9}{12-9} \times 33 = \frac{2}{3} \times 23 + \frac{1}{3} \times 33 = 26$
4. $h = \frac{12-11}{12-9} \times 23 + \frac{11-9}{12-9} \times 33 = \frac{1}{3} \times 23 + \frac{2}{3} \times 33 = 30$

La representación final de la matriz se observa en

$$I' = \begin{bmatrix} 10(1) & 13(2) & 17(3) & 20(4) \\ 17(5) & 20(6) & 24(7) & 27(8) \\ 23(9) & 26(10) & 30(11) & 33(12) \\ 30(13) & 33(14) & 37(15) & 40(16) \end{bmatrix} = \begin{bmatrix} 10 & 13 & 17 & 20 \\ 17 & 20 & 24 & 27 \\ 23 & 26 & 30 & 33 \\ 30 & 33 & 37 & 40 \end{bmatrix}. \tag{5}$$

Esta operación se realiza para cada cuadrante identificado.

2.3.2. Resultado de interpolación

Se observa que para la imagen

$$I = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}, \tag{6}$$

y para interpolación bilineal sería

$$I'' = \begin{bmatrix} 10 & 13 & 17 & 20 \\ 17 & 20 & 24 & 27 \\ 23 & 26 & 30 & 33 \\ 30 & 33 & 37 & 40 \end{bmatrix}. \tag{7}$$

2.4. Estándares y normas aplicables

El desarrollo del sistema se ha regido por diversos estándares y buenas prácticas:

- **Estándares de formato de imagen:** Se adopta el formato binario estándar para la representación de imágenes en escala de grises, siguiendo las convenciones establecidas en el procesamiento digital de imágenes [1].

- **Convenciones de programación en ensamblador:** Se siguen las convenciones de llamada de sistema para x86-64 en Linux, utilizando registros específicos para parámetros de entrada y salida según lo establecido por el ABI del sistema [3].
- **Estándares de modularidad:** El código sigue principios de diseño modular, separando funcionalidades específicas en secciones diferenciadas del código ensamblador y manteniendo una clara separación entre procesamiento (ensamblador) y presentación (Python) [6].
- **Manejo de excepciones:** Se implementan mecanismos de detección y gestión de errores tanto en el código ensamblador como en la interfaz de Python, siguiendo las recomendaciones de fiabilidad para sistemas software [6].

Estas consideraciones de estándares y normas garantizan que el sistema desarrollado sea robusto, mantenible y conforme a las prácticas aceptadas en la industria del software y el procesamiento de imágenes. Estas consideraciones de estándares y normas garantizan que el sistema desarrollado sea robusto, mantenible y conforme a las prácticas aceptadas en la industria del software y el procesamiento de imágenes.

3. Elaboración de opciones de solución al problema

En esta sección se presentan las distintas alternativas de solución consideradas para implementar el sistema de interpolación bilineal. Cada opción ha sido analizada en función de criterios técnicos y teóricos, buscando soluciones viables que respondan a los requerimientos establecidos.

3.1. Opciones para el lenguaje de alto nivel del framework

Para desarrollar el framework integrador que permitiera automatizar todo el proceso desde la lectura de la imagen hasta la visualización del resultado, se consideraron las siguientes alternativas:

1. **Python:** Ofrece una amplia variedad de bibliotecas para procesamiento de imágenes como PIL/Pillow, OpenCV y NumPy, además de capacidades para crear interfaces gráficas con Tkinter o PyQt. Su naturaleza interpretada facilita la integración con scripts del sistema operativo y la invocación de programas externos como compiladores de ensamblador [1].

2. **MATLAB/Octave:** Proporcionan potentes capacidades de procesamiento matemático y visualización, especialmente útiles para algoritmos de procesamiento de imágenes. Cuentan con funciones integradas para la interpolación bilineal y la visualización de matrices como imágenes [1]. Sin embargo, la integración con código ensamblador podría requerir el desarrollo de interfaces adicionales.

3. **C/C++:** Ofrecen mayor eficiencia computacional y control sobre los recursos del sistema, lo que podría ser beneficioso para el procesamiento de imágenes grandes. Existen bibliotecas como OpenCV que facilitan el manejo de imágenes, y la integración con código ensamblador es natural mediante el uso de archivos objeto o funciones inline [2].

3.2. Alternativas de ISA para implementación en ensamblador

Se evaluaron tres arquitecturas de conjunto de instrucciones (ISA) para la implementación del algoritmo de interpolación bilineal:

1. **x86-64:** Arquitectura CISC que proporciona instrucciones complejas potencialmente útiles para cálculos matemáticos. Ofrece mayor cantidad de registros en su versión de 64 bits y extensiones SIMD como SSE y AVX que podrían optimizar operaciones en paralelo para procesamiento de píxeles [4]. Su amplia disponibilidad en sistemas de escritorio facilita el desarrollo y pruebas.

2. **ARM:** Arquitectura RISC que se caracteriza por su eficiencia energética y diseño simplificado. Ofrece un conjunto de instrucciones más regular y predecible, aunque potencialmente requeriría más instrucciones para implementar operaciones complejas como las divisiones en punto fijo necesarias para la interpolación bilineal [5].

3. **RISC-V:** Arquitectura RISC de código abierto con especificaciones modulares y extensibles. Su diseño limpio y ortogonal facilitaría la comprensión del código, aunque al igual que ARM, podría requerir secuencias más largas de instrucciones para operaciones complejas [7].

3.3. Opciones para el tamaño y formato de imagen

La selección del tamaño y formato de la imagen de entrada representa un factor crítico para la implementación eficiente del algoritmo:

1. **Tamaño original (390×390 o mayor):** Utilizar la imagen original sin modificaciones garantizaría la preservación de todos los detalles, pero podría complicar la división en cuadrantes si las dimensiones no son múltiplos de 16.

2. **Tamaño 500×500:** Esta dimensión es fácilmente divisible por cuadrantes de 16 (resultando en cuadrantes de 125×125), pero presenta desafíos para la implementación del algoritmo de interpolación bilineal, que requiere bloques de 2×2 píxeles que se expanden a 4×4.

3. **Tamaño 512×512:** Este tamaño es una potencia de 2, lo que facilita diversas operaciones de procesamiento de imágenes. Es divisible por 16 (resultando en cuadrantes de 128×128), y estos cuadrantes son a su vez divisibles por 4, permitiendo una implementación más limpia del algoritmo de interpolación bilineal.

3.4. Estrategias para la organización de píxeles

La forma de organizar y procesar los píxeles en el algoritmo representa una decisión crucial para la implementación en ensamblador:

1. **Organización lineal:** Los píxeles se almacenarían y procesarían en un arreglo unidimensional, donde cada posición contiene el valor de un píxel. Esta aproximación simplifica el almacenamiento, pero complicaría la implementación del algoritmo de interpolación bilineal que requiere acceso a bloques de 2×2 píxeles, necesitando cálculos adicionales de índices [1].

2. **Organización por bloques 2×2 para entrada:** Los píxeles se organizan y escriben en bloques de 2×2 en el archivo de entrada. Esto alinea la estructura de datos con la naturaleza del algoritmo de interpolación bilineal, que opera sobre cuadrantes de 2×2 píxeles para generar bloques expandidos de 4×4.

3. **Organización por bloques 4×4 para salida:** Los píxeles interpolados se escriben en bloques de 4×4 en el archivo de salida. Esta organización facilita la generación del resultado, ya que cada bloque de entrada 2×2 se transforma directamente en un bloque de salida 4×4.

3.5. Diagrama de flujo de las soluciones consideradas

A continuación se presenta un diagrama que ilustra las diferentes rutas de solución consideradas para el problema, en la figura 1.

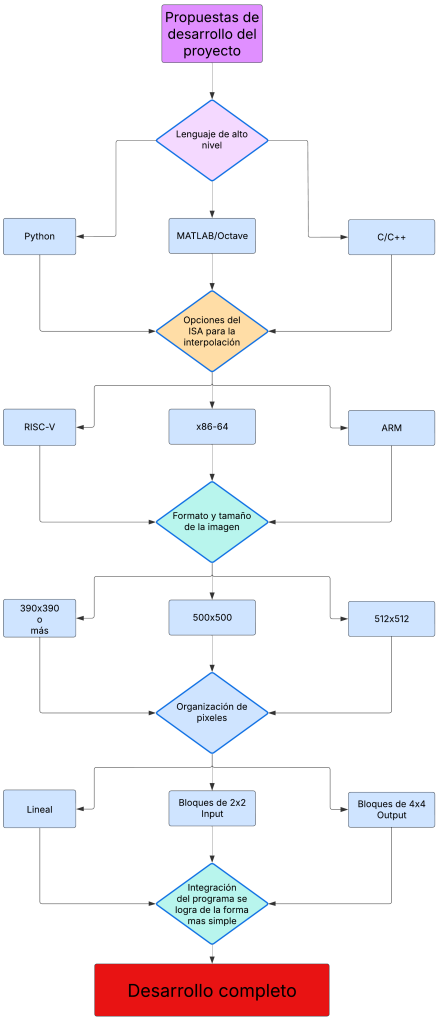

Figura 1: Diagrama de flujo de las opciones de solución consideradas

Y como se puede observar en la imagen 1 se consideraron todas las opciones que nos llevarian al desarrollo completo del problema propuesto.

4. Comparación de soluciones

En esta sección se realiza un análisis comparativo de las opciones presentadas anteriormente, evaluando objetivamente cada alternativa según criterios técnicos, económicos y de compatibilidad con los requerimientos establecidos para el sistema de interpolación bilineal.

4.1. Comparación de lenguajes de alto nivel para el framework

La elección del lenguaje de alto nivel para desarrollar el framework integrador es crucial para la usabilidad y versatilidad del sistema completo. La Tabla 1 presenta una comparación detallada de las tres alternativas consideradas.

Criterio	**Python**	**MATLAB/Octave**	**C/C++**
Facilidad de integración con ensamblador	Alta (mediante subprocess y llamadas al sistema)	Media (requiere mecanismos adicionales)	Alta (enlace directo con código objeto)
Bibliotecas para manejo de imágenes	Excelentes (PIL, NumPy, OpenCV)	Excelentes (funciones integradas)	Buenas (OpenCV, libjpeg)
Facilidad de desarrollo de GUI	Alta (Tkinter, PyQt)	Media (GUIDE/App Designer)	Baja (requiere bibliotecas externas)
Curva de aprendizaje	Baja	Media	Alta
Rendimiento	Medio	Medio-Alto para operaciones matriciales	Alto

Cuadro 1: Comparación de lenguajes de alto nivel para el framework

4.2. Evaluación comparativa de arquitecturas ISA

La selección de la arquitectura de conjunto de instrucciones (ISA) determina fundamentalmente las capacidades y limitaciones del código ensamblador. La Tabla 2 presenta la comparación entre las tres arquitecturas consideradas.

Criterio	**x86-64**	**ARM**	**RISC-V**
Paradigma	CISC (instrucciones complejas)	RISC (instrucciones simples)	RISC (instrucciones simples)
Instrucciones matemáticas	Amplias, incluye divisiones	Limitadas, requiere más instrucciones	Limitadas, requiere más instrucciones
Registros disponibles	16 registros de propósito general	16 registros (32 en versiones más recientes)	32 registros de propósito general
Extensiones SIMD	SSE, AVX para procesamiento paralelo	NEON	Vector extensions (opcionales)
Herramientas disponibles	Excelente (NASM, MASM, GAS)	Buena (GCC, Clang)	Limitada pero creciente
Documentación	Extensa	Buena	Limitada pero creciente
Complejidad de programación	Alta debido a irregularidades	Media, más consistente	Baja, diseño ortogonal

Cuadro 2: Comparación de arquitecturas ISA

Como señalan [4], las arquitecturas CISC como x86-64 ofrecen instrucciones más potentes y especializadas, lo que potencialmente reduce la cantidad de código necesario para implementar operaciones complejas como la interpolación bilineal. La disponibilidad de instrucciones de división y multiplicación en un solo paso es particularmente relevante para los cálculos de ponderación en el algoritmo.

Por otro lado, [5] destacan que las arquitecturas RISC como ARM proporcionan un conjunto de instrucciones más regular y predecible, facilitando el aprendizaje y la depuración. RISC-V, siendo una arquitectura abierta y

modular, ofrece ventajas en términos de personalización y adaptabilidad [7], pero su ecosistema de herramientas es menos maduro.

4.3. Análisis de opciones de dimensionamiento de imágenes

El tamaño de la imagen es determinante para la eficiencia del algoritmo y la correcta implementación de la interpolación bilineal. La Tabla 3 compara las diferentes opciones consideradas.

Criterio	**Original (390×390+)**	**500×500**	**512×512**
Compatibilidad con división en 16 cuadrantes	Problemática si no es múltiplo de 16	Buena (cuadrantes de 125×125)	Excelente (cuadrantes de 128×128)
Compatibilidad con interpolación bilineal	Depende de las dimensiones exactas	Problemática (125 no es divisible por 4)	Excelente (128 es divisible por 4)
Utilización de memoria	Variable	Media	Media-Alta
Preservación de información original	Total	Parcial (requiere redimensionamiento)	Parcial (requiere redimensionamiento)
Simplicidad de implementación	Baja (manejo de dimensiones irregulares)	Media	Alta

Cuadro 3: Comparación de opciones de dimensionamiento de imágenes

El análisis revela que aunque mantener las dimensiones originales de la imagen preservaría todos los detalles, podría complicar significativamente la implementación del algoritmo si las dimensiones no son adecuadas para la división en cuadrantes y la aplicación de la interpolación bilineal.

4.4. Evaluación de estrategias para la organización de píxeles

La forma de organizar los píxeles en memoria y en los archivos de entrada/salida afecta directamente a la complejidad del código ensamblador. La Tabla 4 presenta la comparación entre las estrategias consideradas.

Criterio	**Organización lineal**	**Organización por bloques (2×2 entrada, 4×4 salida)**
Simplicidad de almacenamiento	Alta	Media
Complejidad de acceso a píxeles adyacentes	Alta (requiere cálculos de índices)	Baja (naturalmente agrupados)
Compatibilidad con el algoritmo de interpolación	Baja (necesita reorganización)	Alta (alineada con la lógica del algoritmo)
Eficiencia de procesamiento en ensamblador	Baja (múltiples cálculos de direcciones)	Alta (acceso secuencial a bloques)
Complejidad del código ensamblador	Alta	Media

Cuadro 4: Comparación de estrategias para la organización de píxeles

Como señala [3], la organización de datos en memoria es crucial para la eficiencia del código ensamblador. La organización lineal de píxeles simplifica el almacenamiento pero complica significativamente el algoritmo de interpolación bilineal, que requiere acceso a bloques de 2×2 píxeles.

En contraste, la organización por bloques (2×2 para entrada y 4×4 para salida) alinea perfectamente la estructura de datos con la lógica del algoritmo, permitiendo un procesamiento secuencial de los bloques. [2] enfatizan que este tipo de organización de datos orientada al algoritmo puede reducir drásticamente la complejidad del código, especialmente en lenguajes de bajo nivel como el ensamblador.

5. Propuesta Final

Tras evaluar exhaustivamente las distintas alternativas y sus implicaciones técnicas, se presenta a continuación la propuesta definitiva para la implementación del sistema de interpolación bilineal en lenguaje ensamblador. Esta selección representa la solución más equilibrada y eficiente considerando los requerimientos establecidos y las restricciones inherentes a la programación en bajo nivel.

5.1. Solución seleccionada

La propuesta final comprende una combinación específica de tecnologías y enfoques que abordan de manera óptima los desafíos del proyecto. La Tabla 5 presenta un resumen de las opciones seleccionadas para cada componente del sistema.

Componente	**Selección Final**	**Justificación Principal**
Lenguaje de alto nivel	Python con librerías tkinter, PIL y numpy	Flexibilidad e integración
Arquitectura ISA	x86-64 de 64 bits con NASM	Capacidades matemáticas CISC
Tamaño de imagen	512×512 píxeles	Divisibilidad óptima (16 y 4)
Organización de píxeles entrada	Bloques de 2×2 píxeles	Alineación con algoritmo
Organización de píxeles salida	Bloques de 4×4 píxeles	Simplifica código ensamblador
Sistema operativo	Linux/Ubuntu	Facilidad para NASM y scripts

Cuadro 5: Componentes seleccionados para la implementación final

5.2. Descripción del flujo de trabajo implementado

El sistema implementado sigue un flujo de trabajo estructurado y eficiente que maximiza las ventajas de cada componente seleccionado:

1. **Selección y preparación de la imagen:** El programa inicia en Python, permitiendo al usuario seleccionar una imagen JPEG mediante una interfaz gráfica desarrollada con tkinter. Independientemente del tamaño original, la imagen se redimensiona automáticamente a 512×512 píxeles.

2. **División en cuadrantes y selección:** La imagen se divide en 16 cuadrantes (4×4), cada uno de 128×128 píxeles. El usuario selecciona uno de estos cuadrantes a través de la interfaz gráfica.

3. **Generación del archivo de entrada:** El cuadrante seleccionado se convierte a escala de grises (valores de 0 a 255) utilizando numpy y PIL. Estos valores se escriben en un archivo binario `input.img`, organizados en bloques de 2×2 píxeles.

4. **Procesamiento en ensamblador:** Se ejecuta automáticamente un script `run_output.sh` que compila y ejecuta el código ensamblador `output_gen.asm`. Este código lee el archivo `input.img`, implementa el algoritmo de interpolación bilineal y genera un archivo `output.img` con los resultados organizados en bloques de 4×4 píxeles.

5. **Visualización del resultado:** Python lee el archivo `output.img` generado por el ensamblador, interpreta correctamente la organización de píxeles y muestra la imagen resultante de 256×256 píxeles en la interfaz gráfica.

Esta estructura de flujo garantiza que todo el procesamiento matemático de la interpolación bilineal se realice exclusivamente en ensamblador, mientras que Python se encarga únicamente de la interfaz de usuario y la visualización.

5.3. Justificación técnica de la solución seleccionada

La combinación específica de tecnologías y enfoques metodológicos seleccionada ofrece ventajas significativas para este proyecto particular:

Python como lenguaje integrador: La selección de Python responde a su excepcional capacidad para crear un framework unificado. Como señala [1], Python ha emergido como estándar de facto para aplicaciones de procesamiento de imágenes debido a su amplio ecosistema de bibliotecas especializadas. Su naturaleza interpretada facilita la invocación de programas

externos mediante subprocesos, característica fundamental para la integración con el compilador NASM. Las bibliotecas tkinter para interfaces gráficas, PIL para manipulación de imágenes y numpy para operaciones matriciales proporcionan un entorno completo para las operaciones de alto nivel sin necesidad de recurrir a software propietario.

x86-64 como arquitectura de implementación: La elección de x86-64 aprovecha las ventajas inherentes a su naturaleza CISC (Complex Instruction Set Computing). [4] destacan que las arquitecturas CISC ofrecen instrucciones matemáticas complejas que pueden ejecutar operaciones en un solo paso, lo que resulta particularmente beneficioso para algoritmos como la interpolación bilineal que requieren divisiones (especialmente divisiones por 3 para calcular ponderaciones de 1/3 y 2/3). La disponibilidad del compilador NASM en entornos Linux facilita la implementación y pruebas del código ensamblador.

Dimensiones de 512×512 píxeles: La elección de este tamaño específico no es arbitraria, sino que responde a consideraciones matemáticas precisas. Un análisis inicial con imágenes de 500×500 píxeles reveló que, aunque este tamaño es divisible por 16 (para obtener cuadrantes), los cuadrantes resultantes de 125×125 no son divisibles por 4, lo que genera complicaciones en la implementación del algoritmo de interpolación. Como argumenta [2], la alineación de las estructuras de datos con las operaciones algorítmicas es fundamental para la eficiencia computacional. Las dimensiones de 512×512 garantizan que:

- La imagen sea divisible en 16 cuadrantes iguales (4×4) de 128×128 píxeles
- Cada cuadrante sea divisible por 4, permitiendo una perfecta implementación del algoritmo de interpolación
- La imagen resultante de la interpolación (256×256) mantenga la divisibilidad por 16, facilitando la organización por bloques de 4×4 píxeles

Organización de píxeles por bloques: La decisión de organizar los píxeles en bloques (2×2 para entrada, 4×4 para salida) constituye una innovación metodológica que simplifica dramáticamente la implementación en ensamblador. [3] enfatiza que la complejidad en programación en ensamblador frecuentemente surge no del algoritmo en sí, sino de las operaciones de acceso a memoria y cálculo de direcciones.

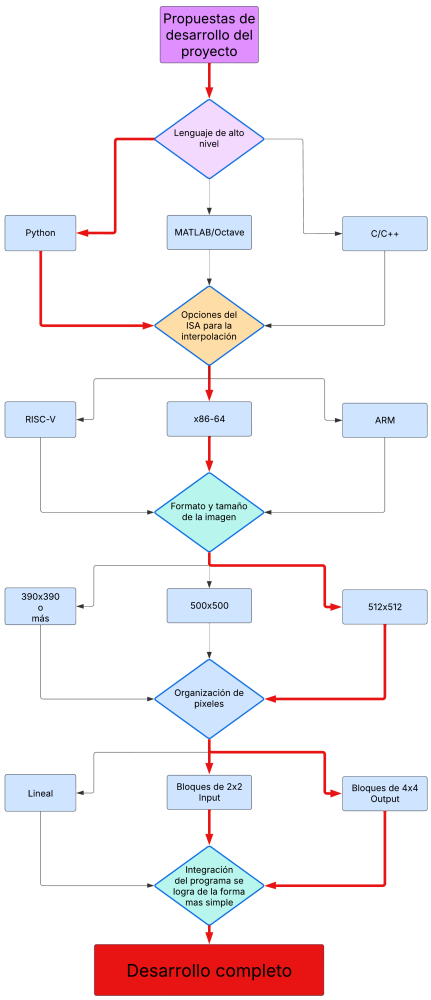

Figura 2: Diagrama de flujo del sistema de interpolación bilineal implementado

Al estructurar los datos en bloques que se corresponden naturalmente con la organización lógica del algoritmo, se evita la necesidad de implementar complejas rutinas de salto en el código ensamblador. Python asume la responsabilidad de la interpretación espacial de los datos, mientras que el código ensamblador puede procesar secuencialmente los bloques sin preocuparse por la organización bidimensional de la imagen.

5.4. Diagrama de flujo de la solución implementada

La Figura 2 ilustra el flujo completo del sistema, mostrando la interacción entre los componentes de alto nivel (Python) y bajo nivel (ensamblador x86-64), así como la transformación de los datos a lo largo del proceso.

5.5. Conclusiones sobre la propuesta

La propuesta final representa una solución integrada que equilibra eficientemente los requisitos técnicos con las limitaciones prácticas de la programación en ensamblador. El enfoque adoptado demuestra que la combinación estratégica de lenguajes de alto y bajo nivel puede producir sistemas que aprovechan lo mejor de ambos mundos: la versatilidad y facilidad de desarrollo de Python con el rendimiento y control preciso del ensamblador x86-64.

La organización de datos en bloques estructurados constituye una contribución metodológica significativa, mostrando cómo el diseño cuidadoso de las estructuras de datos puede simplificar dramáticamente la implementación de algoritmos complejos en lenguajes de bajo nivel. Como señala [8] al analizar la Ley de Amdahl, las optimizaciones más efectivas suelen provenir no de mejoras incrementales en secciones específicas del código, sino de transformaciones estructurales en la organización y flujo de los datos.

El sistema implementado cumple con todos los requerimientos establecidos, realizando el procesamiento de interpolación bilineal exclusivamente en ensamblador mientras proporciona una interfaz amigable para el usuario. La división en componentes claramente definidos (interfaz, preprocesamiento, procesamiento en ensamblador y visualización) facilita el mantenimiento y posibles extensiones futuras del sistema.

Referencias

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. Morgan Kaufmann, 2017.

[3] S. P. Dandamudi, *Guide to Assembly Language Programming in Linux*. Springer, 2005.

[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Morgan Kaufmann, Elsevier Inc., 2012.

[5] S. L. Harris and D. M. Harris, *Digital Design and Computer Architecture, ARM® Edition*. Morgan Kaufmann, Elsevier Inc., 2016.

[6] P. R. Srivastava, Y. Singh, and A. K. Verma, *Software Engineering and Testing*. BPB Publications, 2009.

[7] D. Barbier, "Sifive 2 series risc-v core ip - webinar may 15," 2019, presentación disponible en https://info.sifive.com/risc-v-second-webinar-series.

[8] A. Kumari, "An analytical study of amdahl's and gustafson's law," 2019, electronic copy available at: https://ssrn.com/abstract=3435202.