UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

AMPLIACIÓN MEJORADA DE IMÁGENES DIGITALES POR COMPUTADORA

TRABAJO DE GRADUACIÓN

PRESENTADO POR

Karlos Manuel Portillo

PARA OPTAR AL GRADO DE

Ingeniero en Ciencias de la Computación

Y POR

Jorge Alonso López Salazar

PARA OPTAR AL GRADO DE

Ingeniero en Electrónica

CIUDADELA DON BOSCO, MARZO DE 2003

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

ESCUELA DE ELECTRÓNICA

ESCUELA DE COMPUTACIÓN

AMPLIACIÓN MEJORADA DE IMÁGENES DIGITALES POR COMPUTADORA

Ing. Oscar Durán Vizcarra

Asesor

Ing. Eduardo Rivera

Jurado

Ing. Karim Heredia

Jurado

CIUDADELA DON BOSCO, MARZO DE 2003

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

ESCUELA DE ELECTRÓNICA

ESCUELA DE COMPUTACIÓN

AMPLIACIÓN MEJORADA DE IMÁGENES DIGITALES POR COMPUTADORA

Ing. Federico Huget

Rector

Lic. Mario Olmos

Secretario General

Pbro. Víctor Bermúdez

Vicerrector Académico

Ing. Carlos Bran

Decano Facultad de Ingeniería

CIUDADELA DON BOSCO, MARZO DE 2003

Por mi familia, por mi compañera de viaje, por mis amigos, y por todas las buenas intenciones, acciones y sentimientos de los que he sido objeto por parte de todos ellos...

Por las dificultades, por las oportunidades, por la sucesión ininterrumpida de días vividos (y los que restan)...

Por las ideas, las ilusiones, las esperanzas, los triunfos y los fracasos, las experiencias y el conocimiento adquirido...

Por la culminación exitosa de esta empresa, por los retos venideros, y por otras tantas cosas sabiamente elegidas y colocadas en el transcurso de mi diario vivir, necesarias para la evolución de mi ser...

Por tu presencia...

Gracias, Señor.

Jorge Alonso López Salazar

Gracias a la vida que me ha dado tanto, me ha dado cariño, apoyo, amistad, lazos fuertes que por ningún motivo a través del tiempo serán rotos.

Gracias, por haberme hecho llegar a un hogar donde pude encontrar paz y felicidad, gracias por darme una madre que ha sido la piedra angular en mi total desarrollo, dos abuelos con un corazón enorme que reboza de amor hacia mí, y el cual es correspondido desde lo mas profundo de mi ser.

Gracias, por darme amigos que me han apoyado en todo momento, en las distintas fases de mi vida, esperando poder algún día, retribuir toda la ayuda que ellos sin interés alguno me han brindado.

La vida me ha dado muchas cosas, triunfos y fracasos, alegrías y tristezas, pruebas que me han hecho ser más fuerte, que me ayudan a seguir adelante, dichas pruebas solo han sido superadas gracias al apoyo del que me ha dado todo, empezando por ella misma: *la vida*, por ende todo lo que he mencionado antes, se lo debo a Él.

Por eso y por todas las demás cosas por venir

Gracias Señor, nuestro Dios.

Karlos Manuel Portillo.

UNIVERSIDAD DON BOSCO
FACULTAD DE INGENIERÍA

AMPLIACIÓN MEJORADA DE IMÁGENES DIGITALES POR COMPUTADORA

TRABAJO DE GRADUACIÓN

PRESENTADO POR

Karlos Manuel Portillo

PARA OPTAR AL GRADO DE

Ingeniero en Ciencias de la Computación

Y POR

Jorge Alonso López Salazar

PARA OPTAR AL GRADO DE

Ingeniero en Electrónica

CIUDADELA DON BOSCO, MARZO DE 2003

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE LISTADOS DE CÓDIGO FUENTE

1. INTRODUCCIÓN

Desde principios de la historia, el hombre se ha tropezado con problemas que lo han desconcertado, motivándolo a encontrar, en la medida de lo posible, soluciones a cada uno de ellos.

Es por ello que a partir de grandes descubrimientos realizados por filósofos, físicos, matemáticos y estudiosos de tantas otras áreas de la ciencia, el hombre ha sentado las bases para el desarrollo de técnicas capaces de solucionar problemas ínfimos o proponer teorías que han cautivado por mucho tiempo, como por ejemplo: descifrar el genoma humano. Todos estos hechos marcan cómo el hombre ha ido avanzando día con día en su lucha por comprender los fenómenos que acontecen en el mundo que le rodea.

Esta evolución se muestra más acentuada en los últimos 100 años de la existencia humana, siendo un hecho trascendental la invención del transistor, lo que constituyó el punto de partida para gran parte del desarrollo tecnológico que el mundo está experimentando hoy en día.

En el campo del procesamiento de imágenes, dicha evolución ha sido favorecida por el auge de las computadoras digitales, las cuales han permitido la aparición de múltiples herramientas útiles en diversas áreas, como en la digitalización, compresión, edición, e impresión de imágenes.

Siguiendo la misma línea de desarrollo, se desea encontrar una solución al problema de ampliar una imagen sin perder la calidad de la misma; algo que no ha sido logrado satisfactoriamente en muchas de las aplicaciones de procesamiento de imágenes disponibles comercialmente.

Por esta razón y en vista de que el procesamiento digital de imágenes ha adquirido, en años recientes, un papel importante en las

tecnologías de la información y de cómputo, y que actualmente, es la base de una creciente variedad de aplicaciones que incluyen diagnosis médica, percepción remota, exploración espacial, visión por computadora, etc., en este Trabajo de Graduación se tratará de proporcionar una solución aceptable al mencionado problema.

Dicha solución se apoyará en una serie de conocimientos matemáticos, los cuales serán aplicados para construir diversos algoritmos para el ampliado de imágenes digitales.

Los procesos matemáticos a realizar estarán basados en la aproximación de funciones mediante el uso de diferentes tipos de interpolación, como por ejemplo: interpolación lineal, polinomial e interpolación cúbica segmentaria. Partiendo de los datos de la imagen de origen y aplicando dichos tipos de interpolación se tratará de deducir todos los demás datos que conformarán la imagen ampliada.

Para que el usuario pueda generar y apreciar la nueva imagen, se le proporcionará una herramienta de fácil manejo, desarrollada en una aplicación orientada a objetos, teniendo a la mano todos los elementos necesarios para ampliar la imagen, no siendo necesario el preocuparse por los complicados cálculos que se realizarán en la parte trasera del proceso.

Al hecho de fusionar dichas técnicas matemáticas y procesos efectuados por computadora, para así lograr descubrir o hacer resaltar información contenida en una imagen se le conoce como Procesamiento Digital de Imágenes (PDI).

El PDI tiene múltiples intereses de estudio, los cuales abarcan una gran cantidad de áreas de la ciencia y la técnica; pero para los fines de este Trabajo de Graduación se enfoca a solamente uno, el cual es:

El mejoramiento de la calidad de la información contenida en una imagen digital ampliada con el fin de que pueda ser interpretada más fácilmente por el observador.

Dicha información se obtendrá a partir del desarrollo de un sistema computacional, el cual será descrito en los capítulos posteriores.

2. OBJETIVOS

2.1. Objetivo General

Estudiar, utilizar y probar diversos métodos para realizar la ampliación de una imagen, fundamentados en la investigación de diferentes formas de inferir información faltante en la misma, e implementarlos en una aplicación capaz de interactuar con los procesos matemáticos involucrados, presentando el resultado en un ambiente gráfico y fácil de utilizar, proporcionando un mejor resultado que con los métodos tradicionales de ampliación, y tratando de conservar en la medida de lo posible la misma resolución de la imagen original.

2.2. Objetivos Específicos

- Realizar una investigación acerca de cómo inferir información faltante en una imagen ampliada, partiendo de los datos previamente establecidos por la imagen original, determinando con estos valores funciones capaces de representar cualquier punto en la imagen para cada fila o columna de datos específica.
- Efectuar pruebas de evaluación de los métodos de ampliado, con el fin de conocer cuál de ellos proporciona mejores resultados y en cuáles circunstancias.
- Desarrollar una aplicación capaz de presentar al usuario una imagen ampliada con una mejor resolución que la que obtendría con los métodos tradicionales de ampliación. Dicha aplicación se encargará de interactuar con las operaciones matemáticas, procesar la información y devolver como resultado la imagen ampliada.

3. ALCANCES Y LIMITACIONES

3.1. Alcances

Debido a que el proyecto contempla un alto grado de investigación, el resultado final es una serie de métodos con los cuales se pretende mejorar el proceso de ampliación que actualmente se utiliza. Dichos métodos están basados en operaciones matemáticas, uso de algoritmos y evaluación de polinomios, cuyo objetivo es proporcionar una imagen ampliada basada en la estimación de datos a partir de los puntos originales. Cada uno de los métodos será evaluado para determinar cuál o cuáles son los más adecuados, basados en criterios de tiempo de ejecución del proceso, calidad de la imagen ampliada y utilización de los recursos del sistema.

El producto final obtenido será un sistema que permita generar una imagen ampliada a partir de una imagen digital original, y que presente un mayor grado de detalle que ésta, utilizando cualquiera de los métodos de ampliación desarrollados. Además del análisis hecho en el proceso de desarrollo, el usuario final será capaz de determinar cuál a su parecer es el mejor método sobre la base de su propio criterio. Las imágenes susceptibles de ser ampliadas serán tanto en escala de grises, como en color en formato RGB, ambas en formatos JPEG o de mapa de bits de Windows (BMP). EL usuario podrá interactuar fácilmente con la aplicación debido a que será desarrollada en un ambiente gráfico en el cual no necesita más que el ratón de la computadora para llevar a cabo fácilmente todo el proceso de ampliado. Dicha aplicación interactuará con otra herramienta encargada de realizar los procesos matemáticos, devolviendo posteriormente los resultados obtenidos a la aplicación gráfica para presentar al usuario la imagen ampliada.

Básicamente, se dispondrá de dos herramientas en la aplicación final: el zoom in (acercamiento o ampliado), y el zoom out (alejamiento o restablecimiento de las dimensiones anteriores).

Las ampliaciones sucesivas de una misma imagen podrán realizarse en dos modalidades, dependiendo de si se toma como imagen base a la imagen de dimensiones originales, o a la última ampliación realizada de la misma.

3.2. Limitaciones

- La cantidad máxima de ampliación de una imagen depende de las capacidades físicas de la computadora (RAM, microprocesador) en la que se esté ejecutando la aplicación.
- El producto final es un zoom y no un filtro, por lo que no debe esperarse obtener un amplio grado de detalle en imágenes que no contienen suficiente información para poder analizarse; por ejemplo, fotografías tomadas a una gran distancia, capturadas en áreas de poca iluminación, en movimiento, etc. En otras palabras, para toda imagen que no presente un mínimo de calidad en la información no podrá obtenerse el detalle esperado.
- Sólo podrán ampliarse imágenes digitales de mapa de bits, y las comprimidas según el formato JPEG, puesto que el proceso requiere la representación de las mismas como un arreglo bidimensional de píxeles, cosa que no ocurre con las imágenes de tipo vectorial.
- Las imágenes soportadas por el sistema deberán ser de un byte por píxel en escala de grises, así como de tres bytes por píxel en color, según el formato RGB.
- Las imágenes a ser ampliadas no pueden ser de dimensiones elevadas, debido a que el alto contenido de información de éstas conlleva a realizar una gran cantidad de operaciones durante la ampliación, ocasionando que el proceso sea prohibitivamente lento.

4. MARCO TEÓRICO

En este capítulo se abordará la teoría básica necesaria para la comprensión de ciertos tópicos discutidos a lo largo del documento, tales como *imágenes digitales* e *interpolación*. Si el lector está familiarizado con lo expuesto en esta parte, puede omitir la lectura de este capítulo sin riesgo a la pérdida de continuidad.

4.1. Imágenes

Una *imagen* es una representación óptica de un objeto iluminado por una fuente radiante. En el proceso de formación de una imagen participan los siguientes elementos:

- Un objeto, o cuerpo del cual se desea capturar su imagen;
- la fuente de radiación y
- el sistema de formación de la imagen.

La fuente de radiación genera ondas electromagnéticas (por ejemplo luz visible o rayos X) que inciden sobre el objeto. Parte de esta energía incidente es absorbida por el cuerpo, mientras que el resto se refleja, en principio, en todas direcciones. Las características de las ondas absorbidas y reflejadas por el cuerpo dependen de las características físicas de éste, a nivel atómico, y de las propiedades de las ondas incidentes sobre él. Así por ejemplo, un objeto en particular podría reflejar ondas de una frecuencia específica al incidir sobre él un tipo de radiación determinada, mientras que un segundo cuerpo reflejaría ondas de una frecuencia diferente, ante el mismo tipo de radiación[1].

Las ondas reflejadas contienen entonces información acerca del cuerpo sobre el cual incidieron, y son éstas (o sus características) las

[1] Éste es el principio de formación de los colores.

que deben ser registradas de alguna forma. Cuando dichas ondas alcanzan al sistema adquisidor, alteran las propiedades de algún componente de éste en alguna manera, en función de la información proveniente de la imagen. Para el caso de imágenes "analógicas" — las generadas por las cámaras fotográficas tradicionales, por ejemplo, el compuesto químico de la película reacciona con la radiación incidente (generalmente luz visible), formando así las diversas regiones de claridad, oscuridad y/o color presentes en la imagen revelada. Por otro lado, para la formación de imágenes digitales se hace uso de sensores, cuyo trabajo consiste en la generación de señales eléctricas como resultado de haber sido excitados por la incidencia de la luz.

4.1.1. Captura de imágenes digitales

La formación de la imagen digital es el primer paso en cualquier aplicación de procesamiento de imágenes digitales. El sistema de formación de imágenes digitales consiste básicamente de un sistema óptico, de un sensor y de un digitalizador[2] (Figura 4.1), los cuales se describen a continuación.

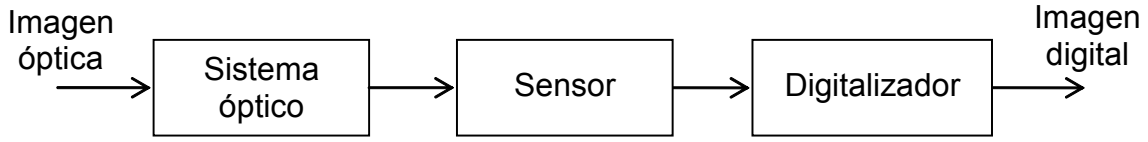


Figura 4.1. Modelo de un sistema de formación de imágenes digitales.

- Sistema óptico (o simplemente óptica): Se conoce con este nombre al arreglo de lentes utilizados para concentrar la radiación proveniente del objeto, y hacerla incidir sobre los sensores. Comúnmente, la óptica se comporta como un sistema *pasa bajos*, por lo que suprime el contenido de alta frecuencia de la imagen de

[2] La *iluminación* también forma parte de los sistemas adquisidores. Sin embargo, no se aborda en este documento.

entrada. Por lo tanto, la imagen de salida se vuelve una versión un poco borrosa o desenfocada de la imagen original.

- Sensores: Son los dispositivos utilizados para captar la información luminosa de la escena, y convertirla en señales eléctricas. Existen varios tipos de sensores; por ejemplo los tubos vídicon estándar, los dispositivos de inyección de carga (del inglés, Charge Injection Devices) y los dispositivos de carga acoplada (del inglés Coupled Charge Devices). En muchos casos, la relación entre la imagen de entrada y el parámetro eléctrico de salida es altamente no lineal.
- Digitalizador: La salida del dispositivo sensor es todavía una señal análoga que debe ser muestreada y digitalizada antes de que pueda ser procesada por la computadora. El muestreo y la digitalización se realizan por medio de un convertidor analógico–digital (A/D).

4.1.2. Representación de imágenes digitales

Generalmente, los sensores en el sistema adquisidor de imágenes están espacialmente dispuestos formando una matriz rectangular. Cada uno de los sensores es el encargado de recibir la radiación luminosa correspondiente a una región del espacio en particular, y por medio del convertidor A/D, generar un número que represente la información recibida. Dicha información puede ser la intensidad de luz en ese punto (nivel de gris, también conocido como *luminancia*) o su color (*crominancia*), en cuyo caso se registran tres números (utilizando tres sensores por punto de captura), uno para cada color primario[3], según el formato RGB (del inglés, Red–Green–Blue, o rojo–verde–azul). Dado que, independientemente de qué tan cercanos se encuentren espacialmente los sensores en la matriz, siempre existirá algún espacio entre los sensores en el que incidirán rayos de luz que no podrán ser registrados, se dice entonces que la imagen capturada sufre un

[3] Los colores primarios *aditivos* son el rojo, el verde y el azul.

muestreo espacial. Esto conlleva al concepto de *resolución*, que puede entenderse como la cantidad de puntos de captura de información por unidad de área, o, en otras palabras, la densidad espacial de la información de la imagen. Mientras mayor sea la resolución de una imagen, mayor será la información contenida en ésta, comparada con otra imagen de dimensiones idénticas, y, por lo tanto, será mejor apreciada por el observador[4]. La resolución comúnmente se mide en puntos por pulgada (*dpi*, del inglés dots–per–inch) o *ppi* (del inglés, pixels–per–inch, o píxeles por pulgada).

Cada posición o celda de la matriz de sensores se conoce con el nombre de píxel (del inglés, *picture element*). Para cada píxel, se debe registrar la siguiente información:

a. Su ubicación espacial en el plano que forma la imagen, y

b. la información contenida en dicha posición (nivel de gris o color).

El almacenamiento de dicha información da origen a lo que se conoce como imágenes de mapas de bits[5]. En dichas imágenes, la información de luminancia o crominancia para cada píxel se almacena en un archivo, y la posición que ocupan estos datos dentro del mismo coincide (o está relacionada) con la ubicación espacial de los píxeles dentro de la imagen. De esta manera, la información registrada por los sensores se encuentra totalmente definida en el archivo. La cantidad de bits con la que se almacena esta información da origen a diversos tipos de imagen de mapa de bits. Así, por ejemplo, se cuenta con los siguientes tipos:

---

[4] Sin embargo, más allá de ciertos límites, es difícil que el ojo humano note la diferencia.

[5] Existe otro tipo de imágenes, las *vectoriales*, pero no se abordan en este documento por las razones que se exponen en la Sección 3.2 Limitaciones.

- Imágenes de 1 bit por píxel. En este tipo de imágenes cada píxel sólo puede tener uno de dos valores: uno o cero. Como basta 1 bit para definir esa alternativa, se les llama "imágenes de 1 bit" (también conocidas como "imágenes de mapa de bits de alto contraste, de línea, o en blanco y negro"). La información en este caso se limita a indicar si un píxel es blanco (su bit es 1) o negro (su bit es 0). En la Figura 4.2 se muestra un ejemplo de imagen de este tipo.

Figura 4.2. Imagen de 1 bit por píxel.

- Imágenes de escala de grises (8 bits por píxel). En este caso, cada píxel puede tener 256 valores diferentes (las 256 posibilidades combinatorias de un byte u octeto de bits), siendo el 0 el valor correspondiente a un píxel no iluminado (negro), y el 255 a uno totalmente iluminado (blanco). Los valores intermedios representan linealmente diferentes tonos de gris. La Figura 4.3 ejemplifica este tipo de imagen.

Figura 4.3. Imagen de escala de grises.

- Imágenes RGB (24 bits por píxel). Éstas son imágenes a color, en donde dicho color se obtiene de la combinación ponderada de los tres colores primarios (rojo, verde y azul). Cada color primario se almacena con un número binario de 8 bits (con valores desde 0 hasta 255), indicando la intensidad de la componente de color respectiva. Un valor de 0 indica la ausencia de dicho color, mientras que 255 indica su total presencia. El color blanco se obtiene con el máximo valor de las componentes de color (todas en 255), mientras que el negro con todas en su valor mínimo (0). Este sistema es el empleado por las pantallas de televisión y los monitores para generar los colores, por medio de un cañón de haces de luz para cada una de las componentes. Con este sistema, un píxel puede adquirir uno de $16_1270{,}216$ colores posibles (256 x 256 x 256). A continuación se presenta un ejemplo de este tipo de imagen.

Figura 4.4. Imagen RGB.

- Imágenes CMYK (32 bits por píxel). Si a cada píxel se le asignan 4 bytes, se podría representar (teóricamente) los valores de CMYK (del inglés, cyan –cian, magenta –magenta, yellow –amarillo, y black –negro), propios de la cuatricromía profesional: 1 byte para el cian, uno para el magenta, otro para el amarillo, y un último para el negro; todos ellos en el rango de 0 a 255, y con un significado idéntico a RGB para dichos valores. Sin embargo, dado que los monitores de computadora solamente pueden generar colores mediante el formato RGB, la calidad de color de las imágenes CMYK sólo puede ser apreciada en imágenes impresas.

Una imagen digital de mapa de bits puede representarse matemáticamente como una función $f(x,y)=z$ de dos variables independientes ("x" e "y") y una variable dependiente ("z"). El par (x,y) representa la posición de fila y columna, respectivamente, de cada píxel de la imagen, mientras que a la variable "z" la conforma la luminancia o crominancia de dicho punto en la misma. Estas variables son de naturaleza discreta, puesto que la imagen consta de un número finito de píxeles, y la luminancia y crominancia de éstos se encuentra cuantificada, dado que se representan con un conjunto finito de números enteros (por ejemplo, de 0 a 255, como ya se discutió antes). Significa entonces que dicha función puede modelarse como una matriz de dos dimensiones (para el caso de imágenes en escala de grises) o

de tres dimensiones (si se trata de imágenes en color RGB), como se muestra a continuación.

$$f(x,y)=\begin{bmatrix} f(1,1)=z_{11} & f(1,2)=z_{12} & f(1,3)=z_{13} & \cdots & f(1,m)=z_{1m} \\ f(2,1)=z_{21} & f(2,2)=z_{22} & f(2,3)=z_{23} & \cdots & f(2,m)=z_{2m} \\ f(3,1)=z_{31} & f(3,2)=z_{32} & f(3,3)=z_{33} & \cdots & f(3,m)=z_{3m} \\ \vdots & \vdots & \vdots & \cdots & \\ f(n,1)=z_{n1} & f(n,2)=z_{n2} & f(n,3)=z_{n3} & \cdots & f(n,m)=z_{nm} \end{bmatrix}$$

(4.1)

En donde *n* es el número de filas de la imagen, y *M* es el número de columnas. Los valores de *z* pueden ser tanto de luminancia, si se trata de imágenes en escala de grises; y de crominancia, para imágenes a color RGB. En este último caso, *z* está conformada por un vector fila de 3 dimensiones, conteniendo los valores de las tres componentes de color (rojo, verde y azul). Por tanto, la matriz que representa a una imagen es de orden $n \times m \times p$, siendo *p* = 1 si se trata de una imagen en escala de grises, y *p* = 3, para una imagen a color.

4.2. Ajuste de curvas[6]

A menudo se proporcionan datos mediante un conjunto de puntos discretos. Sin embargo, a veces se requieren estimaciones de puntos entre esos valores discretos. Por otro lado, en otras ocasiones lo que se busca es una versión simplificada de una función muy complicada. Una manera de hacer esto último es calculando valores de la función en un conjunto de valores discretos a lo largo del rango de interés; después se puede obtener una función más simple con dichos valores. A estas dos aplicaciones se les conoce con el nombre de *ajuste de curvas*. Para nuestro caso particular, el interés se centra en la primera de las aplicaciones mencionadas, puesto que lo que se pretende es la

[6] Adaptado de [1].

estimación de los valores de los píxeles faltantes en una imagen dada, con objeto de ampliarla.

Hay dos esquemas generales en el ajuste de curvas que se distinguen entre sí sobre la base de la cantidad de error asociada con los datos. Primero, donde los datos muestran un grado significativo de error o "ruido", la estrategia es derivar una curva simple que represente el comportamiento general de los datos. Ya que cada punto individual puede estar incorrecto, no es necesario (o deseable) intersecar cada uno de ellos. En vez de esto, la curva se diseña de tal manera que siga un patrón sobre los puntos tomados como un todo. A un procedimiento de esta naturaleza se le conoce con el nombre de *regresión*. Segundo, donde se conoce que los datos son muy exactos, el proceso a seguir es el de ajustar una curva o una serie de curvas que pasen exactamente por cada uno de los puntos. Algunos ejemplos son los valores de la densidad del agua y de la capacidad de calor de los gases como una función de la temperatura. A la estimación de valores entre puntos discretos conocidos se le conoce con el nombre de *interpolación*. Para ambas estrategias existen métodos matemáticos muy definidos, los cuales presentan ciertas ventajas y desventajas particulares, por lo que la idoneidad de su uso depende del tipo de problema de que se trate.

Los métodos se dividen en dos amplias categorías dependiendo de la incertidumbre de los datos. Para las mediciones imprecisas, se usa la regresión para desarrollar la "mejor" curva que ajuste todas las tendencias de los datos sin pasar necesariamente a través de algún punto. Para mediciones precisas, se usa la interpolación para desarrollar una curva que pase directamente a través de cada uno de los puntos.

Todos los métodos de regresión se diseñan de manera que ajusten funciones que minimicen la suma de los cuadrados de los residuos entre los datos y la función (regresión con mínimos cuadrados). La

regresión con mínimos cuadrados lineal se usa en aquellos casos donde una variable dependiente y otra independiente se relacionan de manera lineal. Para situaciones en que las variables dependiente e independiente muestren una relación curvilínea, se dispone de varias alternativas. En algunos casos, se pueden usar transformaciones para "linealizar" la relación. En estos casos se puede aplicar la regresión lineal a variables transformadas para determinar la mejor línea recta. Alternativamente, se puede emplear la regresión polinomial y ajustar una curva directamente a los datos.

La regresión lineal múltiple se usa cuando una variable dependiente es una función de dos o más variables independientes. Se pueden aplicar también transformaciones logarítmicas a este tipo de regresión en algunos casos donde la dependencia múltiple es curvilínea.

La interpolación polinomial está diseñada para ajustar un polinomio único de n-ésimo orden que pase exactamente por los n + 1 puntos. Este polinomio se presenta en dos formatos diferentes. El polinomio de interpolación de diferencias divididas de Newton se adapta idealmente a aquellos casos en que el orden propio del polinomio se desconoce. El polinomio de Newton es apropiado para tales situaciones, ya que se programa fácilmente en un formato que compara los resultados con órdenes diferentes. Además, se puede incorporar con facilidad una aproximación del error en el método. De esta forma, se puede comparar y escoger a partir de los resultados usando varios polinomios de órdenes diferentes.

La otra formulación alternativa es el polinomio de interpolación de Lagrange, el cual es apropiado cuando el orden se conoce a priori. En estos casos, la versión de Lagrange es algo más simple de programar y no requiere de los cálculos y almacenamiento de diferencias divididas finitas.

El otro método de ajuste de curvas es mediante interpolación segmentaria. Este método ajusta un polinomio de orden bajo a cada uno de los intervalos entre los puntos. El ajuste se hace uniforme obligando a que las derivadas de dos polinomios adyacentes en el mismo valor de su punto de conexión sean iguales. La interpolación cúbica segmentaria (también conocida como funciones *spline*) es la versión más común. Los segmentos son muy útiles cuando se ajustan datos que en general son uniformes pero exhiben áreas locales de saltos de los datos. Tales datos tienden a inducir oscilaciones en los polinomios de interpolación de orden superior. La interpolación cúbica segmentaria está menos propensa a estas oscilaciones ya que se limita a variaciones de tercer orden. Este tipo de interpolación tiene dos variantes principales, conocidas comúnmente como c–splines y b–splines. La primera de ellas es una interpolación estrictamente hablando, puesto que las funciones obtenidas pasan exactamente por todos los puntos dados. Sin embargo, con b–splines se tiene un híbrido de ambas estrategias de ajuste de curvas (interpolación y regresión), aprovechando las ventajas de ambos métodos. Así, por ejemplo, b–splines cuenta con los beneficios que implica el uso de segmentos de funciones cúbicas, tales como la disminución de oscilaciones cerca de cambios bruscos en los datos; mientras que presenta además cierta inmunidad ante datos ruidosos o erróneos, al igual que los métodos de regresión. En la Tabla 4.1 se proporciona un resumen de los elementos de juicio relacionados con el ajuste de curvas.

Tabla 4.1. Comparación de las características de los diversos métodos en el ajuste de curvas

Método	Error asociado con los datos	Coincidencia con cada uno de los puntos	Número de puntos con los que coincide exactamente	Esfuerzo de programación	Cometarios
Regresión					
Regresión lineal	Grande	Aproximada	0	Pequeño	
Regresión polinomial	Grande	Aproximada	0	Moderado	Los errores de redondeo se hacen notorios para versiones de orden superior
Regresión lineal múltiple	Grande	Aproximada	0	Moderado	
Interpolación					
Polinomios de diferencias divididas de Newton	Pequeño	Exacto	$n+1$	Pequeño	En general, se prefiere para análisis exploratorio
Polinomios de Lagrange	Pequeño	Exacto	$n+1$	Pequeño	En general, se prefiere cuando se conoce el orden
Interpolación cúbica segmentaria	Pequeño	Dependiente del método[7]	Ajuste de puntos por segmento	Moderado	Las primeras y segundas derivadas son iguales en los nodos

[7] La coincidencia es exacta para c–splines, y aproximada para b–splines.

5. DESCRIPCIÓN DE LOS MÉTODOS DE AMPLIADO

Los métodos de ampliado desarrollados y estudiados en este Trabajo de Graduación pueden clasificarse en función de si utilizan o no algún algoritmo de interpolación, tal como se muestra en la siguiente figura.

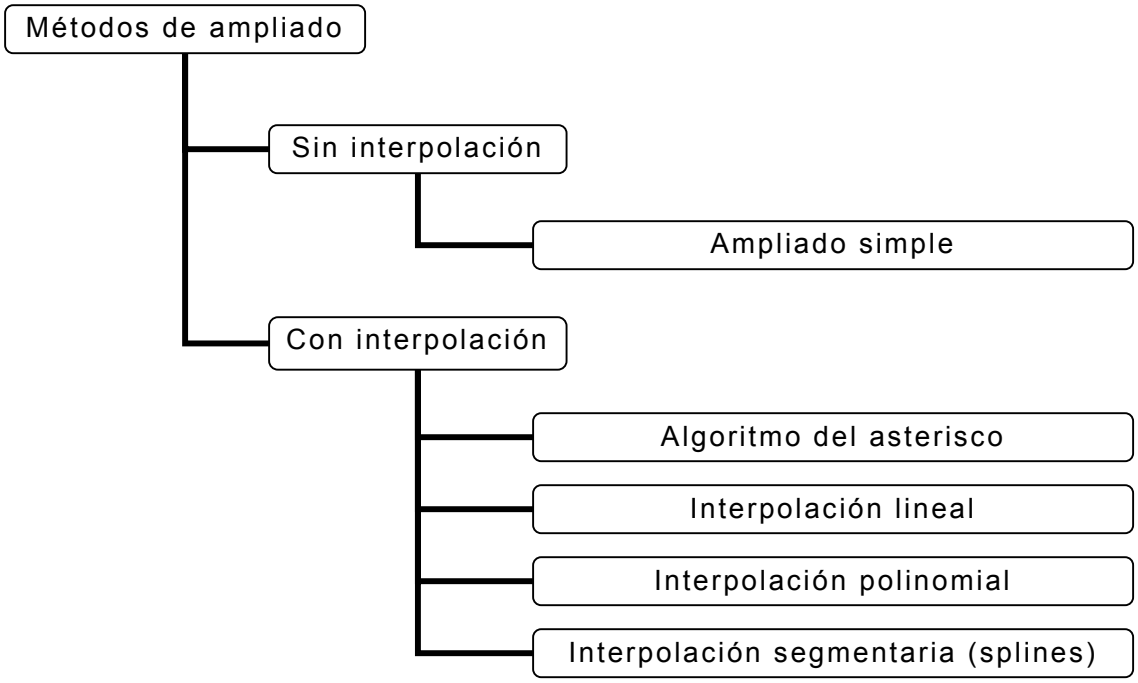


Figura 5.1. Clasificación de los métodos de ampliado estudiados.

También se desarrollaron dos métodos de reducción (véase la sección 5.3), los cuales se muestran en la Figura 5.2.

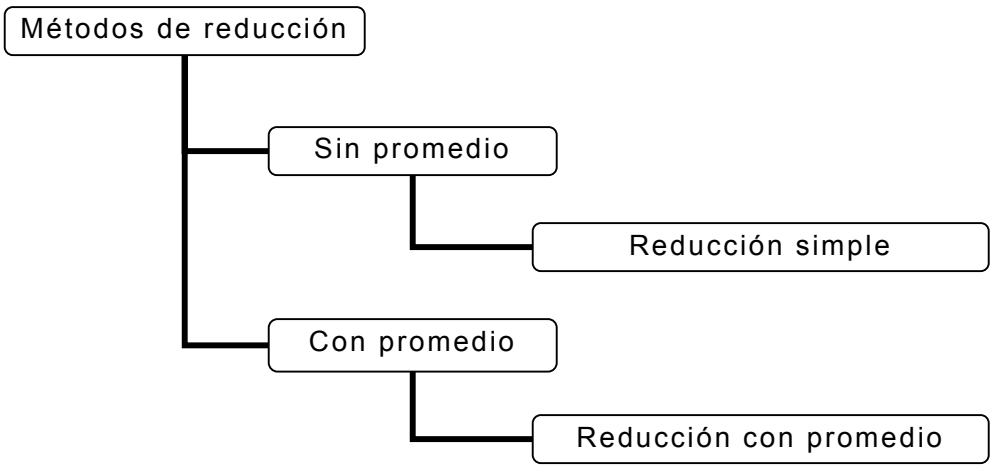


Figura 5.2. Clasificación de los métodos de reducción desarrollados.

Todos los métodos de ampliación / reducción estudiados requieren que se especifique la siguiente información para realizar el procesamiento de la imagen:

- Ubicación del archivo con formato de imagen que contiene la región rectangular que se desea ampliar, y las coordenadas de ésta[8].
- Cantidad de ampliación / reducción.
- Método de ampliación / reducción a utilizar.
- Ubicación y nombre del archivo con formato de imagen en donde se almacenará la imagen ampliada / reducida.

Independientemente del método de ampliación o reducción que se utilice, el proceso de ampliado para una imagen en color en formato RGB consta de los siguientes pasos:

- Asignar a una matriz **M** de tamaño **f** x **c** x 3 el contenido de la imagen o región de la imagen que se desea procesar, siendo **f** y **c** el número de filas y de columnas de ésta, respectivamente, de forma que a cada valor de la tercera dimensión de **M** se asigne un canal de color de la imagen (rojo, verde y azul).

[8] Dicha región puede especificarse como la totalidad de la imagen.

- Realizar el proceso de ampliado o reducción separadamente para cada canal de color de la imagen. Esto corresponde a procesar las submatrices de **M** de tamaño **f** x **c**, para los canales rojo, verde y azul, respectivamente.
- Combinar los resultados obtenidos para cada canal de color en una matriz de salida **M** de tamaño (**f** x **n**) x (**c** x **n**) x 3 (para una ampliación), ó (**f** / **n**) x (**c** / **n**) x 3 (para una reducción), en donde **n** es la cantidad de ampliado / reducción.

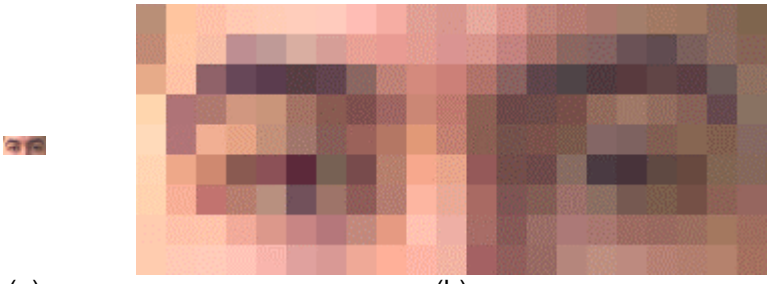
Los métodos de ampliación y reducción que se describirán a continuación asumen que se procesará una imagen en escala de grises o un único canal de color de una imagen RGB.

5.1. Métodos de ampliado sin interpolación

5.1.1. Ampliado simple

El ampliado simple es el único método de ampliado sin interpolación estudiado en este Trabajo de Graduación, y es el que comúnmente realizan la mayoría de herramientas de presentación / edición de imágenes, tales como MSPAINT de Microsoft Corporation, y consiste en la repetición de todos los píxeles de la imagen, tantas veces como sea necesario, para formar cuadrados de **n** píxeles iguales por lado, en donde **n** es el número de veces que se desea ampliar la imagen[9]. En la Figura 5.3 se muestra un ejemplo de este tipo de ampliación.

[9] Por ejemplo, si la imagen original tiene **f** filas y **c** columnas, ésta contendrá **f** x **c** píxeles. Luego, la imagen ampliada **n** veces estará formada por (**f** x **n**) x (**c** x **n**) píxeles.



(a) (b)

Figura 5.3. Ejemplo de ampliado simple.
(a) Imagen original. (b) Imagen ampliada 15 veces.

El ampliado simple no constituye ninguna solución innovadora al problema del procesamiento de imágenes; sin embargo, se incluye en este Trabajo de Graduación puesto que sirve como un punto de referencia con el que se pueden comparar los beneficios del resto de los métodos estudiados.

5.2. Métodos de ampliado con interpolación

Tal como aparece en la Figura 5.1, el resto de los métodos estudiados utilizan uno de los siguientes tipos de interpolación:

- Lineal
- Polinomial
- Segmentaria (splines)

A excepción del ampliado simple, todos los métodos siguen la misma mecánica para el tratamiento de la imagen, difiriendo solamente en el tipo de interpolación utilizado. Dicho procedimiento común consta de los siguientes pasos:

- Creación del enrejado
- Determinación de las diagonales (sólo para el algoritmo del asterisco)
- Proceso de rellenado

Se proseguirá ahora con la explicación de los procesos anteriores, para luego abordar los fundamentos teóricos de los diversos tipos de

interpolación utilizados, finalizando con la descripción de los métodos específicos de ampliación que utilizan dichos algoritmos de interpolación.

5.2.1. Creación del enrejado

1. El punto de partida es colocar cada píxel de la imagen original en el centro de una matriz de **n** x **n**, en donde **n** es el número de veces que se desea ampliar la imagen. A esta región cuadrada de **n** x **n** píxeles se le conocerá como *celda*.

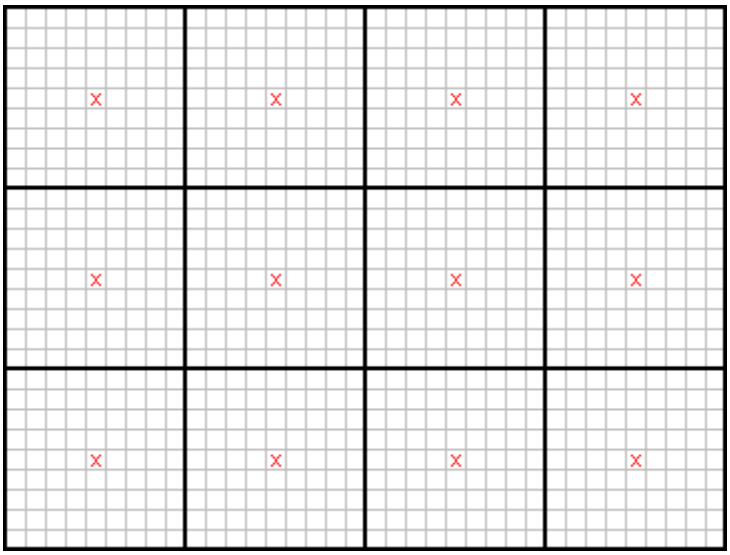


Figura 5.4. Primer paso en el procedimiento común de los métodos de ampliación que utilizan interpolación.

Para el caso mostrado en la Figura 5.4, **n** es impar. Sin embargo, si **n** es par, las celdas de **n** x **n** carecen de un píxel central. En lugar de ello, el centro está constituido por una región cuadrada de 4 píxeles. En este caso, se toma como píxel central al situado en la esquina superior izquierda de dicha región, tal como se muestra en la Figura 5.5.

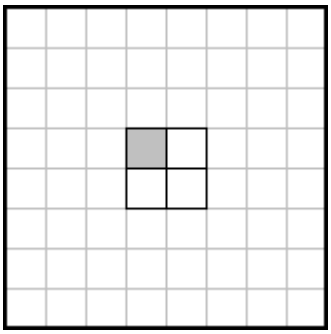


Figura 5.5. Definición de píxel central (en gris) para una cantidad de ampliado par (n = 8).

2. Posteriormente se interpolan los píxeles de todas las columnas (líneas verticales) que contienen a los píxeles originales, en función de estos últimos.

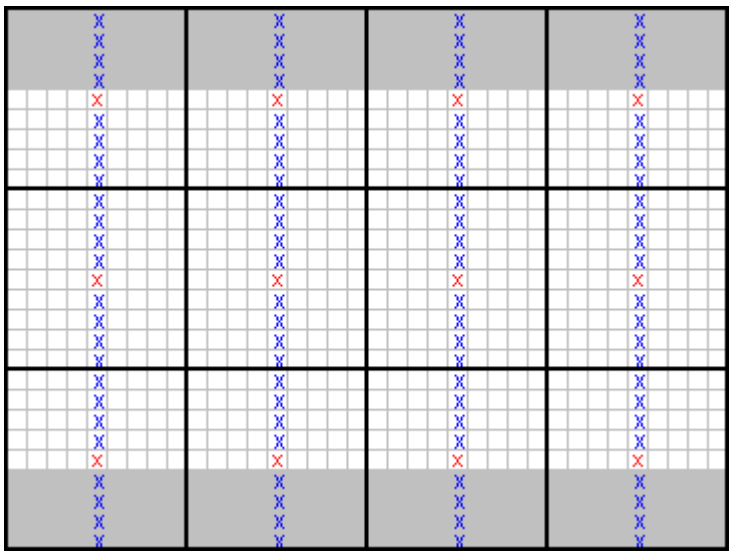


Figura 5.6. Segundo paso en el procedimiento común de los métodos de ampliación que utilizan interpolación.

Con el algoritmo del asterisco no se puede calcular el valor de los píxeles que se encuentran en la periferia de la imagen (los que se ubican dentro de la región sombreada de la Figura 5.6). La razón de este inconveniente se abordará más adelante, cuando se presenten las características de cada uno de los métodos de ampliación.

3. Luego se realiza el mismo procedimiento, pero para las filas (líneas horizontales) que contienen a los píxeles originales.

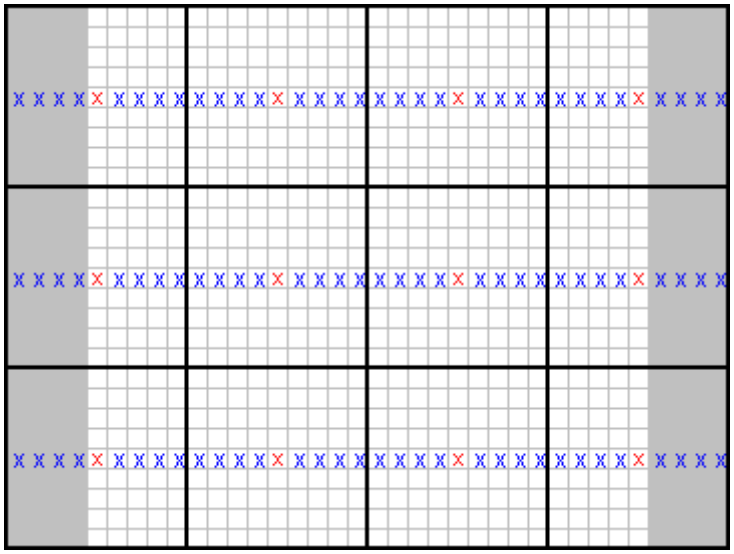


Figura 5.7. Tercer paso en el procedimiento común de los métodos de ampliación que utilizan interpolación.

Nuevamente, la región sombreada muestra los píxeles a los cuales el algoritmo del asterisco no les pueden asignar valores.

4. A continuación, las matrices de los pasos 2 y 3 se combinan, generando la matriz que se muestra en la figura siguiente.

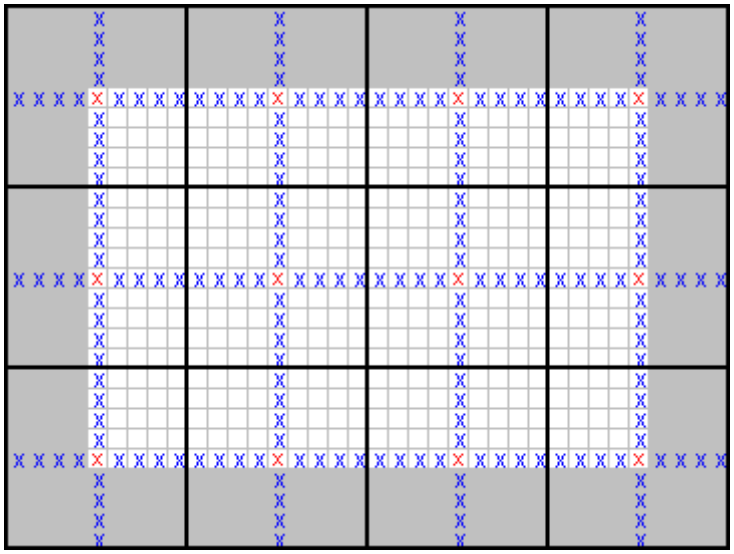


Figura 5.8. Cuarto paso en el procedimiento común de los métodos de ampliación que utilizan interpolación.

5.2.2. Determinación de las diagonales

Este paso en el proceso de ampliación sólo es requerido por los métodos que utilizan el algoritmo del asterisco, y consiste en el cálculo de los valores de los píxeles que se encuentran en la trayectoria que une los centros de todas las celdas adyacentes diagonales, luego de que a la imagen en proceso de ampliación se le ha aplicado la etapa de creación del enrejado (Figura 5.8).

Se desarrollaron dos formas de realizar la determinación de las diagonales, las cuales se describen en seguida.

Diagonales interpoladas

Esta forma de calcular las diagonales consiste en la distribución uniforme del valor del píxel central de dos celdas adyacentes diagonales sobre todos los píxeles ubicados en la trayectoria de la recta que los une. Por ejemplo, considérese la imagen en proceso de ampliado mostrada en la Figura 5.9. En ella aparece una línea diagonal de píxeles en gris, en cuyos extremos se encuentran dos píxeles encerrados en un círculo, para facilitar su identificación. El valor de los píxeles en gris se calcula interpolando linealmente el valor de los píxeles en los extremos.

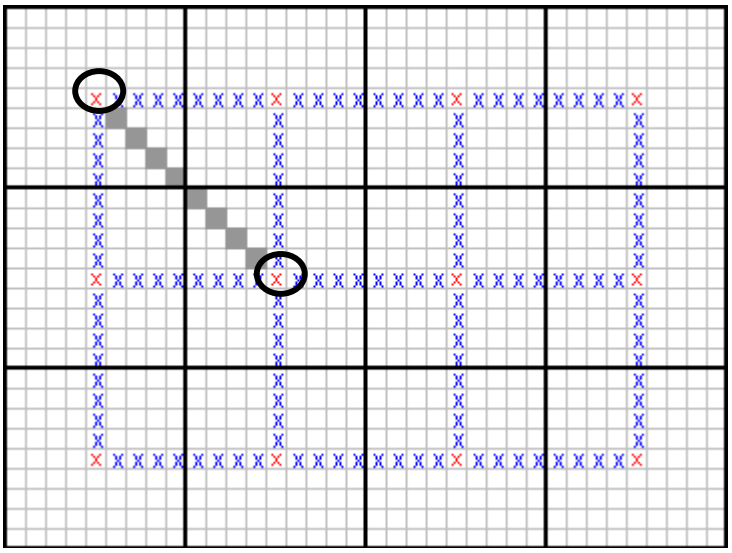


Figura 5.9. Imagen de ejemplo para el cálculo de las diagonales interpoladas.

Este proceso se repite con todas las diagonales de la imagen, tal como lo sugiere la Figura 5.10, hasta que se interpolan todas las diagonales de la misma (Figura 5.11). Nótese que las diagonales ubicadas en la periferia de la imagen no fueron calculadas. Las razones de ello se explicarán más adelante, cuando se describan las características de cada uno de los métodos de ampliado.

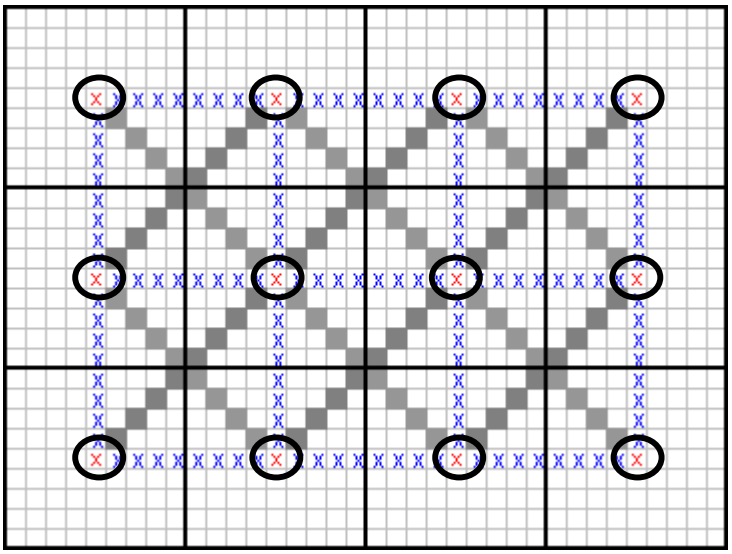


Figura 5.10. Diagonales interpoladas (píxeles oscuros) a partir de los valores de sus extremos (píxeles en círculo).

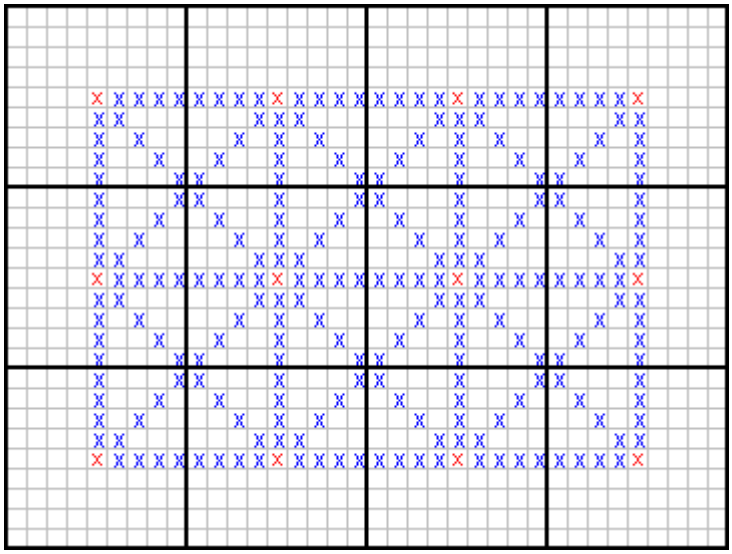


Figura 5.11. Proceso de determinación de diagonales interpoladas finalizado.

Diagonales promediadas

El otro método estudiado para obtener las diagonales de la imagen ampliada consiste en la ejecución de los siguientes pasos para cada uno de los píxeles (P) que conforman las diagonales:

- Localizar la fila (F) a la que pertenece el píxel (P) en cuestión.
- Localizar el píxel más cercano a la izquierda (P_izq) y a la derecha (P_der) del píxel P, en la fila F, y que pertenezcan a la retícula de horizontales y verticales producto de la fase de creación del enrejado.
- Interpolar linealmente el valor de P, a partir de P_izq y P_der.
- Localizar la columna (C) a la que pertenece el píxel (P) en cuestión.
- Localizar el píxel más cercano hacia arriba (P_arriba) y hacia abajo (P_abajo) del píxel P, en la columna C, y que pertenezcan a la retícula de horizontales y verticales producto de la fase de creación del enrejado.
- Interpolar linealmente el valor de P, a partir de P_arriba y P_abajo.
- Promediar los dos valores interpolados para P.

La figura siguiente clarifica los pasos antes mencionados.

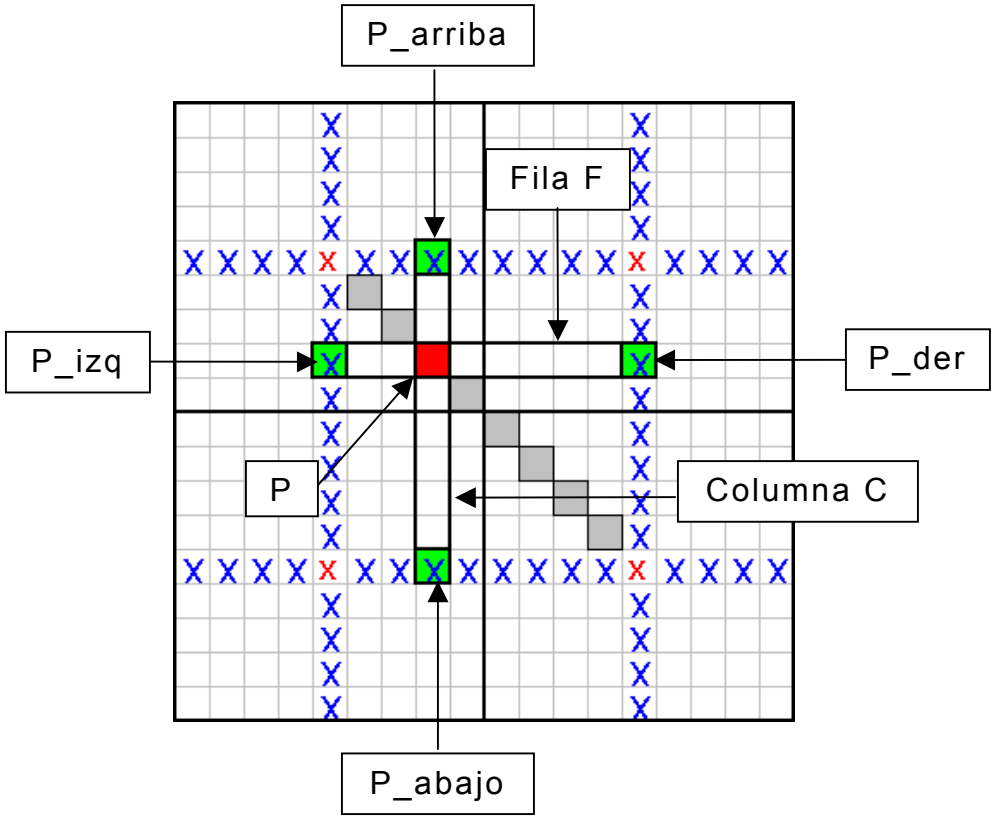


Figura 5.12. Determinación de las diagonales promediadas.

Al finalizar, la imagen luce como en la Figura 5.11, si bien los valores de los píxeles en las diagonales son diferentes que los obtenidos por el método anterior. Por otro lado, al igual que con dicho método, los valores de los píxeles ubicados en diagonales en la periferia de la imagen no se calculan.

5.2.3. Proceso de rellenado

Este proceso es aquél por medio del cual se calculan los valores de los píxeles intermedios en una imagen en proceso de ampliación, o sea, todos los que aún no tienen un valor asignado luego de haberle aplicado a la imagen el proceso de creación del enrejado y de determinación de las diagonales (para el caso del algoritmo del asterisco solamente). Para este Trabajo de Graduación se desarrollaron tres procesos de relleno diferentes, los cuales se listan a continuación:

- Relleno radial
- Relleno en espiral
- Relleno en malla

Los dos primeros únicamente se utilizan para el algoritmo del asterisco, el cual requiere la aplicación del proceso de determinación de las diagonales. A continuación se describen cada uno de ellos.

Relleno radial

Este tipo de relleno se efectúa en las imágenes en proceso de ampliación a las que se les ha aplicado la etapa de determinación de diagonales, en cualquiera de sus dos versiones.

Para describir este proceso de relleno, considérese la Figura 5.13. En ella se muestra una imagen de 3x3, ampliada 9 veces. El proceso de relleno se ha aplicado a la celda central. Las flechas en la figura se encuentran sobre las líneas de píxeles a los que se les asignarán valores por medio de interpolación lineal, mientras que las puntas de dichas flechas señalan a los píxeles que darán origen a tales interpolaciones. Este proceso se repite en todas las celdas de la imagen ampliada. El nombre de este método de rellenado proviene de la dirección en la que se realiza la interpolación, tal como lo sugieren las flechas en la Figura 5.13.

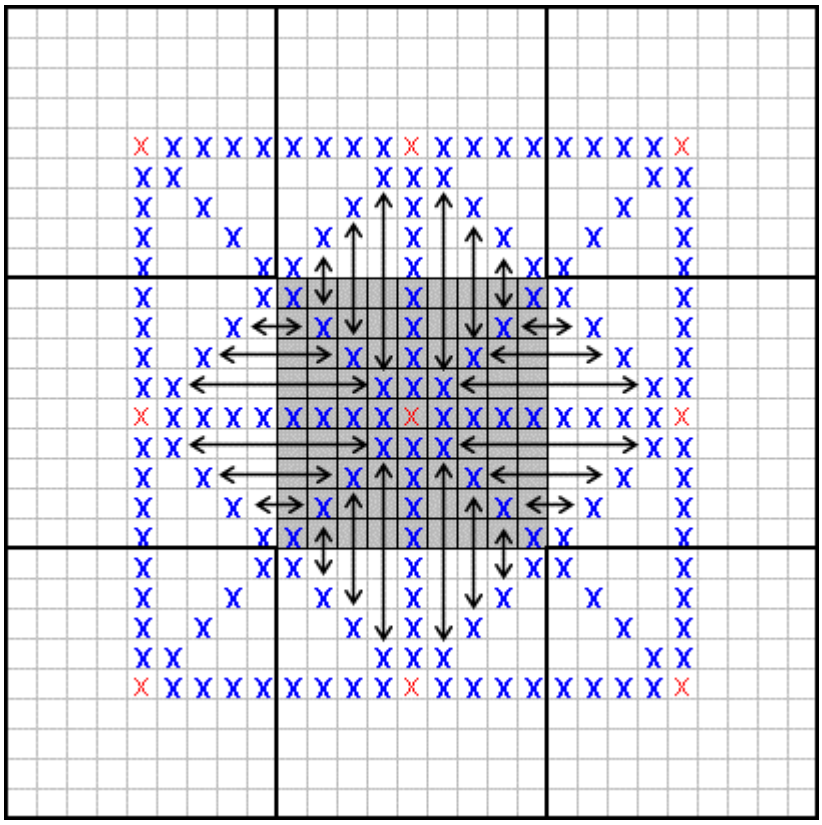


Figura 5.13. Ejemplo del proceso de rellenado radial.

Relleno en espiral

Este método de relleno también se aplica sólo a las imágenes en proceso de ampliación a las que se les ha determinado las diagonales. Para describir este proceso, refiérase a la Figura 5.14. En ella se muestra una celda de la imagen ampliada, y nuevamente unas flechas, que son las que indican los píxeles que están siendo interpolados linealmente, y las parejas de píxeles utilizados para dichas interpolaciones. También puede observarse que la disposición concéntrica de las flechas se asemeja a un espiral, lo que dio origen al nombre del algoritmo.

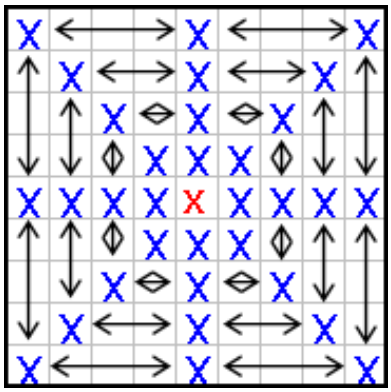


Figura 5.14. Ejemplo del proceso de rellenado en espiral.

Relleno en malla

El otro método estudiado para rellenar los píxeles intermedios de la imagen ampliada consiste en la ejecución de los siguientes pasos para cada uno de los píxeles (P) que conforman la imagen, y cuyos valores aún no han sido asignados.

- Localizar la fila (F) a la que pertenece el píxel (P) en cuestión.
- Localizar los píxeles (P_fila) que pertenecen a la fila F y al enrejado.
- Interpolar el valor de P, a partir de los píxeles P_fila[10].
- Localizar la columna (C) a la que pertenece el píxel (P) en cuestión.
- Localizar los píxeles (P_col) que pertenecen a la fila C y al enrejado.
- Interpolar el valor de P, a partir de los píxeles P_col.
- Promediar los dos valores interpolados para P.

La figura siguiente clarifica los pasos antes mencionados.

[10] El método de interpolación lineal sólo requiere de dos valores para realizar la interpolación. En ese caso, sólo el píxel más cercano a la izquierda y a la derecha (en una fila) o más cercano hacia arriba y abajo (en una columna) participarían en la interpolación.

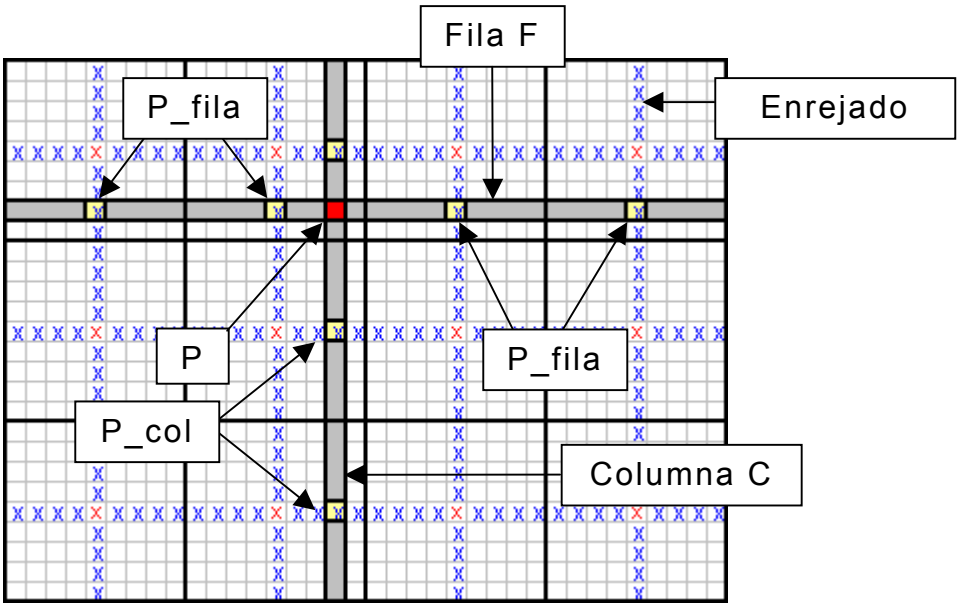


Figura 5.15. Ejemplo del proceso de rellenado en malla.

5.2.4. Métodos de interpolación

Interpolación lineal

Este tipo de interpolación consiste en la distribución uniforme de los valores de dos píxeles dados, entre los píxeles intermedios a éstos. Por ejemplo, considérese el caso que se presenta en la figura siguiente. En ella se muestran dos píxeles, con valores de 100 (color claro) y 10 (color oscuro). Entre dichos puntos se encuentran 8 píxeles más, a los que se les asignarán los valores interpolados, siendo éstos, de izquierda a derecha, 90, 80, 70,..., y 20.

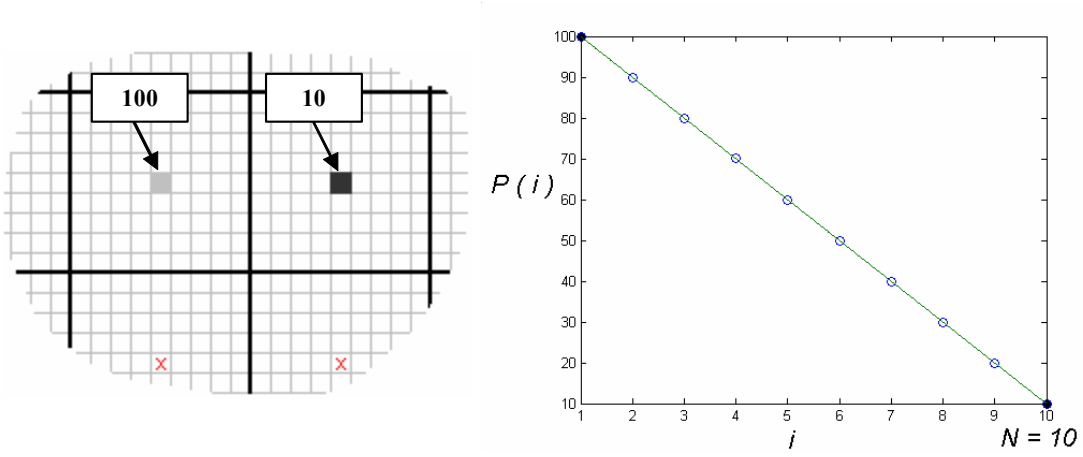


Figura 5.16. Dos píxeles de una imagen en proceso de ampliación con interpolación lineal.

Los valores mencionados pueden obtenerse a partir de la siguiente expresión:

$$P_i = P_1 + \frac{P_N - P_1}{N-1}(i-1) \qquad i = 1,\ 2,\ 3,\ ...,\ N \tag{5.1}$$

en donde N es el número total de píxeles: los intermedios más los dos extremos (que en el ejemplo de la Figura 5.16, N = 10); P_i es el valor del píxel en la posición i; y en donde las posiciones i = 1 e i = N corresponden a los píxeles extremos (con valores de 100 y 10, respectivamente, en este ejemplo).

Interpolación polinomial

Este tipo de interpolación aborda el problema que se plantea a continuación: se proporciona una tabla de n puntos (x_i, y_i), los cuales corresponden a parejas de datos:

x	x_0	x_1	x_2	...	x_{n+1}
y	y_0	y_1	y_2	...	y_{n+1}

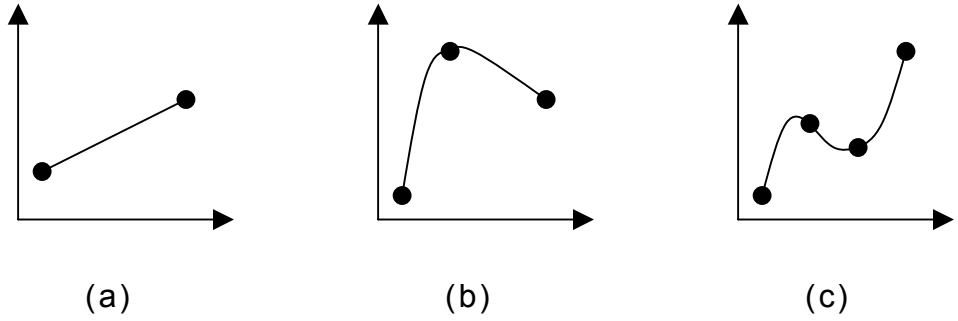
y se busca un polinomio p con el menor grado posible, para el cual

$$p(x_i) = y_i \qquad (1 \le i \le N) \tag{5.2}$$

La fórmula general de un polinomio de n-ésimo orden es:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \qquad (5.3)$$

Para $n+1$ puntos, existe uno y sólo un polinomio de n-ésimo orden que pasa a través de todos los puntos. Por ejemplo, hay sólo una línea recta (es decir, un polinomio de primer orden) que conecta dos puntos (Figura 5.17a). De manera similar hay sólo un polinomio de segundo orden que conecta a tres puntos (Figura 5.17b). El polinomio de interpolación consiste en determinar el único polinomio de n-ésimo orden que se ajusta a los $n+1$ puntos dados. Este polinomio proporciona una fórmula para calcular los valores intermedios.



(a) (b) (c)

Figura 5.17. Ejemplos de interpolación polinomial:
a) Primer orden; b) segundo orden y c) tercer orden.

Aunque existe uno y sólo un polinomio de n-ésimo orden que se ajusta a los $n+1$ puntos, existe una gran variedad de fórmulas matemáticas –todas ellas equivalentes– mediante las cuales se puede expresar este polinomio. Dos de ellas, muy populares, son los polinomios de Newton y los de Lagrange. Para los propósitos de este Trabajo de Graduación, se escogió la forma de Lagrange, debido a que no necesita de la resolución de ecuaciones lineales (a diferencia del polinomio de Newton), lo que conlleva a una mayor eficiencia computacional, sobre todo cuando el número de puntos de datos es grande (como ocurre a menudo con las imágenes digitales), además de que es menos susceptible a los efectos de los errores de redondeo [4].

Polinomio de interpolación de Lagrange

El polinomio de interpolación de Lagrange, simplemente es una reformulación del polinomio de Newton que evita los cálculos de las diferencias divididas[11]. Éste se puede representar concretamente como:

$$p_n(x) = \sum_{i=0}^{n} L_i(x) p(x_i) \tag{5.4}$$

siendo:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{5.5}$$

en donde $\prod$ denota “el producto de”. Por ejemplo, la versión lineal (n = 1) es:

$$p_1(x) = \frac{x - x_1}{x_0 - x_1} p(x_0) + \frac{x - x_0}{x_1 - x_0} p(x_1) \tag{5.6}$$

y la versión de segundo orden es:

$$p_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} p(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} p(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} p(x_2) \tag{5.7}$$

y así sucesivamente.

Interpolación segmentaria (splines)

Una función spline está formada por varios polinomios, cada uno definido sobre un subintervalo, que se unen entre sí obedeciendo a ciertas condiciones de continuidad. Para una definición formal, se procederá como sigue: supóngase que se han especificado $n+1$ puntos

---

[11] Véase [1], Pág. 349 y siguientes para una explicación más detallada de la interpolación polinomial de Lagrange y la de Newton con diferencias divididas.

$t_0 < t_1 < \ldots < t_n$. Estos puntos se llaman *nudos*. Supóngase además que se ha fijado un entero $k \geq 0$. Una *función spline de grado k* con nudos en $t_0, t_1, \ldots, t_n$ es una función *S* que satisface las siguientes condiciones:

a) En cada intervalo $[t_{i-1}, t_i)$, *S* es un polinomio de grado $\leq k$.

b) *S* tiene una derivada de orden $(k-1)$ continua en $[t_0, t_n]$.

Por consiguiente, *S* es un polinomio continuo por pedazos de a lo sumo grado *k*, que tiene derivadas continuas de orden hasta $(k-1)$.

Para los propósitos de este Trabajo de Graduación, se ha utilizado la interpolación *cúbica* segmentaria, es decir, funciones spline de orden tres ($k = 3$), debido a que éstas presentan la propiedad de que las conexiones entre ecuaciones cúbicas adyacentes son visualmente suaves [1], siendo ésta una característica de mucha utilidad para evitar los cambios bruscos de color en las regiones adyacentes de una imagen.

C-splines

La función c-spline[12] consiste en polinomios cúbicos segmentarios que se ajustan a puntos de datos dados. La esencia de la interpolación de c-spline radica en aplicar un polinomio cúbico a cada intervalo entre dos puntos de datos consecutivos. Por otro lado, también se requiere que la primera y segunda derivadas de los polinomios cúbicos sean continuas en cada punto de datos. Por consiguiente, tanto el valor funcional como la primera y segunda derivadas son continuas en todo el dominio. Sin embargo, para determinar los coeficientes del polinomio

[12] Gran parte del material sobre c-splines y b-splines fue extraído de [4]. Algunas de las expresiones matemáticas mostradas aquí difieren de aquellas encontradas en dicho libro. Ello es producto de errores tipográficos en la mencionada fuente bibliográfica.

cúbico de cada intervalo es preciso determinar simultáneamente los coeficientes de todos los intervalos.

En el ajuste de una función c-spline $f(s)$ a los puntos de datos (s_i, f_i) intervienen las siguientes cantidades:

- $s_i, \quad i = 1, 2, \ldots, n$: conocidas.
- $f_i, \quad i = 1, 2, \ldots, n$: conocidas.
- $f_i', \quad i = 1, 2, \ldots, n$: por determinar[13].

Considérese un intervalo, $s_i < s < s_{i+1}$ con $h_i = s_{i+1} - s_i$, como se muestra en la Figura 5.18. Si se utiliza la coordenada local $t = s - s_i$, se puede escribir un polinomio cúbico para un intervalo así:

$$f(t) = a + bt + ct^2 + et^3 \tag{5.8}$$

donde $0 \le t \le h_i$.

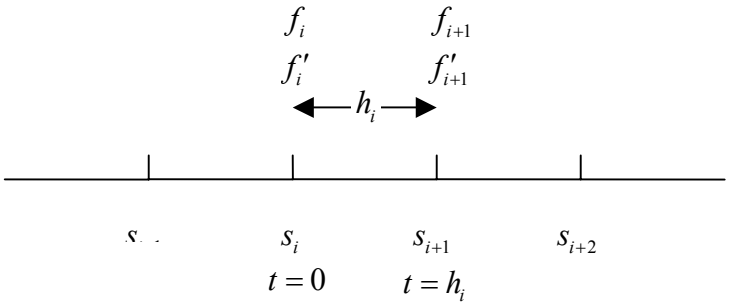


Figura 5.18. Intervalo entre dos puntos de datos para c-spline.

Primero se necesita que $f(t)$ sea igual al valor conocido de la función $f(t)$ en $t = 0$ y $t = h_i$:

$$f_i = a \tag{5.9}$$

[13] f_i' y f_i'' son, respectivamente, la primera y la segunda derivada de f_i.

$$f_{i+1} = a + bh_i + ch_i^2 + eh_i^3 \tag{5.10}$$

Las dos ecuaciones anteriores no bastan para determinar las cuatro constantes *a*, *b*, *c* y *e*, pero si se especifican dos condiciones más se contará con suficientes ecuaciones para determinar las constantes. Estas condiciones surgen del requisito de que f'' y f' sean continuas en cada uno de los puntos. Si se logra esto, toda la curva será continua en la función ajustada, la primera derivada y la segunda derivada.

La segunda derivada de la ecuación (5.8) es

$$f''(t) = 2c + 6et \tag{5.11}$$

que en los puntos i e $i+1$ se convierte, respectivamente, en

$$f_i'' = 2c \tag{5.12}$$

$$f_{i+1}'' = 2c + 6eh_i \tag{5.13}$$

donde f_i'' y f_{i+1}'' son valores de f'' en i e $i+1$, respectivamente, y que se deberán determinar. Las dos ecuaciones anteriores pueden escribirse así:

$$c = \frac{f_i''}{2} \tag{5.14}$$

$$e = \frac{f_{i+1}'' - f_i''}{6h_i} \tag{5.15}$$

El coeficiente a ya está dado por la ecuación (5.9). El coeficiente b se determina eliminando a, c y e en la ecuación (5.10) mediante las ecuaciones (5.9), (5.14) y (5.15),

$$b = \frac{f_{i+1} - f_i}{h_i} - \frac{f_{i+1}'' + 2f_i''}{6} h_i \tag{5.16}$$

Así, el polinomio cúbico de la ecuación (5.8) está expresado por:

$$f(t) = f_i + \left(\frac{f_{i+1} - f_i}{h_i} - \frac{f''_{i+1} + 2f''_i}{6} h_i \right) t + \frac{f''_i}{2} t^2 + \frac{f''_{i+1} - f''_i}{6h_i} t^3 \qquad (5.17)$$

Diferenciando la ecuación (5.17) e igualando la primera derivada de f en $t = 0$ y $t = h_i$ a f'_i y f'_{i+1}, respectivamente,

$$f'_i = -\frac{h_i}{6}\left(f''_{i+1} + 2f''_i\right) + \frac{1}{h_i}\left(f_{i+1} - f_i\right) \qquad (5.18)$$

$$f'_{i+1} = \frac{h_i}{6}\left(2f''_{i+1} + f''_i\right) + \frac{1}{h_i}\left(f_{i+1} - f_i\right) \qquad (5.19)$$

Para el intervalo adyacente de $s_{i-1} < s < s_i$, la ecuación (5.19) se convierte en

$$f'_i = \frac{h_{i-1}}{6}\left(2f''_i + f''_{i-1}\right) + \frac{1}{h_{i-1}}\left(f_i - f_{i-1}\right) \qquad (5.20)$$

donde $h_{i-1} = x_i - x_{i-1}$. Para que haya continuidad, la f'_i de la ecuación (5.20) debe ser igual a la de la ecuación (5.18). Si se elimina f'_i de las dos ecuaciones resulta

$$h_{i-1} f''_{i-1} + \left(2h_{i-1} + 2h_i\right) f''_i + h_i f''_{i+1} = 6\left(\frac{1}{h_{i-1}} f_{i-1} - \left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right) f_i + \frac{1}{h_i} f_{i+1} \right) \qquad (5.21)$$

donde toda la ecuación se multiplicó por 6.

Se puede escribir la ecuación anterior para todos los puntos excepto para los dos extremos. Es decir, de f''_i hay $n-2$ ecuaciones, mientras que el número de no determinadas es n. Por tanto, se necesitan dos ecuaciones más para determinar todas las f''_i no determinadas, y se pueden obtener de las condiciones de frontera. A continuación se explican tres formas de especificar las condiciones de frontera en los dos extremos.

a) *Especificar f_i'' en los extremos.*

Si se prescriben los valores de f_i'' en los dos extremos, $i=1$ e $i=n$, el conjunto de ecuaciones se convierte en

$$\begin{aligned}
&(2h_1+2h_2)f_2''+h_2f_3'' \\
&=6\left(\frac{1}{h_1}f_1-(\frac{1}{h_1}+\frac{1}{h_2})f_2+\frac{1}{h_2}f_3\right)-h_1f_1'' \\
&h_{i-1}f_{i-1}''+(2h_{i-1}+2h_i)f_i''+h_if_{i+1}'' \\
&=6\left(\frac{1}{h_{i-1}}f_{i-1}-(\frac{1}{h_{i-1}}+\frac{1}{h_i})f_i+\frac{1}{h_i}f_{i+1}\right) \\
&h_{n-2}f_{n-2}''+(2h_{n-2}+2h_{n-1})f_{n-1}'' \\
&=6\left(\frac{1}{h_{n-2}}f_{n-2}-(\frac{1}{h_{n-2}}+\frac{1}{h_{n-1}})f_{n-1}+\frac{1}{h_{n-1}}f_n\right)-h_{n-1}f_n''
\end{aligned} \qquad (5.22)$$

Las ecuaciones anteriores constituyen $n-2$ ecuaciones para $n-2$ incógnitas, f_i''. Si se escriben las ecuaciones en forma de matriz, la matriz de coeficientes se convertirá en una forma especial, llamada matriz tridiagonal, en la que todos los elementos son cero, excepto por las tres líneas diagonales. Si bien es cierto dicha ecuación puede resolverse mediante la solución estándar para ecuaciones lineales, existe un método de resolución especial que es mucho más eficiente[14]. Aunque en la mayor parte de las situaciones no se conocen las f'' en los extremos, se puede adoptar la estrategia de suponer que $f''=0$ en los extremos. Geométricamente, esto equivale a suponer que la curva se convierte en una recta hacia los extremos. Sin embargo, dada la naturaleza no determinística de las imágenes digitales, esta suposición no se considera con mucha validez. El siguiente punto ofrece un panorama más aceptable.

[14] En el ANEXO E se encuentra el desarrollo de la solución de la ecuación tridiagonal.

b) *Extrapolar f'' desde dentro.*

Esta opción fue la que se decidió implementar en el Trabajo de Graduación, debido a que los valores de la segunda derivada de la función en los dos extremos no se escogen arbitrariamente, sino en base a la tendencia general de los datos, esto es, extrapolando linealmente dichos valores con la ayuda de las segundas derivadas en los dos puntos más cercanos. Este artificio puede considerarse bastante atinado, partiendo del hecho de que la segunda derivada de una función cúbica (como lo son las funciones spline) es lineal.

La extrapolación de f_1'' a partir de f_2'' y f_3'' se escribe así:

$$f_1''=\left(1+\frac{h_1}{h_2}\right)f_2''-\frac{h_1}{h_2}f_3'' \qquad (5.23)$$

Si se hace $i=2$ en la ecuación (5.21) y se elimina con la ecuación (5.23), se obtiene

$$\left(3h_1+2h_2+\frac{h_1^2}{h_2}\right)f_2''+\left(h_2-\frac{h_1^2}{h_2}\right)f_3''=6\left(\frac{1}{h_1}f_1-(\frac{1}{h_1}+\frac{1}{h_2})f_2+\frac{1}{h_2}f_3\right) \qquad (5.24)$$

La ecuación anterior sustituye a la primera ecuación de (5.22).

Por otro lado, la extrapolación de f_n'' a partir de f_{n-2}'' y f_{n-1}'' viene dada por:

$$f_n''=\left(1+\frac{h_{n-1}}{h_{n-2}}\right)f_{n-1}''-\frac{h_{n-1}}{h_{n-2}}f_{n-2}'' \qquad (5.25)$$

Haciendo $i=n-1$ en la ecuación (5.21) y combinándola con la ecuación (5.25), resulta en

$$\left(h_{n-2}-\frac{h_{n-1}^2}{h_{n-2}}\right)f''_{n-2}+\left(2h_{n-2}+3h_{n-1}+\frac{h_{n-1}^2}{h_{n-2}}\right)f''_{n-1}$$
$$=6\left(\frac{1}{h_{n-2}}f_{n-2}-(\frac{1}{h_{n-2}}+\frac{1}{h_{n-1}})f_{n-1}+\frac{1}{h_{n-1}}f_n\right) \qquad (5.26)$$

La ecuación anterior sustituye a la última ecuación de (5.22), resultando el sistema siguiente:

$$\left(3h_1+2h_2+\frac{h_1^2}{h_2}\right)f''_2+\left(h_2-\frac{h_1^2}{h_2}\right)f''_3$$
$$=6\left(\frac{1}{h_1}f_1-(\frac{1}{h_1}+\frac{1}{h_2})f_2+\frac{1}{h_2}f_3\right)$$
$$h_{i-1}f''_{i-1}+\left(2h_{i-1}+2h_i\right)f''_i+h_if''_{i+1}$$
$$=6\left(\frac{1}{h_{i-1}}f_{i-1}-(\frac{1}{h_{i-1}}+\frac{1}{h_i})f_i+\frac{1}{h_i}f_{i+1}\right) \qquad (5.27)$$
$$\left(h_{n-2}-\frac{h_{n-1}^2}{h_{n-2}}\right)f''_{n-2}+\left(2h_{n-2}+3h_{n-1}+\frac{h_{n-1}^2}{h_{n-2}}\right)f''_{n-1}$$
$$=6\left(\frac{1}{h_{n-2}}f_{n-2}-(\frac{1}{h_{n-2}}+\frac{1}{h_{n-1}})f_{n-1}+\frac{1}{h_{n-1}}f_n\right)$$

El sistema de ecuaciones anterior, expresado en forma matricial, resulta:

$$A\cdot B=C \qquad (5.28)$$

con

$$A=\begin{bmatrix} 3h_1+2h_2+\frac{h_1^2}{h_2} & h_2-\frac{h_1^2}{h_2} & 0 & 0 & 0 & 0 \\ h_2 & 2(h_2+h_3) & h_3 & 0 & 0 & 0 \\ 0 & h_3 & 2(h_3+h_4) & h_4 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & 0 & 0 & 0 & h_{n-2}-\frac{h_{n-1}^2}{h_{n-2}} & 2h_{n-2}+3h_{n-1}+\frac{h_{n-1}^2}{h_{n-2}} \end{bmatrix}$$

(5.29)

$$B=\begin{bmatrix} f_2'' \\ f_3'' \\ f_4'' \\ \vdots \\ f_{n-2}'' \\ f_{n-1}'' \end{bmatrix} \qquad (5.30)$$

$$C=\begin{bmatrix} 6\left(\frac{1}{h_1}f_1-(\frac{1}{h_1}+\frac{1}{h_2})f_2+\frac{1}{h_2}f_3\right) \\ 6\left(\frac{1}{h_2}f_2-(\frac{1}{h_2}+\frac{1}{h_3})f_3+\frac{1}{h_3}f_4\right) \\ 6\left(\frac{1}{h_3}f_3-(\frac{1}{h_3}+\frac{1}{h_4})f_4+\frac{1}{h_4}f_5\right) \\ \vdots \\ 6\left(\frac{1}{h_{n-3}}f_{n-3}-(\frac{1}{h_{n-3}}+\frac{1}{h_{n-2}})f_{n-2}+\frac{1}{h_{n-2}}f_{n-1}\right) \\ 6\left(\frac{1}{h_{n-2}}f_{n-2}-(\frac{1}{h_{n-2}}+\frac{1}{h_{n-1}})f_{n-1}+\frac{1}{h_{n-1}}f_n\right) \end{bmatrix} \qquad (5.31)$$

Este sistema de ecuaciones también constituye una matriz tridiagonal, por lo que puede resolverse por el método mencionado anteriormente.

c) *Condición de frontera cíclica*

La condición de frontera cíclica se aplica si el primer dato y el último son idénticos y las derivadas en esos puntos de datos también son idénticas. Esto ocurre si el conjunto de datos completo representa un ciclo de una curva que se repite. Éste, en general, no es el caso cuando se trata de imágenes digitales, debido a la naturaleza no determinística de la información (a menos que se trate de una imagen de patrones repetitivos).

B-splines cúbica

La función b-spline consiste en polinomios segmentarios determinados por una serie de puntos de control, (s_i, p_i), $i = 1, 2, \ldots, n$. La función b-spline basada en polinomios cúbicos se denomina b-spline cúbica y pertenece a la familia de c-spline recién descrita pero con un carácter diferente. Se les dice puntos de control a los pares (s_i, p_i) porque la b-spline no pasa por ellos excepto en condiciones especiales. La función b-spline resulta útil para generar curvas suaves en aplicaciones en las que la tendencia general de los datos es más importante que un ajuste estricto.

Un segmento de la b-spline cúbica se determina con cuatro puntos de control consecutivos, (s_{i-1}, p_{i-1}), (s_i, p_i), (s_{i+1}, p_{i+1}) y (s_{i+2}, p_{i+2}), mediante[15]

[15] Para una deducción matemática de (5.32), así como excelentes fundamentos teóricos de b-splines, splines en general, y sus aplicaciones, consúltese [5].

$$f(s)=\frac{1}{6}[(1-t)^3 p_{i-1}+(3t^3-6t^2+4)p_i+ \\ +(-3t^3+3t^2+3t+1)p_{i+1}+t^3 p_{i+2}],\quad 0\le t\le 1 \tag{5.32}$$

donde $t=s-s_i$ es una coordenada local y $s_i=i$. Para $s=s_i$ y $s=s_{i+1}$ (o lo que es equivalente, $t=0$ y $t=1$, respectivamente), f, f' y f'' tienen los siguientes valores:

$$f(s_i)=\frac{p_{i-1}+4p_i+p_{i+1}}{6},\quad f'(s_i)=\frac{p_{i+1}-p_{i-1}}{2},\quad f''(s_i)=p_{i+1}-2p_i+p_{i-1} \\ f(s_{i+1})=\frac{p_i+4p_{i+1}+p_{i+2}}{6},\quad f'(s_{i+1})=\frac{p_{i+2}-p_i}{2},\quad f''(s_{i+1})=p_{i+2}-2p_{i+1}+p_i \tag{5.33}$$

Las curvas b-spline cúbicas determinadas por cuatro puntos de control se ilustran en la Figura 5.19, donde se da a *s* los valores de 1, 2, 3 y 4 y los valores de *p* varían de un caso a otro. Puede verse que $f(s)$ no pasa por los puntos de control en los casos A y B. Sin embargo, si tres ordenadas consecutivas son idénticas, como en el caso C, o si las tres ordenadas consecutivas cambian linealmente, como en el caso D, la curva pasa por el punto intermedio de los tres.

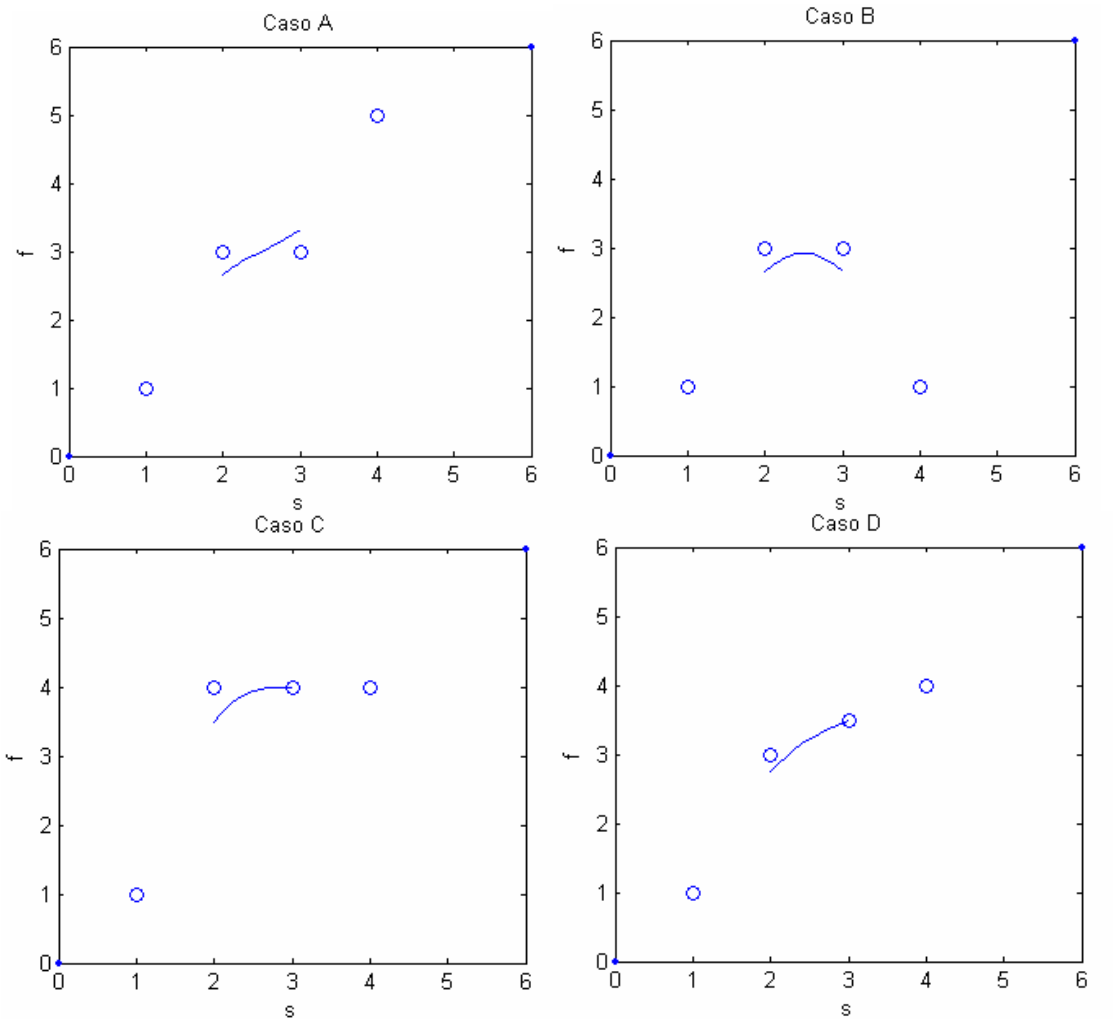


Figura 5.19. Segmentos de b-spline determinados por cuatro puntos de control.

Si el número de puntos de control es mayor que 4, una serie de curvas b-spline cúbicas se convierte en una sola curva, como se ilustra en la Figura 5.20. Tanto la función b-spline como su primera y su segunda derivadas se vuelven continuas.

La curva de la Figura 5.20 no pasa por ninguno de los puntos de frontera. En cambio, si se repite $f = 1$ tres veces al principio de los puntos de control y $f = 2$ tres veces al final, la curva satisfará estas condiciones de frontera, como se ilustra en la Figura 5.21.

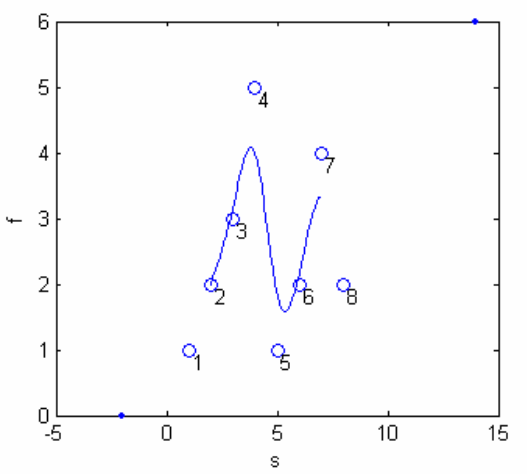


Figura 5.20. Función b-spline determinada por 10 puntos de control.

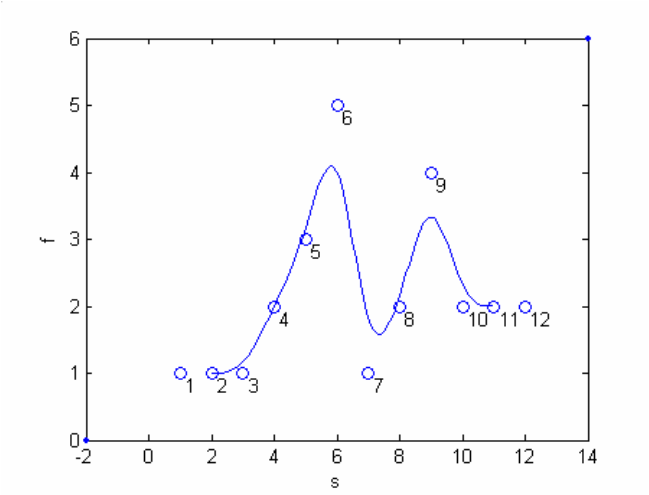


Figura 5.21. La función b-spline satisfaciendo las condiciones de frontera, por medio de la repetición de puntos en los bordes.

5.2.5. Ampliado con base en el algoritmo del asterisco.

El algoritmo del asterisco es un método sencillo de ampliación de imágenes digitales, desarrollado por los autores del Trabajo de Graduación, que se basa en la interpolación lineal para efectuar el proceso de ampliado. Se desarrollaron cuatro tipos diferentes de este algoritmo, los cuales se muestran en la Tabla 5.1. Las diferencias entre los diversos métodos radican en el tipo de determinación de diagonales empleado, así como en el tipo de rellenado.

Tabla 5.1. Variantes del algoritmo del asterisco

Tipo de método	Procesos involucrados	
	Tipo de diagonales	Tipo de rellenado
1	Interpoladas	Espiral
2	Interpoladas	Radial
3	Promediadas	Espiral
4	Promediadas	Radial

El proceso de ampliación realizado por los métodos anteriores consta de los siguientes pasos:

- Creación del enrejado (véase la sección 5.2.1).
- Determinación de las diagonales (véase la sección 5.2.2).
- Proceso de rellenado (véase la sección 5.2.3).

Los métodos de ampliación basados en el algoritmo del asterisco presentan algunas limitantes, las cuales se exponen a continuación.

- Dado que para asignar valores con interpolación lineal a cada uno de los píxeles es necesario que existan dos píxeles originales, los píxeles que se encuentran en la periferia de la imagen ampliada no pueden ser rellenados, ya que sólo se cuenta con uno de los dos píxeles necesarios. Sin embargo, esto resulta en una pequeña línea negra que enmarca a la imagen y que se vuelve apreciable conforme la imagen se amplía un gran número de veces (véase la Figura 5.8 en la sección 5.2.1: Creación del enrejado).
- La imagen ampliada presenta ciertas imperfecciones en la interface entre dos regiones de alto contraste, dando la impresión visual de que ambas regiones han sido unidas por una costura imaginaria. Sin embargo, dependiendo del tamaño de la imagen, estas alteraciones podrían pasar desapercibidas.

En las figuras que siguen se muestran ejemplos de cada uno de los tipos de ampliado con base en el algoritmo del asterisco.

(a) (b)

Figura 5.22. Ejemplo de ampliado con algoritmo del asterisco de tipo 1. (a) Imagen original. (b) Imagen ampliada 15 veces.

(a) (b)

Figura 5.23. Ejemplo de ampliado con algoritmo del asterisco de tipo 2. (a) Imagen original. (b) Imagen ampliada 15 veces.

(a) (b)

Figura 5.24. Ejemplo de ampliado con algoritmo del asterisco de tipo 3. (a) Imagen original. (b) Imagen ampliada 15 veces.

(a) (b)

Figura 5.25. Ejemplo de ampliado con algoritmo del asterisco de tipo 4 (a) Imagen original. (b) Imagen ampliada 15 veces.

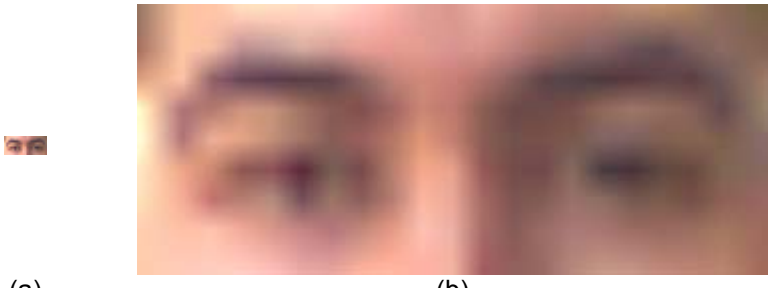
5.2.6. Ampliado con base en interpolación lineal

Al igual que el algoritmo del asterisco, este método utiliza la interpolación lineal para efectuar el proceso de ampliado. Dicho proceso consta de los siguientes pasos:

- Creación del enrejado (véase la sección 5.2.1).
- Proceso de rellenado tipo malla (véase la sección 5.2.3).

Con este método se presentan las mismas dificultades en la orilla de la imagen que con el algoritmo del asterisco, y es que los píxeles ubicados en dicha región no pueden ser interpolados debido a que se necesitan dos puntos para definir una línea recta. Sin embargo, se ha optado por extrapolar dichos valores a partir de la línea recta más cercana a los puntos en cuestión, en lugar de generar un marco negro alrededor de la imagen. Puede pensarse que esta solución podría haberse aplicado también al método con base en el algoritmo del asterisco; sin embargo, ello sólo podría haber sido factible en la construcción del enrejado, pero no en la determinación de las diagonales, puesto que la extrapolación de valores para las diagonales conlleva a resultados que se desvían excesivamente de la tendencia general de la imagen, tal como lo demostraron las pruebas preliminares en este respecto.

La Figura 5.26 muestra un ejemplo de ampliación con interpolación lineal.



(a) (b)

Figura 5.26. Ejemplo de ampliado con interpolación lineal. (a) Imagen original. (b) Imagen ampliada 15 veces.

5.2.7. Ampliado con base en interpolación polinomial

Este método utiliza la interpolación polinomial para efectuar el proceso de ampliado. Dicho proceso consta de los siguientes pasos:

- Creación del enrejado (véase la sección 5.2.1).
- Proceso de rellenado tipo malla (véase la sección 5.2.3).

La interpolación polinomial conlleva un inconveniente para el ampliado de imágenes, y es que los datos interpolados pueden presentar grandes oscilaciones con extremos altos a medida que el número de puntos aumenta. Generalmente una imagen ampliada ha de contener muchos más datos que la imagen original, facilitando la apreciación de dicho inconveniente por parte del observador. Los picos y las crestas de las oscilaciones pueden, con mucho, superar los límites establecidos por el rango numérico de la representación digital de imágenes, esto es, valores desde 0 hasta 256 para la crominancia y la luminancia. Este hecho se manifiesta como parches de colores en la imagen ampliada, puesto que todos los píxeles con valores mayores a 256 se truncan a 256, así como todos los píxeles con valores menores a 0 se truncan a 0. Mientras mayor sea la imagen original o la cantidad de ampliación, los efectos de las oscilaciones por la interpolación polinomial serán más visibles.

Sin embargo, este efecto de generación de picos y valles elevados por causa de las oscilaciones se produce en los extremos de la serie de datos. Por ejemplo, considérese la serie de datos siguiente[16]:

[16] Estos datos fueron obtenidos de una imagen real.

Tabla 5.2. Datos de ejemplo para la interpolación polinomial

x	y	x	y	x	y
1	255	8	239	15	181
2	255	9	255	16	191
3	255	10	255	17	189
4	255	11	243	18	183
5	249	12	164	19	179
6	224	13	145	20	181
7	217	14	153	21	166

La Figura 5.27 muestra la gráfica de los datos anteriores, los cuales podrían constituir los valores de los píxeles de una fila en una imagen a ser ampliada.

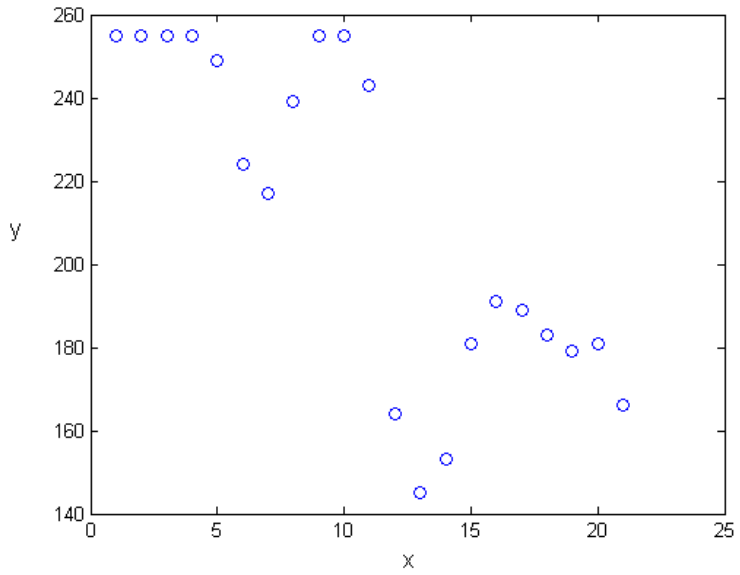


Figura 5.27. Gráfica de los datos de la Tabla 5.2.

Como se dijo anteriormente, los picos y valles elevados producto de las oscilaciones en la interpolación polinomial aparecen en los extremos de la serie de datos. Si se grafican los datos interpolados, excluyendo varios puntos en los extremos de la serie, la curva interpolada parece ajustarse satisfactoriamente a la tendencia general de los datos, tal como se puede observar en la Figura 5.28. Sin embargo, el resultado de la interpolación polinomial para los extremos de la curva dista mucho del comportamiento mostrado por los datos (Figura 5.29).

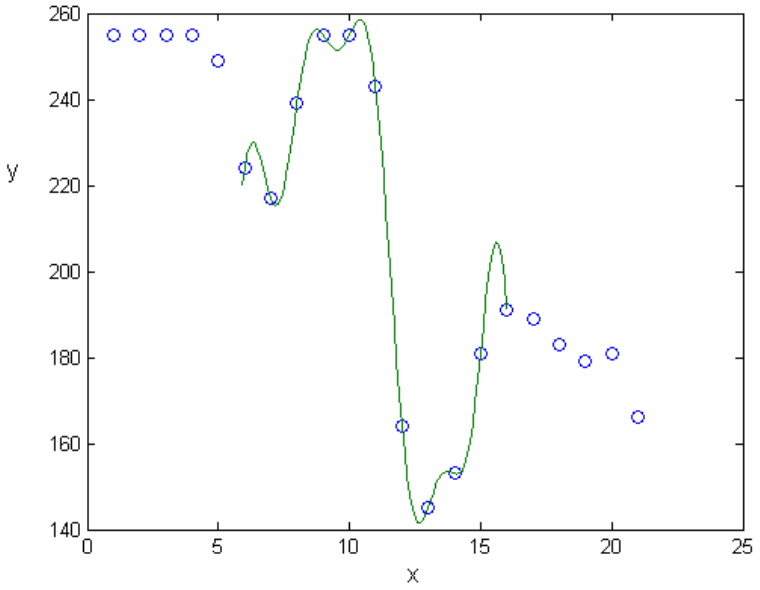


Figura 5.28. Datos de la Tabla 5.2 (círculos vacíos) y curva ajustada por interpolación polinomial (línea continua) excluyendo los extremos de la misma.

Las curvas de la Figura 5.28 y de la Figura 5.29 no lucen similares (aunque sus valores son idénticos) debido a la escala vertical utilizada para darle cabida a los picos y valles generados cerca de los extremos (obsérvese el multiplicador de "$x10^4$" en el eje vertical). Significa entonces que en dichas regiones se generaron valores tan altos como 500, y tan bajos como –5000, violando definitivamente el rango de datos de 0 a 256.

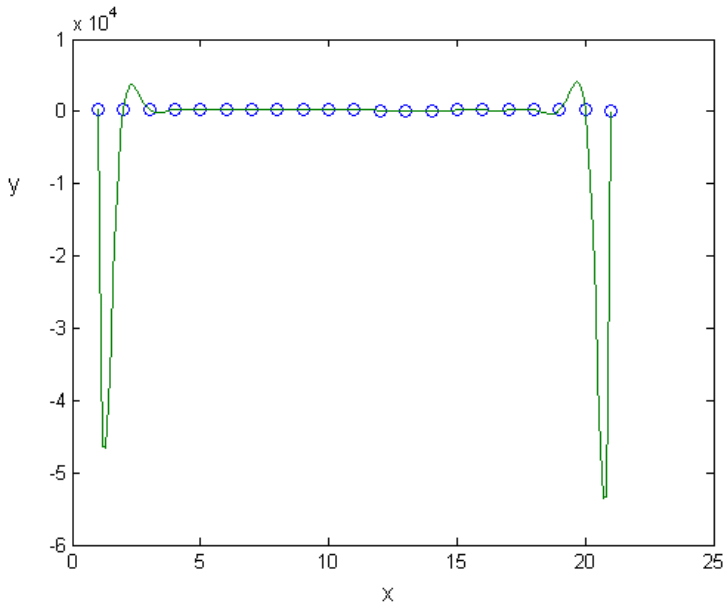


Figura 5.29. Datos de la Tabla 5.2 (círculos vacíos) y curva ajustada por interpolación polinomial (línea continua).

Por lo tanto, las imágenes ampliadas con interpolación polinomial, tal como se ha descrito el método, resultarían inservibles para imágenes grandes o cantidad de ampliación alta, reduciendo enormemente la utilidad del proceso. Una forma de aminorar estos efectos negativos es añadiendo cierta cantidad de datos de relleno al inicio y al final de la serie, de manera que los puntos que sí conforman los datos reales se encuentren lejos de las orillas, y no sean alcanzados por los picos y valles producto de las oscilaciones.

Si bien esta medida puede proporcionar una solución satisfactoria, su desempeño está sujeto a dos factores:

- La cantidad de puntos añadidos al inicio y al final de la serie de datos.
- El tiempo de procesamiento utilizado en interpolar los datos de relleno.

A raíz de una serie de experimentos realizados por los autores puede afirmarse lo siguiente con respecto a los dos factores mencionados:

- A mayor cantidad de datos de relleno, mejor el ajuste que proporciona la curva a la tendencia general de los datos (hasta cierto límite[17]).
- El tiempo de procesamiento puede alcanzar valores intolerables o imprácticos si el número de datos de relleno es demasiado alto.

De inmediato se pone de manifiesto que es necesario establecer una relación de compromiso entre ambos factores, puesto que el aumentar los datos de relleno también aumenta el tiempo de procesamiento. Así que la solución adoptada en este Trabajo de Graduación ha sido la de tomar como datos de relleno en cada extremo un número igual de puntos que el tamaño de la serie, de forma que la nueva serie de datos sea tres veces más larga. Los datos añadidos al inicio son todos iguales, e iguales al primer valor de la serie original; situación similar ocurre en el extremo final de la serie, pero ahora, iguales al último punto de la serie original.

En la figura siguiente se muestra la gráfica de los datos de la Tabla 5.2, pero ahora aplicando la solución recién expuesta.

[17] Los experimentos realizados para determinar dicho límite, es decir, un número de datos de relleno tal que un incremento adicional en la cantidad de los mismos no produzca un cambio substancial en la tendencia de la curva, no produjeron resultados concluyentes, puesto que dicho límite varía en función del número de datos originales y de la dispersión existente entre éstos. Adicionalmente, el beneficio de la inclusión de datos de relleno se ve disminuido ante la presencia de varios datos consecutivos iguales.

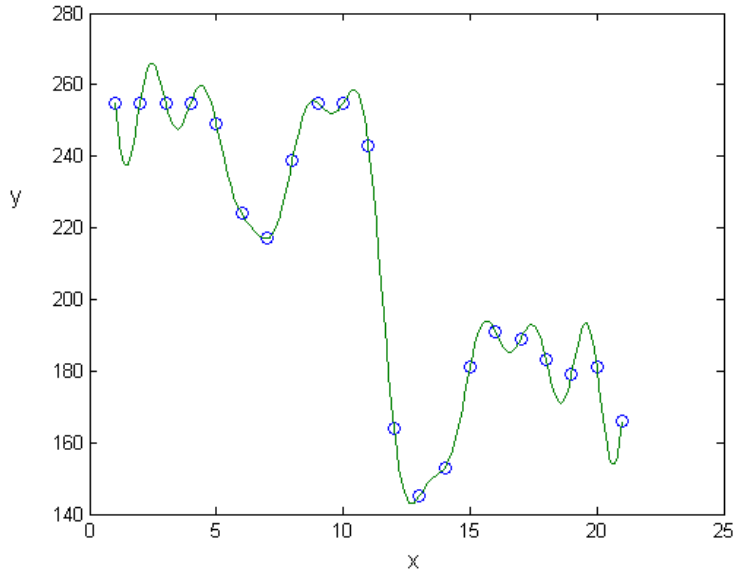


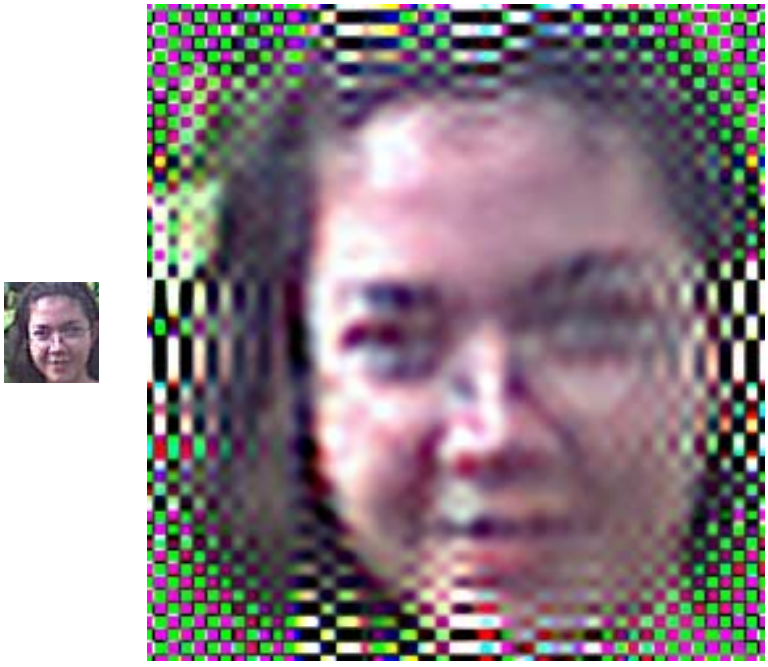
Figura 5.30. Datos de la Tabla 5.2 (círculos vacíos) y curva ajustada por interpolación polinomial corregida (línea continua).

Adicionalmente a las dificultades presentadas debido a las oscilaciones producto de la interpolación polinomial, existe una limitante ampliamente conocida de dicho método de interpolación, y es precisamente que su utilidad no contempla la extrapolación de valores. A medida que se extrapolan valores más y más lejanos de la serie de datos que dio origen al polinomio, los resultados obtenidos se alejan más de la tendencia general de los datos. Sin embargo, para puntos de extrapolación cercanos a los extremos, los resultados podrían ser aceptables, si bien aún se alejan de la tendencia de la curva interpolada. Es por ello que, en lugar de generar un marco negro en la periferia de la imagen, tal como lo hace el método de ampliación basado en el algoritmo del asterisco, se ha optado por extrapolar dichos valores, permitiendo la generación de la imagen completa, a expensas de ligeras variaciones en los colores de la misma, las cuales son visualmente identificadas como imperfecciones en la imagen. La Figura 5.31 muestra un ejemplo de imagen ampliada con el método basado en interpolación polinomial.

(a) (b)

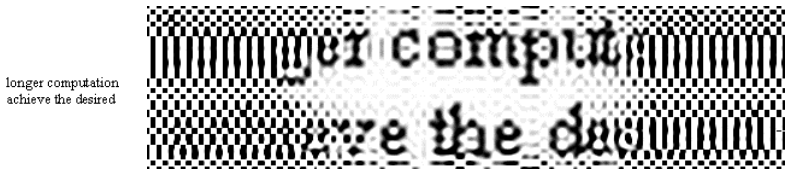
Figura 5.31. Ejemplo de ampliado con interpolación polinomial. (a) Imagen original. (b) Imagen ampliada 15 veces.

Puede observarse en la imagen anterior las imperfecciones en los colores producidas por las oscilaciones de las curvas interpoladas. Si bien en esta imagen tales imperfecciones podrían llegar a ser aceptables (puesto que no impiden sobremanera el reconocimiento de los objetos de la misma por parte del observador), existen casos en donde definitivamente las oscilaciones no pueden ser permitidas, debido a que causan la destrucción de una buena parte de la información contenida en la imagen. Por ejemplo, considérese el caso mostrado en la Figura 5.32 y en la Figura 5.33.



(a) (b)

Figura 5.32. Ejemplo de ampliado con interpolación polinomial cuando las oscilaciones son intolerables. (a) Imagen original. (b) Imagen ampliada 5 veces.



(a) (b)

Figura 5.33. Otro ejemplo de ampliado con interpolación polinomial cuando las oscilaciones son intolerables. El texto escaneado original es "longer computation achieve the desired". (a) Imagen original. (b) Imagen ampliada 5 veces.

En dichas imágenes, los efectos de las oscilaciones han sido tan severos que un buen porcentaje de éstas es ininteligible. Ello pone de manifiesto la necesidad de optar por una solución más satisfactoria al problema de las oscilaciones en la interpolación polinomial. Dicha solución se ha implementado con el método de interpolación polinomial adaptativa, que se describe en la sección siguiente.

Interpolación polinomial adaptativa[18]

La solución que se propone en este Trabajo de Graduación al problema de las oscilaciones generadas con la interpolación polinomial puede deducirse al observar la serie de imágenes que se muestran en la Figura 5.34.

[18] Este método es una implementación original de los autores del Trabajo de Graduación.

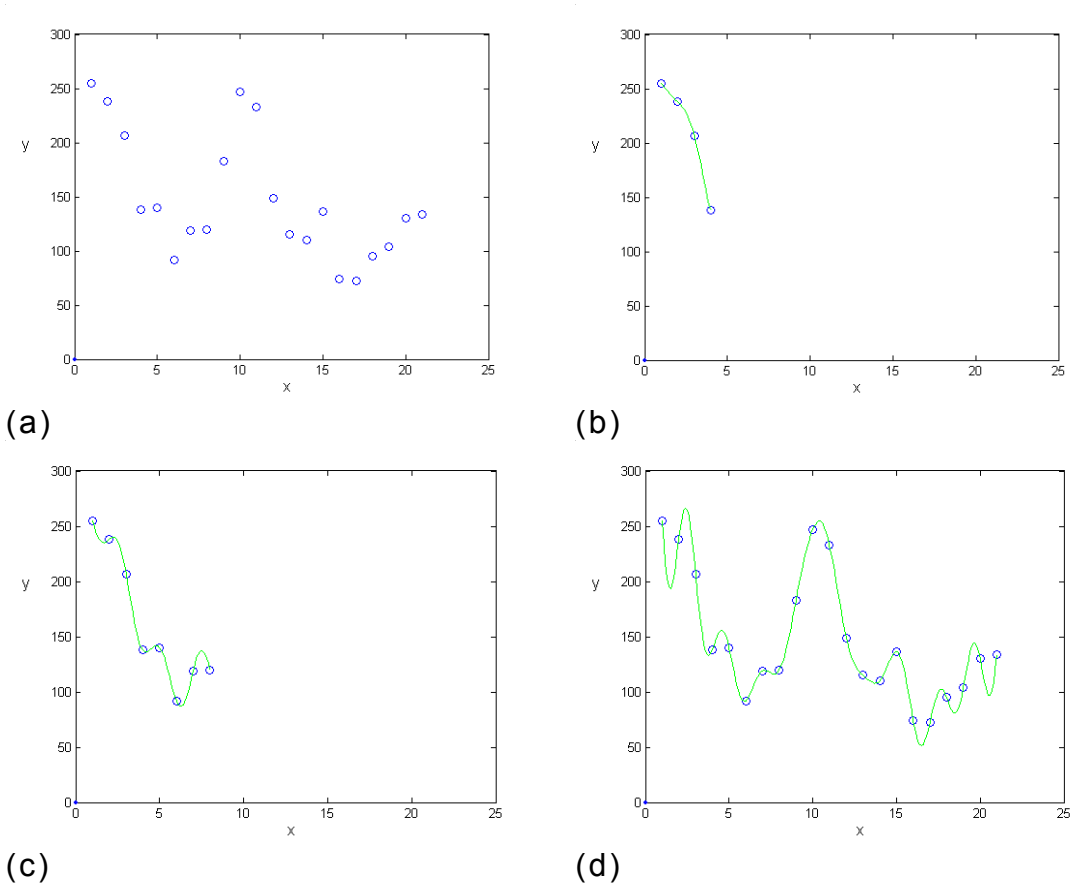


Figura 5.34. Origen de las oscilaciones en la interpolación polinomial. (a) Datos originales. (b) Curva interpolando 4 puntos. (c) Curva interpolando 8 puntos. (d) Curva interpolando todos los puntos.

En la figura anterior puede observarse que cuando se interpolan pocos puntos de datos (b), el polinomio no se desvía significativamente de la tendencia general de los mismos. Sin embargo, a medida que se agrega una mayor cantidad de puntos, dicha desviación crece, hasta el grado de hacerse inaceptable. Así, en (c), la curva que une los mismos cuatro puntos que en (a) comienza a desfigurarse, y en (d) ya se originan oscilaciones en ellos. De manera que la presencia de las oscilaciones puede significar que el polinomio ya no es válido para una cantidad de puntos dada. Por lo tanto, la solución consistiría en encontrar tantos polinomios como sea necesario, cada uno ajustado a una cantidad de puntos tales que la curva interpolada no produzca oscilaciones. Por ejemplo, podría establecerse que para los primeros cuatro puntos en (a), el polinomio adecuado es el que genera la curva

en (b). A partir del cuarto punto debería determinarse otro polinomio, y así sucesivamente.

Ahora el problema radica en definir un criterio para decidir cuándo las curvaturas de un polinomio constituyen una oscilación indeseable, y cuándo se trata simplemente de las regiones cóncavas o convexas presentes en toda curva. Es decir, la forma del trazo mostrado en (b) es curvo, y no por ello constituye una oscilación; no así el trazo mostrado en (d). El criterio utilizado en este Trabajo de Graduación establece que si la curva interpolada entre dos puntos de datos dados se sale en algún punto de la región de aceptación mostrada en la Figura 5.35, se dice que el polinomio en cuestión no es válido.

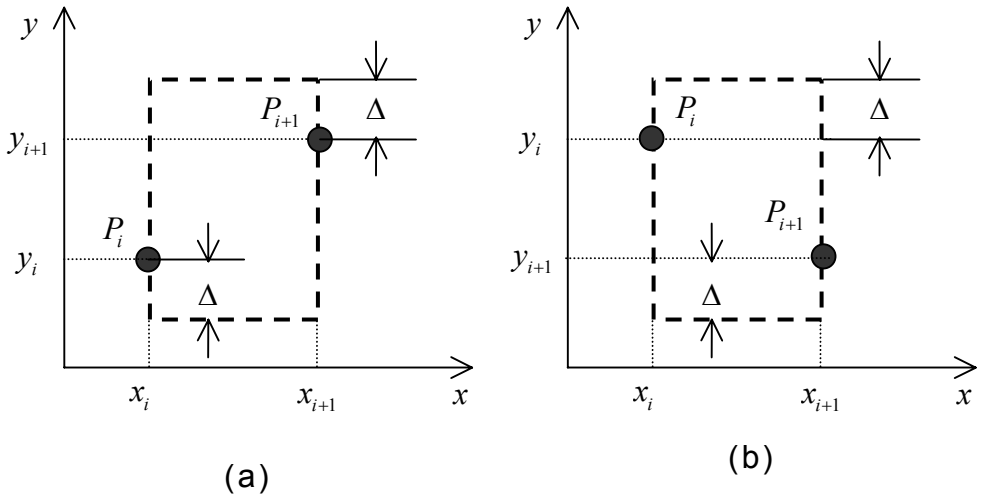


Figura 5.35. Definición de la región de aceptación.

La región de aceptación se define entre dos puntos de datos consecutivos $P_i(x_i) = y_i$ y $P_{i+1}(x_{i+1}) = y_{i+1}$ como la región rectangular cuyas esquinas superiores izquierda y derecha tienen coordenadas $(x_i, \max(y_i, y_{i+1}) + \Delta)$ y $(x_{i+1}, \max(y_i, y_{i+1}) + \Delta)$, respectivamente; y con esquinas inferiores izquierda y derecha con coordenadas iguales a $(x_i, \min(y_i, y_{i+1}) - \Delta)$ y $(x_{i+1}, \min(y_i, y_{i+1}) - \Delta)$ respectivamente. Aquí, Δ es un valor de tolerancia que controla el valor máximo en el que puede desviarse la curva de los puntos de datos extremos. Las palabras $\max$ y

$\min$ aluden a las funciones máximo y mínimo, respectivamente, y se han utilizado para incluir tanto el caso en el que $y_{i+1} > y_i$ como el caso contrario (Figura 5.35 a y b). El valor de Δ influye en el número de polinomios necesarios para representar toda una serie de datos; a mayor valor de Δ, mayor tolerancia a las oscilaciones. Su valor se ha fijado igual a 1, debido a que los valores de los píxeles están cuantificados, lo que significa que, por ejemplo, tanto 155, 155.42 y 155.99, se truncan a 155 en una imagen digital. Así que, si se permiten ligeramente las oscilaciones con $\Delta = 1$, se necesitarán menos polinomios para representar toda la serie de datos, conllevando a que probablemente se necesite menos tiempo de procesamiento, sin que esto repercuta significativamente en los valores de los píxeles interpolados.

Finalmente, es necesario aclarar que los diversos polinomios que interpolan la serie de datos no son dependientes entre sí, por lo que no debe esperarse ningún tipo de continuidad en los puntos de conexión de éstos. En general, puede afirmarse que existirá un cambio abrupto en la pendiente de la curva en los puntos en los que se da un cambio de polinomio. Si bien esto dista mucho de los objetivos buscados con la interpolación cúbica segmentaria, ello puede ayudar a que la imagen no aparezca borrosa. En efecto, el requisito de continuidad en los tonos de color solamente es aplicable a las regiones de un mismo objeto dentro de una imagen. Por otro lado, en las fronteras de dichas regiones se espera que el cambio de color o tonalidad sea brusco, puesto que ello es responsable de la facilidad con la que el observador identifica los diversos elementos contenidos en la escena. Por lo tanto, es de esperarse que este método contribuya a los aspectos antes mencionados. En la figura siguiente se muestran los datos de la Figura 5.34 (a) ajustados por una curva obtenida con interpolación polinomial adaptativa. Los círculos rellenos indican los puntos en donde termina un polinomio e inicia el siguiente.

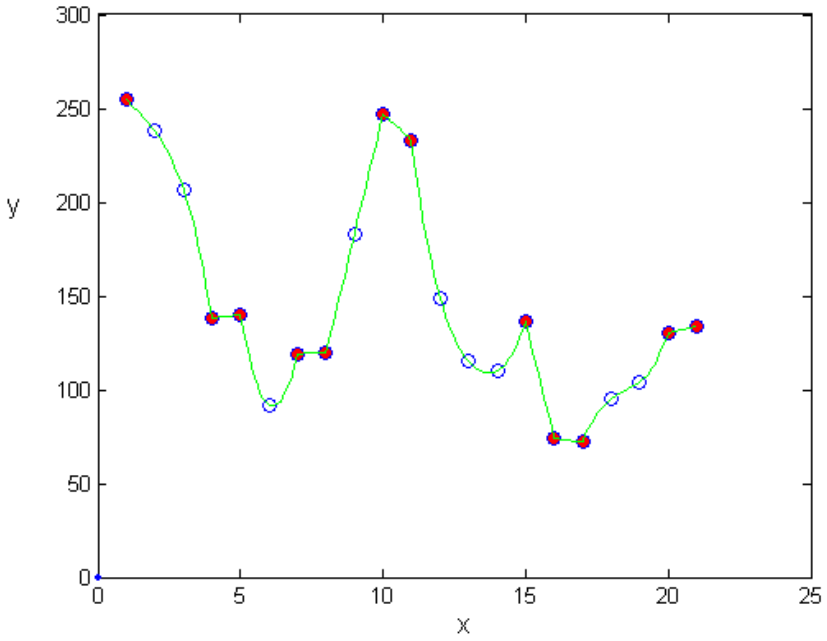
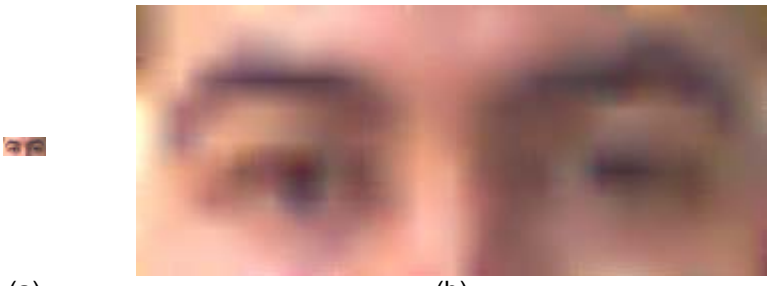


Figura 5.36. Ejemplo de interpolación polinomial adaptativa.

En la Figura 5.37 se muestra un ejemplo de imagen ampliada con interpolación polinomial adaptativa.



(a) (b)

Figura 5.37. Ejemplo de ampliado con interpolación polinomial adaptativa.
(a) Imagen original. (b) Imagen ampliada 15 veces.

5.2.8. Ampliado con base en interpolación segmentaria

Como se dijo en la sección 5.2.4 Métodos de interpolación, las funciones spline presentan dos variantes que fueron aprovechadas para la elaboración de métodos de ampliación, siendo éstas las funciones c-spline y las b-spline. En ambos casos, el proceso de ampliación consta de las siguientes partes:

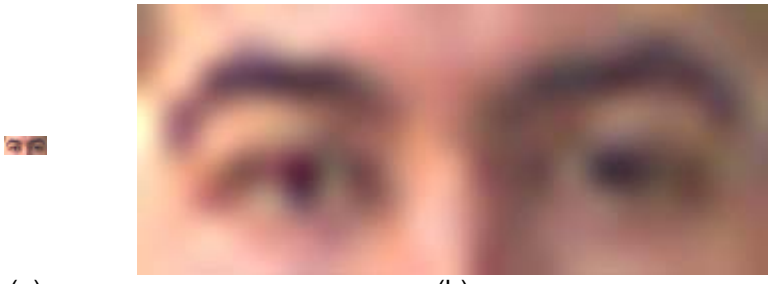
- Creación del enrejado (véase la sección 5.2.1).
- Proceso de rellenado tipo malla (véase la sección 5.2.3).

En el proceso de ampliación con c-splines ocurre un fenómeno similar al que sucede con la interpolación polinomial en cuanto a la extrapolación se refiere, y es que en la periferia de la imagen no existen datos adicionales con los cuales generar los polinomios cúbicos segmentarios[19]. Para solventar esta dificultad, se ha preferido duplicar el primero y último valor de cada serie de datos, de manera que siempre exista un par de puntos con el cual definir un polinomio cúbico segmentario. También pudo haberse optado por extrapolar los valores a partir del polinomio más cercano. Sin embargo, esto ocasionaba que los píxeles extrapolados se alejaran de la tendencia general de la imagen, produciendo alteraciones visibles en la calidad de la misma.

Por otro lado, en el caso de las funciones b-spline, ocurre una situación más particular, y es que debido a que dichas funciones utilizan cuatro puntos para interpolar una curva entre dos de ellos, los valores entre el primero y el segundo, y entre el penúltimo y el último de la serie de datos no están definidos. Para solventar esta situación se optó por la misma solución que para el caso con c-splines, es decir, repetir el primero y último puntos de cada serie de datos, de manera que siempre exista un cuarteto de puntos con los que se pueda definir el polinomio cúbico segmentario para b-splines.

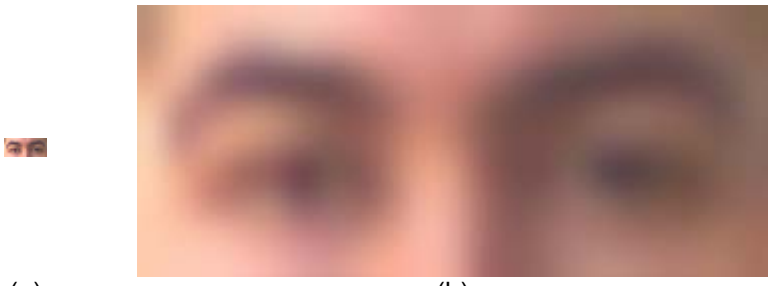
Las figuras que siguen muestran algunos ejemplos de ampliación utilizando los métodos basados en c-splines y b-splines.

[19] Recuérdese que dichos polinomios se definen entre dos puntos de datos consecutivos.



(a) (b)

Figura 5.38. Ejemplo de ampliado con interpolación basada en c-splines. (a) Imagen original. (b) Imagen ampliada 15 veces.



(a) (b)

Figura 5.39. Ejemplo de ampliado con interpolación basada en b-splines. (a) Imagen original. (b) Imagen ampliada 15 veces.

5.3. Métodos de reducción

Adicionalmente a los métodos de ampliado descritos, se desarrollaron dos métodos de reducción, cuya función es la de disminuir las dimensiones de una imagen en un factor especificado. Si bien el propósito de este Trabajo de Graduación es la ampliación de imágenes, estos métodos de reducción se han añadido como herramientas complementarias e ilustrativas para el lector interesado.

Debe aclararse que no todos los métodos de ampliación descritos son reversibles, puesto que en el proceso pudo haberse perdido la información de los píxeles originales debido a la interpolación. Tal es el caso de las funciones b-spline. Por otro lado, es posible que se desee reducir las dimensiones de una imagen que no ha sido previamente ampliada con alguno de los métodos estudiados. Es por ello que se han desarrollado dos formas diferentes de realizar la reducción de las dimensiones, las cuales se describen a continuación.

5.3.1. Reducción simple

Este método es exactamente el proceso inverso del ampliado simple descrito en la sección 5.1.1, y consiste en colocar el píxel central de cada celda de la imagen ampliada en el píxel correspondiente en la imagen reducida. En la Figura 5.40 se muestra un ejemplo del proceso de reducción simple para una imagen de 3x4 celdas, cada una de 3x3 píxeles. Las celdas siempre serán de **n** x **n** píxeles, siendo *n* la cantidad de reducción. Debe aclararse que si el número de filas o de columnas de la imagen a reducir no es suficientemente alto como para dar cabida a un número entero de celdas, la imagen original será recortada antes de ser reducida, de forma que el número de celdas que la conformen sea entero.

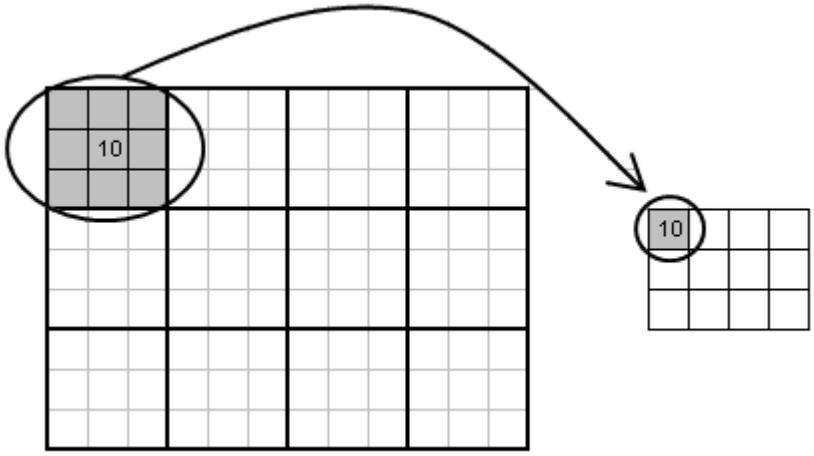


Figura 5.40. Ejemplo de reducción simple.

5.3.2. Reducción con promedio

La otra forma de efectuar la reducción de una imagen descrita en este documento es muy similar a la reducción simple recién expuesta, excepto porque el píxel que se coloca en la imagen reducida no es el píxel central de cada celda, sino el promedio de todos los píxeles que la conforman (por ejemplo, todos los píxeles en gris en la Figura 5.40).

6. DISEÑO DE LA APLICACIÓN

La aplicación de usuario final debe ser capaz, principalmente, de lo siguiente:

- Ampliar imágenes digitales por diversos métodos.
- Proporcionar al usuario final facilidades para el aprovechamiento de sus funciones.

La primera de las características implica el uso de algoritmos y de procesos matemáticos, tal como se ha descrito en secciones anteriores. Por otro lado, el segundo aspecto conlleva al uso de una interfaz gráfica de usuario, que proporcione mecanismos para acceder a las capacidades del programa fácilmente, así como permita la selección de las imágenes a ser tratadas, y su visualización antes y después del proceso de ampliación, entre otras funciones.

Si bien todas estas características pueden ser implementadas utilizando una única herramienta de desarrollo de software, debe reconocerse que existen herramientas que presentan mayor facilidad para llevar a cabo algún conjunto común de tareas, y que la experiencia del programador en el uso de dichas herramientas puede estar más orientada hacia algunas de ellas, que hacia otras.

Dado que las imágenes digitales se representan numéricamente como matrices, y que los algoritmos de los diversos métodos de ampliación estudiados requieren de operaciones aplicadas a un gran número de datos, es recomendable utilizar una herramienta de programación que facilite tales procedimientos. Uno de dichos programas es MATLAB[20], el cual ha sido elegido para desarrollar la

[20] MATLAB es una marca registrada de The MathWorks Inc, Estados Unidos.

parte de la aplicación que se encarga de ampliar las imágenes utilizando los diversos métodos expuestos.

Por otro lado, la interfaz gráfica de usuario puede ser desarrollada fácilmente con un lenguaje de programación orientado a eventos, tal como Visual Basic,[21] siendo éste el programa elegido para dicho propósito.

En este punto, aún es necesario establecer la forma en la que se vincularán los programas desarrollados en ambas herramientas para formar una aplicación consolidada. Una forma de llevar a cabo dicha vinculación es mediante la creación de un programa ejecutable en MATLAB, que se invoque desde la aplicación en Visual Basic. Otra manera podría ser por medio de la elaboración de una biblioteca enlazada dinámicamente (dynamically linked library, o DLL) de la aplicación en MATLAB, y utilizarla desde el entorno de desarrollo de Visual Basic. La solución adoptada es esta última. Finalmente, la aplicación final consiste de un programa ejecutable, diseñado y compilado en Visual Basic, que utiliza la biblioteca del programa creado en MATLAB. También se hace necesaria la creación de un paquete de instalación de la aplicación, puesto que ésta utiliza bibliotecas que deben ser registradas en el sistema operativo para su correcto funcionamiento.

El esquema general que se ha seguido para realizar la aplicación de usuario final es el siguiente:

- Creación de los archivos M en MATLAB[22] responsables de los diversos métodos de ampliación estudiados.

---

[21] Visual Basic es una marca registrada de Microsoft Corporation.

[22] Los archivos M son archivos de texto que contienen código en el lenguaje de programación de MATLAB. Para mayor información, consúltese [10] u [11].

- Elaboración de una biblioteca enlazada dinámicamente (DLL) a partir de los archivos M creados y de las funciones de MATLAB utilizadas.
- Diseño de la interfaz gráfica de usuario en Visual Basic, utilizando la biblioteca del programa creado en MATLAB para realizar el procesamiento de las imágenes.
- Elaboración del programa ejecutable y del paquete de instalación de la aplicación final de usuario, en Visual Basic.

Las diversas herramientas de software utilizadas se resumen en la Tabla 6.1.

Tabla 6.1. Herramientas de software utilizadas para el desarrollo de la aplicación de usuario final

Herramienta	Fabricante	Propósito
MATLAB 6.5 Release 13	The MathWorks Inc.	Crear archivos M para el ampliado de imágenes
MATLAB COM Builder 1.0	The MathWorks Inc.	Crear biblioteca enlazada dinámicamente y su instalador
Visual Basic 6.0	Microsoft Corporation	Crear interfaz gráfica de usuario
Visual C++ 6.0	Microsoft Corporation	Crear compilaciones de C/C++ para MATLAB COM Builder
Visual Studio Installer 1.1	Microsoft Corporation	Crear paquete de instalación
Visual Interdev 6.0	Microsoft Corporation	Crear paquete de instalación
Service Pack 5 para Visual Studio 6.0	Microsoft Corporation	Crear paquete de instalación

En los anexos se describen separadamente los pasos recién mencionados para la elaboración de la aplicación de usuario final.

6.1. Requerimientos del sistema

La aplicación de usuario final, entendida como un programa ejecutable en ambiente gráfico para la ampliación de imágenes digitales, que utiliza una biblioteca enlazada dinámicamente para realizar los procesos matemáticos, no requiere de ninguna aplicación además del sistema operativo Windows. Lo único que un usuario necesita para utilizar esta aplicación es haber ejecutado satisfactoriamente el paquete de instalación de la misma, disponible en

el disco compacto que acompaña a este documento. Sin embargo, para el correcto funcionamiento de la aplicación, es necesario garantizar los requerimientos de hardware y de sistema operativo tanto de la aplicación desarrollada en Visual Basic como del DLL compilado con MATLAB COM Builder. Dado que la biblioteca utiliza funciones y algoritmos de MATLAB 6.5 R13, es lógico establecer que los requerimientos mínimos de la aplicación sean al menos similares a los de dicha herramienta de programación. Ocurre la misma situación con los requerimientos por parte del archivo ejecutable que constituye a la aplicación de usuario final, puesto que deberán ser similares a los de cualquier aplicación compilada con Visual Basic 6.0. Por lo tanto, los requerimientos mínimos del sistema se fijarán como los más exigentes de estos dos grupos. Luego de una revisión de la documentación de ambas herramientas de desarrollo de software, se concluye que los requerimientos de MATLAB son los más exigentes, por lo que deben garantizarse para el correcto funcionamiento de la aplicación de usuario final. Dichos requerimientos mínimos se detallan a continuación (se asume que el sistema operativo es Microsoft Windows):

Tabla 6.2. Requerimientos mínimos del sistema

<table>
<tr><th>Sistema</th><th>Procesadores</th><th>Espacio en disco</th><th>RAM</th></tr>
<tr><td>XP</td><td rowspan="5">Pentium, Pentium Pro, II, III, IV, Xeon PIII, AMD Athlon, Athlon XP</td><td rowspan="5">90MB</td><td rowspan="5">128MB (mínimo)
256MB (recomendado)</td></tr>
<tr><td>2000 (Service Pack 1 ó 2)</td></tr>
<tr><td>NT 4.0 (Service Pack 5, 6, ó 6a)</td></tr>
<tr><td>ME</td></tr>
<tr><td>98 (Original Edition y Second Edition)</td></tr>
</table>

Por otro lado, si lo que se desea es probar o editar los archivos M que dieron origen a la biblioteca, se necesita la versión 6.5 R13 de MATLAB, y deben satisfacerse los requerimientos mínimos para dicha herramienta de software. Para mayor información, consúltese http://www.mathworks.com/products/system.shtml/Windows. Adicionalmente, si

se desea utilizar la biblioteca desde el ambiente de desarrollo de Visual Studio, o si se pretende probar o editar el código fuente de la aplicación de usuario final en Visual Basic, será necesario satisfacer los requerimientos mínimos para dichas herramientas de software (además de contar con dichas aplicaciones). Consúltese [12] para mayor información.

7. EVALUACIÓN DE LOS MÉTODOS DE AMPLIADO

Los diversos métodos de ampliación desarrollados deben ser evaluados con el objeto de identificar sus características, medir su desempeño, y establecer elementos de juicio para fundamentar la selección de uno u otro método para la ampliación de un tipo específico de imágenes.

7.1. Criterios de evaluación

Se han establecido dos criterios para la evaluación; uno de carácter cualitativo, y otro de carácter cuantitativo. El criterio cualitativo lo constituye la apariencia general de la imagen luego de haber sido procesada por algún método de ampliación, mientras que el segundo criterio se basa en el tiempo de procesamiento requerido por cada método al realizar diversas ampliaciones, según se describe más adelante. Las pruebas de ampliación se realizaron sobre tres imágenes, cada una perteneciente a uno de los grupos siguientes:

- Fotografías
- Dibujos
- Texto

Los criterios de evaluación empleados se describen a continuación.

7.1.1. Apariencia de la imagen

Con este criterio se pretende evaluar la calidad de la imagen ampliada, en términos de su semejanza con la imagen original, y de la existencia de características indeseables en la misma, tales como distorsión o alteraciones observables no presentes en la imagen original.

7.1.2. Tiempo de procesamiento

Para la evaluación del tiempo de procesamiento de los diversos métodos se han considerado los siguientes aspectos:

- El tiempo empleado para la ejecución de un programa en una computadora específica no siempre es el mismo, aún si los procesos involucrados son iguales, debido a que el programa en cuestión debe compartir la atención del microprocesador y el uso de memoria con las variadas aplicaciones y procesos que se encuentren en ejecución en la computadora en ese momento.
- El orden en el que se ejecuten los diversos programas en una computadora puede influir en el tiempo de procesamiento de éstos, debido a las alteraciones en la distribución de memoria que pudieron haber sido ocasionadas por los programas precedentes.
- Las características del hardware de la computadora en la que se ejecuten los programas, tales como la frecuencia de reloj del microprocesador y la cantidad de memoria RAM, entre otras, constituyen un factor determinante para el tiempo de procesamiento.

En las pruebas realizadas para la evaluación de los métodos de ampliación se ha intentado considerar en alguna medida los aspectos antes mencionados, de forma que la comparación de los métodos sea más justa y rica en información.

7.2. Pruebas realizadas

Todas las pruebas se aplicaron a tres imágenes digitales de mapas de bits diferentes, en formato RGB, con el mismo número de filas y de columnas (40x40), pero pertenecientes a tres categorías distintas: una fotografía, un dibujo, y un fragmento de texto escaneado.

7.2.1. Prueba de apariencia

Esta prueba consistió en la aplicación de los diez métodos de ampliación desarrollados a las imágenes de prueba. La cantidad de ampliación fue constante e igual a 6 para todos los métodos. Los resultados obtenidos se muestran en el ANEXO A y se discuten en la sección 7.3.1.

7.2.2. Pruebas de tiempo de procesamiento

El objetivo de estas pruebas consiste en permitir la comparación de los diversos métodos estudiados en base al tiempo que demoran en realizar el proceso de ampliado. Por lo tanto, debe intentarse asegurar que todos los métodos evaluados se ejecuten bajo de las mismas condiciones, considerando los aspectos mencionados en la sección 7.1.2. Ello implica la realización de un gran número de pruebas, por lo que resultaría engorroso evaluar los diez métodos de ampliación desarrollados. Como se verá en la sección 7.3.1, algunos de los métodos presentan resultados muy semejantes, mientras que otros no proporcionan una calidad comparable. Por lo tanto, la evaluación del tiempo de procesamiento se realizará empleando solamente el mejor representante de todos los métodos basados en un mismo tipo de interpolación, como se muestra en la Tabla 7.1.

Tabla 7.1. Métodos evaluados por cada tipo de interpolación

Tipo de interpolación	Métodos disponibles	Método evaluado	Nº
Lineal	Cuatro versiones del algoritmo del asterisco	Interpolación lineal	5
	Interpolación lineal		
Polinomial	Interpolación polinomial	Interpolación polinomial adaptativa	9
	Interpolación polinomial adaptativa		
Segmentaria	C-splines	C-splines	7
	B-splines		

El esquema utilizado para realizar dichas pruebas es como sigue:

- Para cada imagen de prueba (una fotografía, un dibujo y un texto escaneado) se evaluarán los tres métodos seleccionados.
- Por cada método seleccionado, se realizarán dos modalidades de prueba, cada una con tres niveles de dificultad:
  - Para la modalidad 1, se mantendrá constante el tamaño de la imagen a ser ampliada, y se aumentará la cantidad de ampliación. Se emplearán tres factores de ampliación distintos.
  - Para la modalidad 2, se mantendrá constante la cantidad de ampliación, pero se aumentará el tamaño de la imagen a ser ampliada. Se emplearán tres tamaños diferentes.
- Para la modalidad 2 se tomará como imagen de entrada a la imagen de salida obtenida con el nivel de dificultad inmediato inferior, excepto para el primer nivel de dificultad, para el cual se tomará como imagen de entrada a la imagen de prueba original.
- Cada prueba individual se realizará tres veces, y el valor de tiempo de procesamiento registrado será el promedio de los tres tiempos obtenidos.

Para permitir la comparación de los resultados obtenidos por ambas modalidades de prueba, debe asegurase que la imagen resultante contenga el mismo número de píxeles para cada nivel de dificultad. Los factores de ampliación (n) y el tamaño del área de las imágenes de entrada (A) para ambas modalidades se resumen en la Tabla 7.2.

Tabla 7.2. Factores de ampliación y área de las imágenes de entrada y salida para los tres niveles de dificultad en las dos modalidades de prueba. Las unidades de área son píxeles cuadrados

Nivel de dificultad	Modalidad 1		Modalidad 2		Área de la imagen ampliada
	n	**A**	**n**	**A**	
Bajo	3	40x40	3	40x40	14,400
Medio	9	40x40	3	120x120	129,600
Alto	27	40x40	3	360x360	$1_1$166,400

- La serie completa de pruebas será realizada en dos computadoras diferentes, para poner de manifiesto la influencia de los recursos de hardware y el tipo de sistema

operativo utilizado sobre los resultados de las mismas. Las características de ambas computadoras se muestran en la Tabla 7.3.

Tabla 7.3. Características de las computadoras utilizadas para las pruebas.

Característica	Computadora A (PC A)	Computadora B (PC B)
Reloj de CPU	2.0 GHz	700 MHz
Memoria RAM	512 MB	192 MB
Sistema Operativo	Windows 98	Windows XP Edición Profesional

- Con el fin de que cada modalidad de prueba, para cada método, para cada imagen, se realice en condiciones semejantes, luego de realizar las pruebas correspondientes a cada modalidad se reiniciará la computadora.
- Debido a lo mencionado en la sección 7.1.2 con respecto a la influencia del orden de ejecución de los programas sobre el tiempo de procesamiento de éstos, las pruebas para cada modalidad, para cada método, se realizarán siempre en orden creciente de dificultad.
- Durante la realización de las pruebas, la única aplicación de usuario en ejecución en la computadora será MATLAB, para evitar la carga del microprocesador con otros procesos[23].

Finalmente, en la Tabla 7.4 se muestra un esquema que resume la forma en la que se realizarán las pruebas de tiempo de procesamiento para cada computadora (PC A y PC B). Las filas rellenas indican los momentos en los que se reiniciará el sistema. Se asume que al inicializar el sistema por primera vez, no se ejecutarán otras aplicaciones, excepto las necesarias para realizar las pruebas. Los métodos de interpolación empleados son los que se listan en la Tabla 7.1.

[23] Esto se refiere a aplicaciones ejecutadas explícitamente por el usuario, no a la diversidad de procesos en ejecución pertenecientes al sistema operativo o a aplicaciones autoejecutables.

Tabla 7.4. Formato para el registro de tiempos de procesamiento

Tipo de imagen	Método de ampliación	Modalidad	Niveles de dificultad	Tiempos	
				PC A	PC B
Fotografía	5	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		
	REINICIAR PC				
	7	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		
	REINICIAR PC				
	9	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		
REINICIAR PC					
Dibujo	5	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		
	REINICIAR PC				
	7	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		
	REINICIAR PC				
	9	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		

Tipo de imagen	Método de ampliación	Modalidad	Niveles de dificultad	Tiempos	
				PC A	PC B
REINICIAR PC					
Texto	5	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		
	REINICIAR PC				
	7	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		
	REINICIAR PC				
	9	1	Bajo		
			Medio		
			Alto		
		REINICIAR PC			
		2	Bajo		
			Medio		
			Alto		

7.3. Análisis de resultados

Los resultados obtenidos luego de las pruebas de ampliación realizadas serán evaluados con miras a obtener conclusiones con respecto a los siguientes aspectos, para cada método de ampliación:

- Tipos de imágenes para los que se recomienda y no se recomienda su uso.
- Características especiales relacionadas con el aspecto de las imágenes ampliadas.
- Rendimiento de los algoritmos de ampliación.

7.3.1. Prueba de apariencia

Luego de la aplicación de los diez métodos de ampliado desarrollados a las imágenes de prueba, se obtuvieron las imágenes que se muestran en el ANEXO A. Sobre la base de dichos resultados

pueden hacerse los comentarios siguientes para los diversos métodos, agrupados por el tipo de interpolación empleado.

Método 0. Ampliado simple.

Este método constituye una buena opción cuando se trata de imágenes de alto contraste, generalmente formadas por líneas horizontales y verticales o figuras simples, tales como imágenes de texto y algunos dibujos. No se recomienda para fotografías o imágenes con una diversidad de colores o detalles alta, debido a que se pone en evidencia la distorsión ocasionada por el ruido de cuantificación.

Métodos 1 al 4. Algoritmo del asterisco.

El algoritmo del asterisco genera imágenes que no ponen de manifiesto el ruido de cuantificación, lo que permite obtener una mejor calidad en las mismas. Sin embargo, los métodos 1 y 2 producen ciertas imperfecciones en forma de pequeños asteriscos en la cercanía de regiones de alto contraste, por lo que no se recomiendan para el ampliado de texto. Los métodos 3 y 4 generan resultados satisfactorios para los tres tipos de imágenes. El principal inconveniente de los métodos de ampliación con el algoritmo del asterisco lo constituye el marco negro producido en la periferia de las imágenes ampliadas.

Método 5. Interpolación lineal.

De los métodos que utilizan interpolación lineal (métodos del 1 al 5), éste es el que proporciona los mejores resultados en cuanto a la apariencia de la imagen, para todas las categorías de imágenes de prueba.

Métodos 6 y 9. Interpolación polinomial.

El método 6, el cual utiliza interpolación polinomial tradicional, produce imágenes ampliadas afectadas por una serie de manchas que

no forman parte de las imágenes originales, lo que implica un deterioro significativo en la calidad visual de las mismas. Este efecto se aprecia más en imágenes que presentan cambios no graduales en los colores en regiones adyacentes, como en las imágenes de prueba tipo texto y tipo dibujo. Por otro lado, el método 9, basado en interpolación polinomial adaptativa, genera resultados visualmente satisfactorios para todas las imágenes de prueba utilizadas, por lo que constituye el mejor método basado en este tipo de interpolación.

Métodos 7 y 8. Interpolación segmentaria.

El método que utiliza c-splines es el que produce las imágenes con mayor definición de todos los métodos de ampliado, y para todas las categorías de imágenes de prueba; sin embargo, genera una especie de franja semitransparente a lo largo de la interface entre dos regiones de alto contraste. Por otro lado, la interpolación con b-splines produce imágenes sin ningún tipo de imperfecciones, excepto porque la imagen parece estar desenfocada o borrosa. Este método se recomienda cuando las imágenes poseen un nivel de ruido alto, puesto que en estas circunstancias, el desenfoque se traduce en más inmunidad al ruido, generando imágenes más definidas que con cualquier otro método (tal como sucede con la imagen de tipo texto).

En la Tabla 7.5 se resumen las características mencionadas arriba, mostrando los tipos de imagen para los cuales es recomendable utilizar cada uno de los métodos de ampliación, así como las imperfecciones producidas en las imágenes ampliadas. Para un tipo de imagen dado, un signo de verificación (✔) indica que el método de ampliación en cuestión se puede utilizar satisfactoriamente. Una "X" (✘) indica que no es recomendable utilizar dicho método para ese tipo de imagen. Un par de signos de interrogación (¿?) indican que la idoneidad del método de ampliación depende de las características de la imagen a ampliar, es

decir, que el método no se comporta de la misma forma todas las veces, para el mismo tipo de imagen.

Tabla 7.5. Resumen de las características de los métodos relacionadas con la apariencia de las imágenes ampliadas.

Método	Fotografía	Dibujo	Texto	Imperfecciones
0	✗	¿?	✓	Ruido de cuantificación
1, 2	✓	¿?	✗	Asteriscos, marco negro
3, 4	✓	✓	✓	Marco negro
5	✓	✓	✓	—
6	¿?	✗	✗	Manchas
7	✓	✓	✓	Franja semitransparente
8	✓	✓	✓	Difusión
9	✓	✓	✓	—

7.3.2. Pruebas de tiempo de procesamiento

En la Tabla 7.6 se muestran los datos recopilados luego de las pruebas de tiempo de procesamiento en ambas computadoras (PC A y PC B según la Tabla 7.3). Las imágenes obtenidas con dichas pruebas no se incluyen en este documento, debido a que son numerosas y de gran tamaño; sin embargo, se encuentran disponibles en el disco compacto que acompaña a este documento.

Tabla 7.6. Datos recopilados en las pruebas de tiempo de procesamiento (en segundos)

Tipo de imagen	Método de ampliación	Modalidad	Niveles de dificultad	Tiempos (segundos)	
				PC A	PC B
Fotografía	5	1	Bajo	1.78	1.32
			Medio	5.27	5.41
			Alto	22.56	31.88
					REINICIAR PC
		2	Bajo	1.78	1.32
			Medio	6.59	7.16
			Alto	28.45	45.43
					REINICIAR PC
	7	1	Bajo	2.09	5.63
			Medio	5.97	30.45
			Alto	25.76	223.00
					REINICIAR PC

Tipo de imagen	Método de ampliación	Modalidad	Niveles de dificultad	Tiempos (segundos)	
				PC A	PC B
		2	Bajo	2.09	5.63
			Medio	7.34	39.75
			Alto	40.50	326.70
					REINICIAR PC
	9	1	Bajo	24.53	25.13
			Medio	69.83	76.50
			Alto	281.75	307.70
					REINICIAR PC
		2	Bajo	24.53	25.13
			Medio	317.84	384.28
			Alto	3091.50	3255.50
					REINICIAR PC
Dibujo	5	1	Bajo	1.87	1.71
			Medio	5.09	5.58
			Alto	22.04	31.32
					REINICIAR PC
		2	Bajo	1.87	1.71
			Medio	6.28	7.06
			Alto	28.25	44.88
					REINICIAR PC
	7	1	Bajo	2.05	5.79
			Medio	5.91	30.37
			Alto	25.63	222.84
					REINICIAR PC
		2	Bajo	2.05	5.79
			Medio	7.36	40.33
			Alto	39.95	325.90
					REINICIAR PC
	9	1	Bajo	19.63	20.21
			Medio	60.47	65.40
			Alto	253.72	277.09
					REINICIAR PC
		2	Bajo	19.63	20.21
			Medio	235.37	239.14
			Alto	2749.80	2840.80
					REINICIAR PC
Texto	5	1	Bajo	1.81	2.00
			Medio	5.09	5.73
			Alto	22.04	31.58
					REINICIAR PC
		2	Bajo	1.81	2.00
			Medio	6.16	7.20
			Alto	27.97	44.81
					REINICIAR PC
	7	1	Bajo	2.03	5.80
			Medio	5.79	30.34
			Alto	25.74	222.55
					REINICIAR PC
		2	Bajo	2.03	5.80
			Medio	7.49	40.25

Tipo de imagen	Método de ampliación	Modalidad	Niveles de dificultad	Tiempos (segundos)	
				PC A	PC B
			Alto	41.54	326.33
	REINICIAR PC				
	9	1	Bajo	21.60	20.43
			Medio	60.46	56.81
			Alto	231.73	221.05
		REINICIAR PC			
		2	Bajo	21.60	20.43
			Medio	251.36	270.90
			Alto	3057.20	3339.20

Los datos recopilados en la Tabla 7.6 pueden ser analizados de diversas formas, dependiendo de qué tipo de comparaciones se desee realizar. Los aspectos sobre los cuales se ha decidido indagar aquí son los siguientes:

- Tendencia de los algoritmos
- Eficiencia de los algoritmos
- Influencia del hardware sobre la eficiencia de los algoritmos

Asimismo, se examinará la influencia de los siguientes factores para cada uno de los aspectos anteriores:

- Tipo de imagen ampliada
- Modalidad de prueba
- Niveles de dificultad

La información recopilada para cada aspecto de evaluación se presentará utilizando gráficos de curvas o de barras para facilitar su análisis.

Tendencia de los algoritmos

Con este análisis se pretende determinar cuál es el comportamiento de cada algoritmo de ampliación a medida que se incrementa el número de datos que deben ser procesados. Dicho incremento constituye el *nivel de dificultad* mencionado en páginas anteriores. Como ya se ha

mencionado, el nivel de dificultad se ha variado en dos modalidades; la primera de ellas consiste en aumentar la cantidad de ampliación, pero conservando la imagen de entrada del mismo tamaño; mientras que con la segunda modalidad se amplían las imágenes previamente ampliadas, en una cantidad constante de veces, de manera que la imagen de entrada sea cada vez mayor.

Las gráficas que se presentan se han agrupado por tipo de imagen y por modalidad de prueba. En cada gráfica se muestran las curvas de tendencia correspondientes a cada método de ampliación.

Una buena curva de tendencia consistiría de una línea recta, significando que el algoritmo consume linealmente más tiempo a medida que se incrementan los datos. Una curva aún mejor, sería una de tipo logarítmico, puesto que la razón de aumento de tiempo con respecto al nivel de dificultad sería cada vez más pequeña. Por otro lado, una mala curva de tendencia podría ser una de tipo exponencial, porque implicaría que el tiempo de procesamiento crecería a una velocidad cada vez mayor.

En las gráficas que siguen se muestran las curvas de tendencia requeridas. Para efectos de brevedad, en las gráficas sólo se considera a la computadora B (la que cuenta con menos recursos de hardware, según la Tabla 7.3). El eje de abscisas es el factor de aumento de píxeles en la imagen de salida, con respecto a la imagen de entrada, es decir, n^2 (siendo n la cantidad de ampliado).

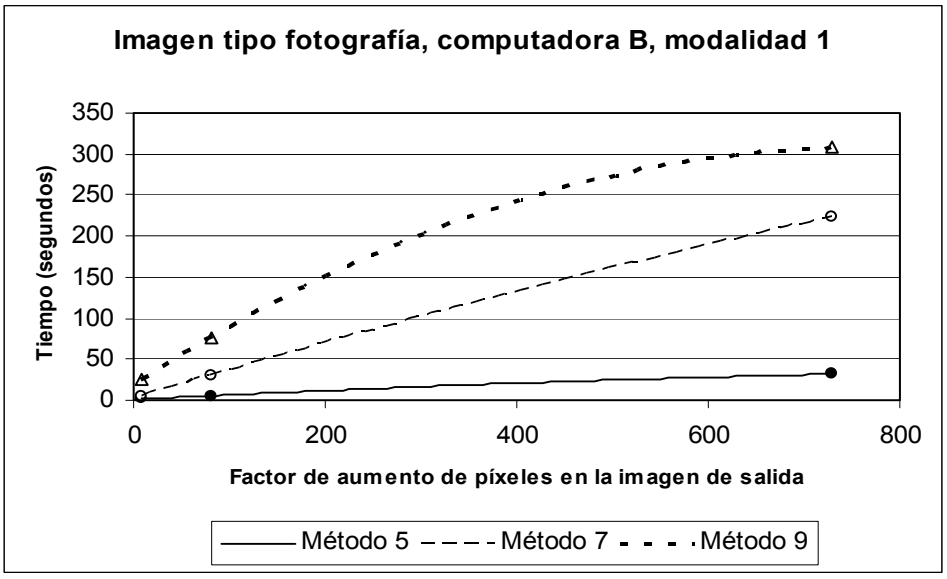


Imagen tipo fotografía, computadora B, modalidad 1
Tiempo (segundos)
350
300
250
200
150
100
50
0
0
200
400
600
800
Factor de aumento de píxeles en la imagen de salida
Método 5
Método 7
Método 9

(a)

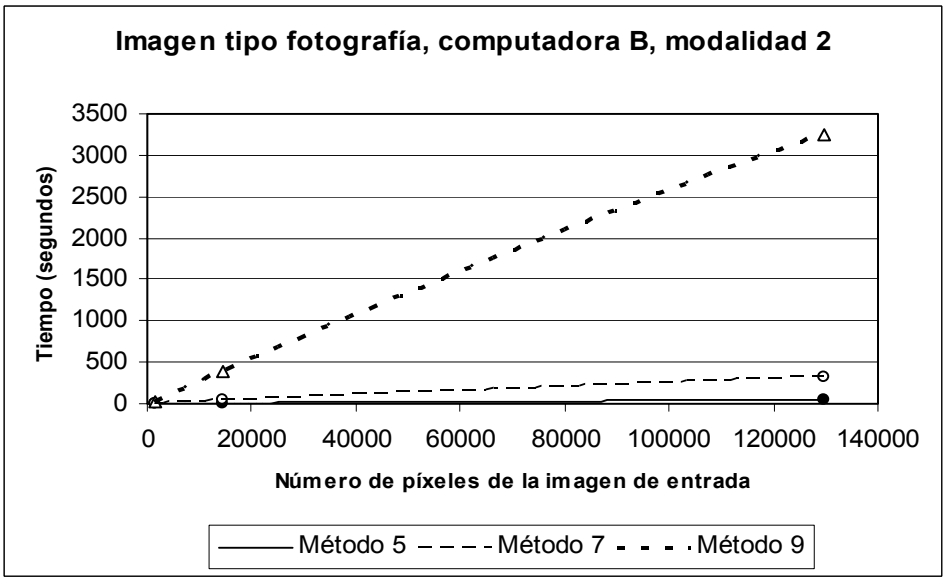


Imagen tipo fotografía, computadora B, modalidad 2
Tiempo (segundos)
3500
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000
80000
100000
120000
140000
Número de píxeles de la imagen de entrada
Método 5
Método 7
Método 9

(b)

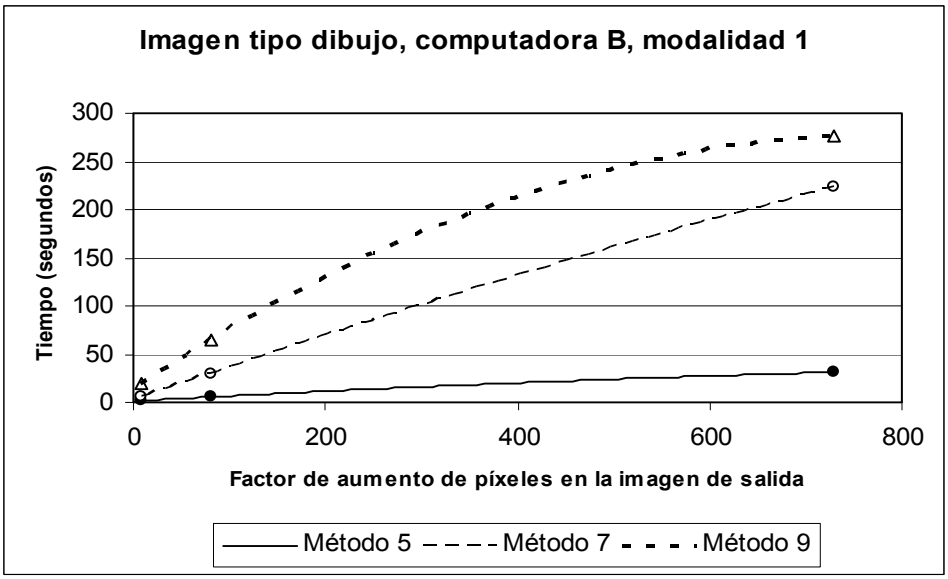


Imagen tipo dibujo, computadora B, modalidad 1
Tiempo (segundos)
300
250
200
150
100
50
0
0
200
400
600
800
Factor de aumento de píxeles en la imagen de salida
Método 5
Método 7
Método 9

(c)

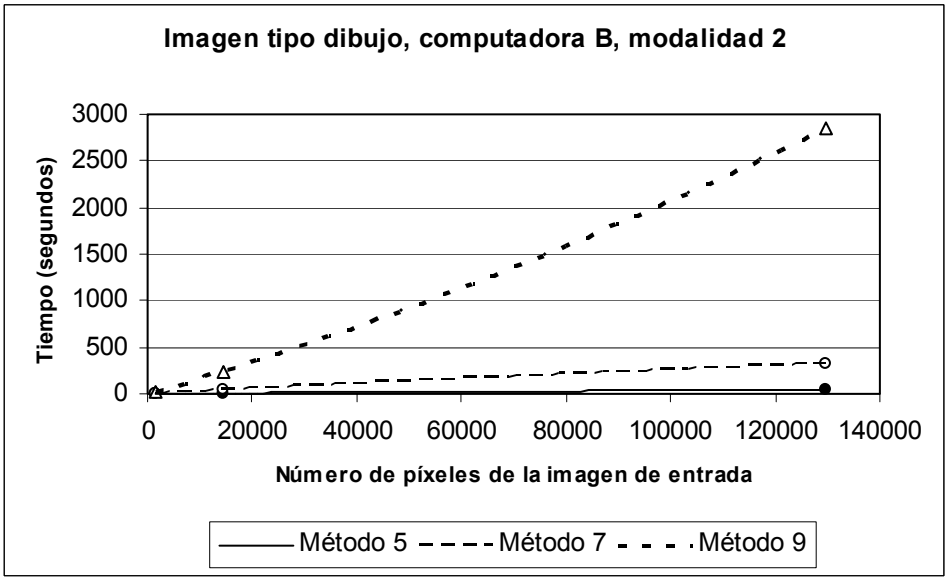
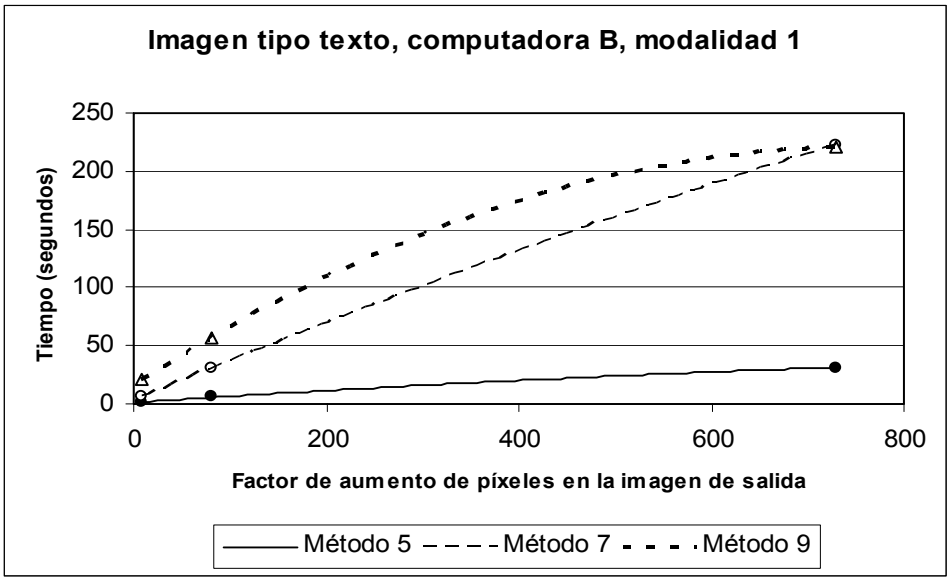
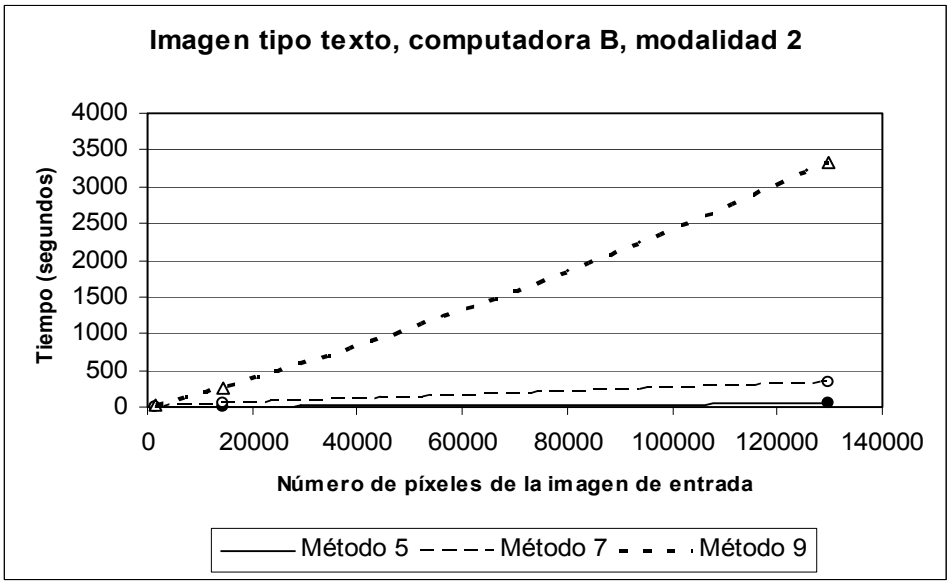


Imagen tipo dibujo, computadora B, modalidad 2
Tiempo (segundos)
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000
80000
100000
120000
140000
Número de píxeles de la imagen de entrada
Método 5
Método 7
Método 9

(d)



(e)



(f)

Figura 7.1. Tendencias de los métodos de ampliación.

A partir de las gráficas anteriores se puede comentar lo siguiente:

- Los tres métodos de ampliación evaluados presentan curvas de tendencia aceptables para los tres tipos de imágenes ampliadas y para ambas modalidades de prueba.

- El método que consume mayor tiempo para realizar la ampliación es el que se basa en interpolación polinomial adaptativa, seguido por c-splines; siendo el método más rápido el de interpolación lineal.
- El aumento en la cantidad de píxeles de la imagen a ser ampliada (modalidad 2) constituye una mayor carga de procesamiento para todos los algoritmos, que el aumento de la cantidad de ampliación (modalidad 1).
- El método 9 (interpolación polinomial adaptativa) sufre una disminución considerable del tiempo de procesamiento para imágenes de tipo texto, en la modalidad 1.

Eficiencia de los algoritmos

La eficiencia del algoritmo se analizará por medio de gráficos de barras, mostrando el tiempo de procesamiento por método, para cada tipo de imagen, y por cada computadora. Para este análisis es recomendable calcular un único valor de tiempo de procesamiento por cada método, involucrando de alguna manera los diversos niveles de dificultad evaluados, y para una modalidad de prueba dada. Dicho valor único para el tiempo de procesamiento será la suma de los tiempos requeridos para cada nivel de dificultad.

En la figura siguiente se muestran las gráficas de tiempo acumulado de procesamiento para los tres métodos estudiados, para cada tipo de imagen, para ambas modalidades de prueba, y para las dos computadoras. Nótese que el método más eficiente, según estas gráficas, sería aquél que consume el menor tiempo.

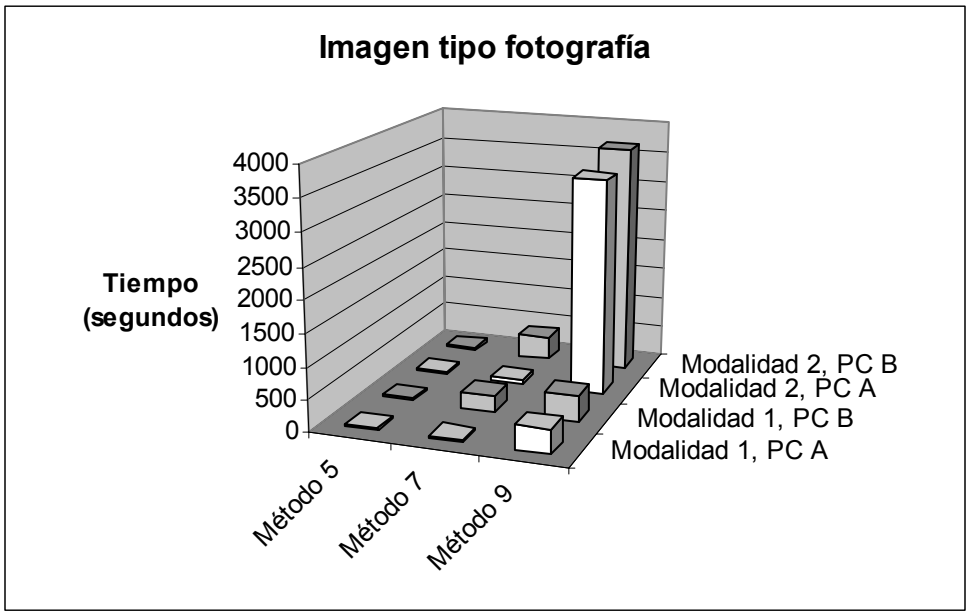


Imagen tipo fotografía
Tiempo (segundos)
4000
3500
3000
2500
2000
1500
1000
500
0
Método 5
Método 7
Método 9
Modalidad 2, PC B
Modalidad 2, PC A
Modalidad 1, PC B
Modalidad 1, PC A

(a)

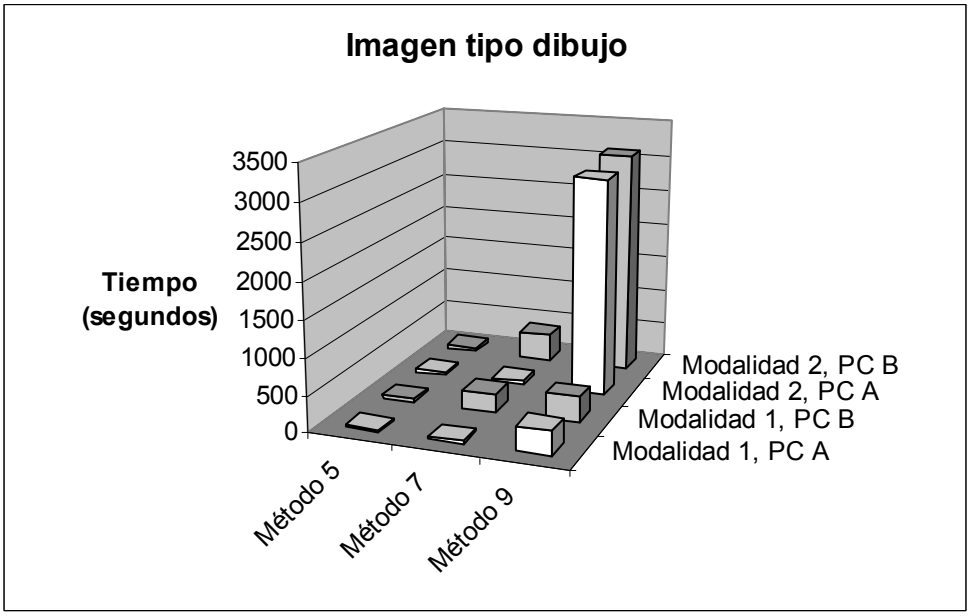
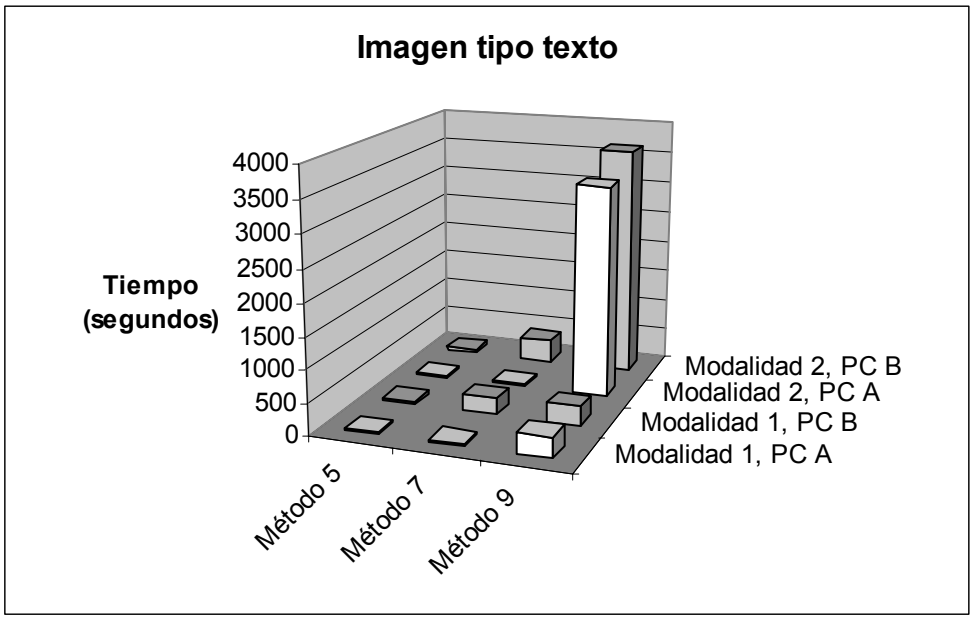


Imagen tipo dibujo
Tiempo (segundos)
3500
3000
2500
2000
1500
1000
500
0
Método 5
Método 7
Método 9
Modalidad 2, PC B
Modalidad 2, PC A
Modalidad 1, PC B
Modalidad 1, PC A

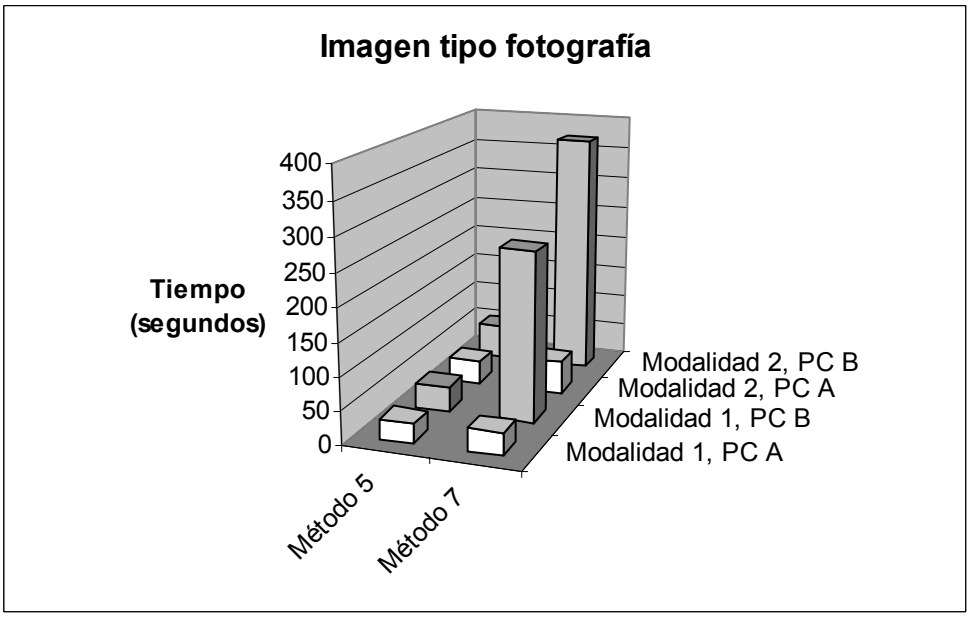
(b)



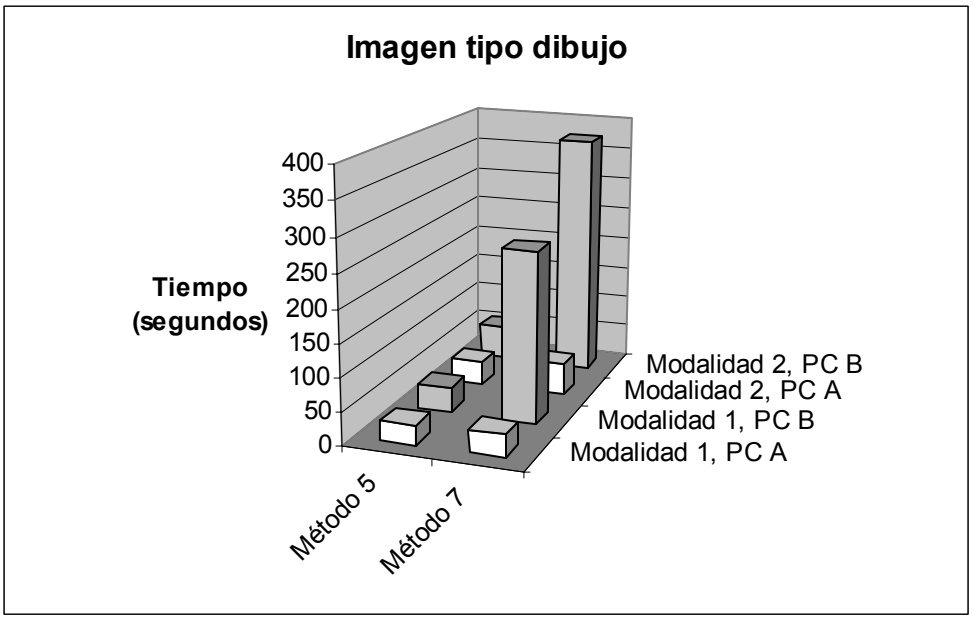
(c)

Figura 7.2. Tiempo de procesamiento acumulado de los métodos de evaluados.

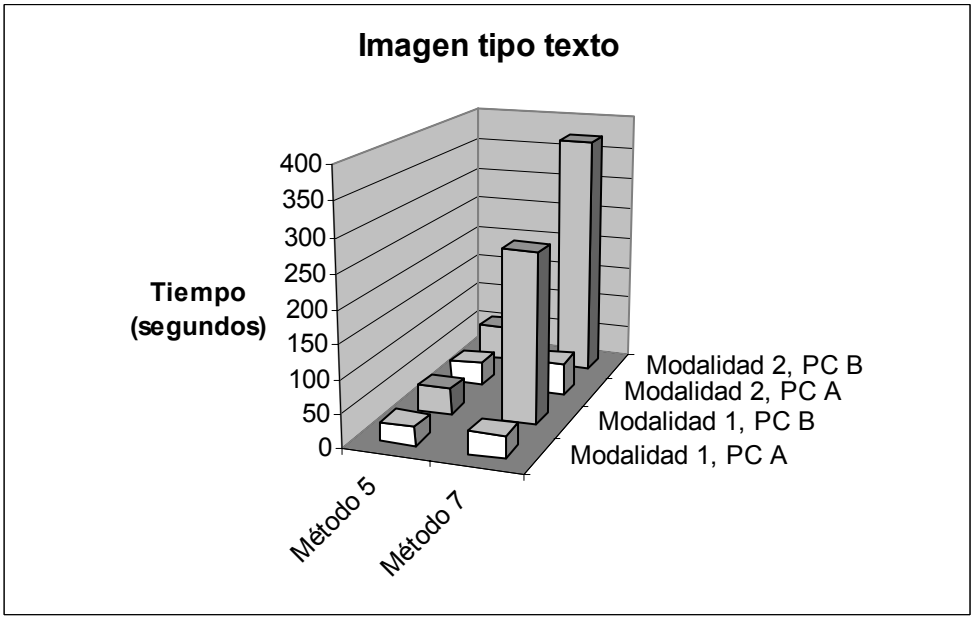
Para permitir la apreciación de las diferencias entre los métodos 5 y 7, en las gráficas que siguen se ha eliminado el método 9.



(a)



(b)



(c)

Figura 7.3. Tiempo de procesamiento acumulado de los métodos de ampliación 5 y 7.

De las figuras anteriores se desprende que el método más eficiente es el número 5 (interpolación lineal), seguido del número 7 (c-splines), mientras que el menos eficiente es el número 9 (interpolación polinomial adaptativa); todo ello para los tres tipos de imágenes, en

ambas computadoras, y con las dos modalidades de prueba. Es interesante notar que el método 7 es el único que presenta un cambio significativo en el tiempo de procesamiento con el aumento de los recursos de la computadora.

Influencia del hardware

Se analizará ahora la influencia del hardware sobre la eficiencia de los algoritmos, vista desde dos perspectivas, para cada método de ampliación.

- *Beneficio del aumento de recursos de hardware*. Este análisis consiste en la estimación de la cantidad de mejora de los tiempos de procesamiento para cada método de ampliado, al realizar un aumento en los recursos de hardware. Dicha mejora está relacionada con la dispersión existente en los datos. Por ejemplo, una mejora alta significaría que los tiempos en ambas computadoras difieren notablemente[24], mientras que una mejora nula implicaría que dichos tiempos son iguales. Por lo tanto, una medida de dicha mejora podría ser la desviación estándar de los tiempos de procesamiento en ambas computadoras. Para facilitar la comparación de los métodos, es conveniente obtener una única medición de mejoría por cada método evaluado. Esta medición única por método puede obtenerse promediando la mejoría de las dos modalidades de prueba y de los tres tipos de imágenes, empleando como resultado por modalidad a la suma de la mejora para cada nivel de dificultad. Sin embargo, para que los cálculos de mejoría de cada prueba puedan ser procesados entre sí, es necesario que sean independientes de la prueba en particular efectuada. Esta independencia puede lograrse dividiendo la desviación estándar de los tiempos entre la media aritmética de los mismos. El resultado así obtenido se conoce como *coeficiente de variación* (CV). En las tablas que siguen se muestran los datos calculados del

[24] Por supuesto, es de esperar que la computadora con mayores recursos presente tiempos menores. Si bien esto no se cumple para algunas pruebas (véase la Tabla 7.6), siempre es válido al incrementar el nivel de dificultad.

coeficiente de variación para los tres métodos de ampliación evaluados, agrupados por tipo de imagen de prueba. La columna titulada "CV(PCA, PCB)" contiene el coeficiente de variación de los tiempos de procesamiento de las computadoras A y B. Para cada modalidad se ha obtenido el coeficiente de variación acumulado[25], es decir, la suma del valor de CV para los tres niveles de dificultad. Finalmente, en la última columna se muestra el promedio por modalidad del coeficiente de variación acumulado, para cada método de ampliación.

Tabla 7.7. Mejora en los tiempos de procesamiento por el aumento de recursos para la imagen tipo fotografía

Método de ampliación	Modalidad	Niveles de dificultad	CV(PCA,PCB)	Promedio por modalidad
5	1	Bajo	0.21	0.53
		Medio	0.02	
		Alto	0.24	
	TOTAL		0.47	
	2	Bajo	0.21	
		Medio	0.06	
		Alto	0.33	
TOTAL			0.59	
7	1	Bajo	0.65	2.72
		Medio	0.95	
		Alto	1.12	
	TOTAL		2.72	
	2	Bajo	0.65	
		Medio	0.97	
		Alto	1.10	
TOTAL			2.72	
9	1	Bajo	0.02	0.17
		Medio	0.06	
		Alto	0.06	
	TOTAL		0.14	
	2	Bajo	0.02	
		Medio	0.13	
		Alto	0.04	
TOTAL			0.19	

[25] Este es un término empleado aquí por facilidad, y no alude a ninguna definición estadística formal.

Tabla 7.8. Mejora en los tiempos de procesamiento por el aumento de recursos para la imagen tipo dibujo

Método de ampliación	Modalidad	Niveles de dificultad	CV(PCA,PCB)	Promedio por modalidad
5	1	Bajo	0.07	0.46
		Medio	0.08	
		Alto	0.25	
	TOTAL		0.41	
	2	Bajo	0.07	
		Medio	0.11	
		Alto	0.33	
TOTAL			0.51	
7	1	Bajo	0.68	2.75
		Medio	0.96	
		Alto	1.12	
	TOTAL		2.76	
	2	Bajo	0.68	
		Medio	0.97	
		Alto	1.09	
TOTAL			2.75	
9	1	Bajo	0.04	0.14
		Medio	0.04	
		Alto	0.03	
	TOTAL		0.12	
	2	Bajo	0.04	
		Medio	0.05	
		Alto	0.06	
TOTAL			0.15	

Tabla 7.9. Mejora en los tiempos de procesamiento por el aumento de recursos para la imagen tipo texto

Método de ampliación	Modalidad	Niveles de dificultad	CV(PCA,PCB)	Promedio por modalidad
5	1	Bajo	0.06	0.42
		Medio	0.06	
		Alto	0.25	
	TOTAL		0.37	
	2	Bajo	0.06	
		Medio	0.08	
		Alto	0.32	
TOTAL			0.47	
7	1	Bajo	0.67	2.75
		Medio	0.95	
		Alto	1.12	
	TOTAL		2.75	

	2	Bajo	0.67	
		Medio	0.98	
		Alto	1.11	
		TOTAL	2.76	
9	1	Bajo	0.02	0.10
		Medio	0.06	
		Alto	0.06	
		TOTAL	0.14	
	2	Bajo	0.02	
		Medio	0.01	
		Alto	0.02	
		TOTAL	0.05	

Finalmente, deben promediarse los resultados obtenidos para cada imagen, de forma de obtener un único resultado por método de ampliación. Dichos promedios se resumen en la Tabla 7.10.

Tabla 7.10. Unidades estimadas de mejoría para cada método de ampliación

Método de ampliación	Unidades estimadas de mejoría
5	0.47
7	2.74
9	0.13

A partir de los datos de la Tabla 7.10 se construye la gráfica que se muestra en la Figura 7.4.

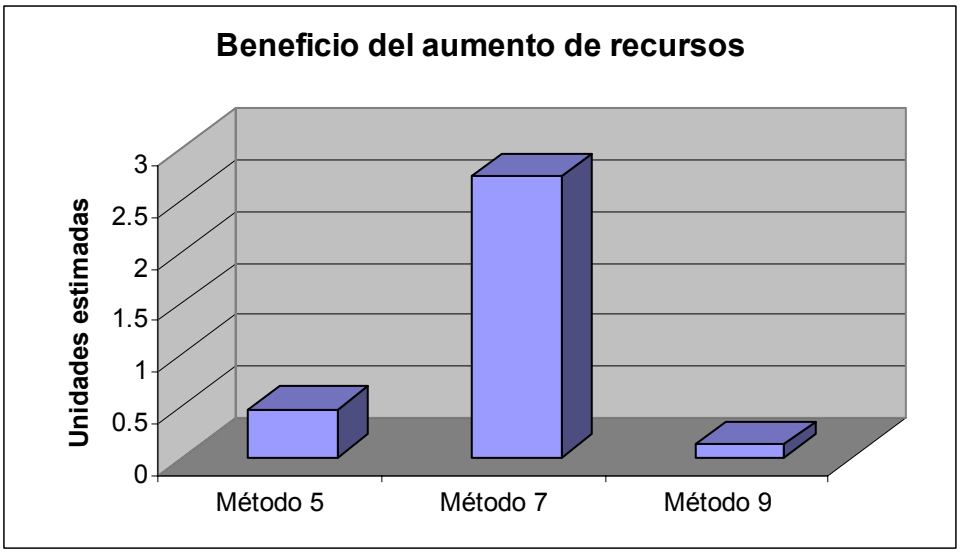


Figura 7.4. Beneficio del aumento de recursos sobre los tiempos de procesamiento para cada método de ampliado.

A partir de la gráfica anterior puede observarse que la eficiencia del método 9 es la menos sensible al aumento de los recursos de hardware, mientras que el método 7 mejora significativamente su desempeño ante tales circunstancias.

- *Influencia del aumento del nivel de dificultad en el beneficio del aumento de recursos de hardware.* Adicionalmente al análisis de la mejora de los tiempos de procesamiento provocado por el aumento de los recursos de hardware, también es interesante investigar cómo se ve influenciado dicho beneficio con el aumento del nivel de dificultad en las pruebas. En otras palabras, una vez establecido que efectivamente resulta beneficioso el aumento de los recursos de hardware, y luego de haber determinado cuál método de ampliación resulta más beneficiado con dicho aumento, se trata ahora de indagar acerca de si este beneficio crece al aumentar los niveles de dificultad, para luego determinar en cuál método de ampliado es mayor este crecimiento. Para llevar a cabo este análisis, se hace uso de la cantidad "PCA / PCB", es decir, la razón del tiempo de procesamiento en la computadora A al tiempo respectivo en la computadora B. Esta cantidad, que podría denotarse por *razón de tiempos*, es un indicador del cambio en el tiempo de procesamiento para una prueba individual dada. Mientras más pequeña sea la razón de tiempos, mayor será la "ganancia" en el tiempo de procesamiento al

aumentar los recursos de hardware. Como se podrá ver más adelante, esta razón disminuye conforme se aumenta el nivel de dificultad, indicando que para imágenes de entrada o cantidad de ampliado cada vez mayores, la mejora en los tiempos debidas a la mejora de recursos es mayor. La tarea consiste por tanto en estimar una cantidad que represente al incremento de dicho beneficio, para luego comparar los diversos métodos en base a estas cantidades. Nuevamente, el coeficiente de variación puede ser utilizado para este propósito. El procedimiento a seguir es el siguiente.

- Calcular la razón de tiempos para cada nivel de dificultad, para todas las modalidades de prueba, para todos los métodos de ampliado, y para todos los tipos de imágenes evaluadas.
- Calcular el coeficiente de variación de los tres niveles de dificultad por cada modalidad de prueba.
- Promediar el coeficiente de variación de las dos modalidades de prueba por cada método de ampliación, por cada tipo de imagen.
- Promediar el resultado obtenido por cada método de ampliación para las tres imágenes de prueba.

En las tablas que siguen se muestran los datos obtenidos en los pasos recién mencionados.

Tabla 7.11. Promedio por modalidad para la imagen de tipo fotografía

Método de ampliación	Modalidad	Niveles de dificultad	PCA / PCB	CV(PCA / PCB)	Promedio por modalidad
5	1	Bajo	1.35	0.32	0.35
		Medio	0.97		
		Alto	0.71		
	2	Bajo	1.35	0.38	
		Medio	0.92		
		Alto	0.63		
7	1	Bajo	0.37	0.57	0.57
		Medio	0.20		
		Alto	0.12		
	2	Bajo	0.37	0.57	
		Medio	0.18		

		Alto	0.12		
9	1	Bajo	0.98	0.04	0.06
		Medio	0.91		
		Alto	0.92		
	2	Bajo	0.98	0.09	
		Medio	0.83		
		Alto	0.95		

Tabla 7.12. Promedio por modalidad para la imagen de tipo dibujo

Método de ampliación	Modalidad	Niveles de dificultad	PCA / PCB	CV(PCA / PCB)	Promedio por modalidad
5	1	Bajo	1.09	0.22	0.24
		Medio	0.91		
		Alto	0.70		
	2	Bajo	1.09	0.27	
		Medio	0.89		
		Alto	0.63		
7	1	Bajo	0.35	0.55	0.55
		Medio	0.19		
		Alto	0.12		
	2	Bajo	0.35	0.55	
		Medio	0.18		
		Alto	0.12		
9	1	Bajo	0.97	0.03	0.02
		Medio	0.92		
		Alto	0.92		
	2	Bajo	0.97	0.01	
		Medio	0.98		
		Alto	0.97		

Tabla 7.13. Promedio por modalidad para la imagen de tipo texto

Método de ampliación	Modalidad	Niveles de dificultad	PCA / PCB	CV(PCA / PCB)	Promedio por modalidad
5	1	Bajo	0.91	0.14	0.16
		Medio	0.89		
		Alto	0.70		
	2	Bajo	0.91	0.19	
		Medio	0.86		
		Alto	0.62		
7	1	Bajo	0.35	0.55	0.53
		Medio	0.19		
		Alto	0.12		
	2	Bajo	0.35	0.52	

		Medio	0.19		
		Alto	0.13		
9	1	Bajo	1.06	0.01	0.04
		Medio	1.06		
		Alto	1.05		
	2	Bajo	1.06	0.08	
		Medio	0.93		
		Alto	0.92		

Finalmente, los promedios de los tres tipos de imágenes para cada método de ampliación se muestran en la Tabla 7.14.

Tabla 7.14. Unidades estimadas del aumento de mejoría para cada método de ampliación.

Método de ampliación	Unidades estimadas
5	0.25
7	0.55
9	0.04

Con los datos de la Tabla 7.14 se construye el gráfico mostrado en la Figura 7.5.

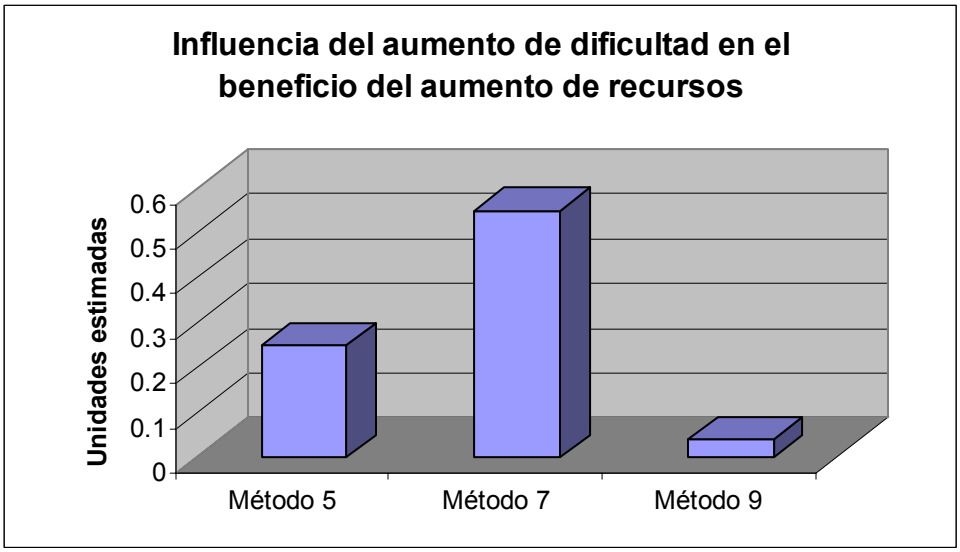


Figura 7.5. Cantidad de aumento del beneficio del aumento de recursos sobre los tiempos de procesamiento para cada método de ampliado, al aumentar el nivel de dificultad.

En la gráfica anterior se puede apreciar que el método 7 es el que aprovecha mejor el aumento de recursos a medida que se incrementa el nivel de dificultad. También puede observarse que para el método 9, el cual no se beneficia mucho del aumento de recursos, dicho beneficio es casi indiferente al incremento del nivel de dificultad.

7.4. Conclusiones

A través de las pruebas realizadas a los métodos de ampliación desarrollados, considerando la calidad de las imágenes resultantes, y el tiempo de procesamiento de los diversos algoritmos, se puede concluir lo siguiente:

1. El uso de los píxeles vecinos diagonales como información adicional a la de las verticales y horizontales para la estimación de píxeles desconocidos en una imagen ampliada no contribuye significativamente con la calidad de ésta, si bien puede ocasionar imperfecciones en la misma.
2. La estrategia de utilizar interpolación unidimensional sobre las filas y columnas de la imagen original, y a partir de ellas estimar los píxeles restantes promediando los resultados de éstas, ha demostrado ser una forma satisfactoria de realizar la ampliación de imágenes digitales.
3. Todos los tipos de interpolación utilizados cuentan con al menos un método de ampliación que proporciona, en general, resultados similares satisfactorios en cuanto a la calidad de la imagen ampliada.
4. Algunos métodos de ampliación generan mejores resultados para cierto tipo de imágenes que para otro, mientras que otros métodos se desenvuelven satisfactoriamente con todos los tipos de imágenes de prueba (véase la Tabla 7.5).
5. Existe una diferencia significativa con respecto al tiempo requerido por cada método para realizar el proceso de ampliado, siendo el más rápido el método 5 (interpolación lineal) seguido del método 7 (interpolación con c-splines); mientras que el método más lento es el número 9 (interpolación polinomial adaptativa).

8. ANEXOS

ANEXO A. IMÁGENES DE PRUEBA Y RESULTADOS

Las imágenes mostradas aquí constituyen los resultados obtenidos luego de la aplicación de la prueba de apariencia a los diez métodos de ampliación. Las imágenes obtenidas con la prueba de tiempos de procesamiento no se incluyen, debido a que son numerosas y de gran tamaño. Sin embargo, dichas imágenes se encuentran disponibles en el disco compacto que acompaña a este documento. Su única función fue la de proporcionar los diversos tiempos de procesamiento analizados, puesto que la evaluación de la calidad visual de las imágenes ampliadas puede juzgarse a partir de los resultados de la prueba de apariencia mostrados aquí.

Los métodos se aplicaron a tres imágenes digitales de mapas de bits en formato RGB, de 40 filas por 40 columnas, pertenecientes a una de tres categorías: fotografías, dibujos y texto escaneado. El factor de ampliación fue de seis, para todas las imágenes y para todos los métodos.

A.1. Categoría fotografía[26]

Imagen original

[26] La fotografía original contiene un marco negro en la periferia de la imagen, para facilitar la identificación de los efectos debidos al contraste alto. No debe confundirse dicho marco con el generado por los métodos basados en el algoritmo del asterisco.

Método 0

Método 1

Método 2

Método 3

Método 4

Método 5

Método 6

Método 7

Método 8

Método 9

A.2. Categoría dibujo

Imagen original

Método 0

Método 1

Método 2

Método 3

Método 4

Método 5

Método 6

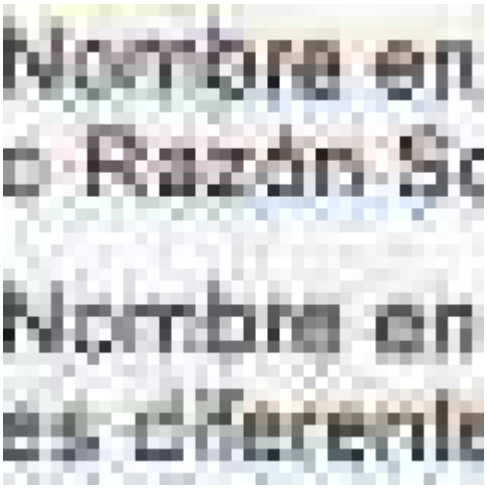
Método 7

Método 8

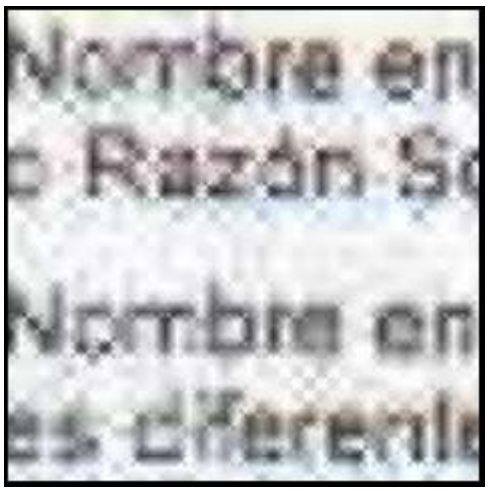
Método 9

A.3. Categoría texto

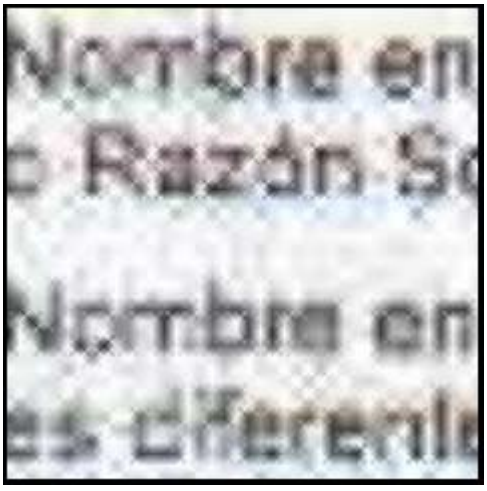
Imagen original



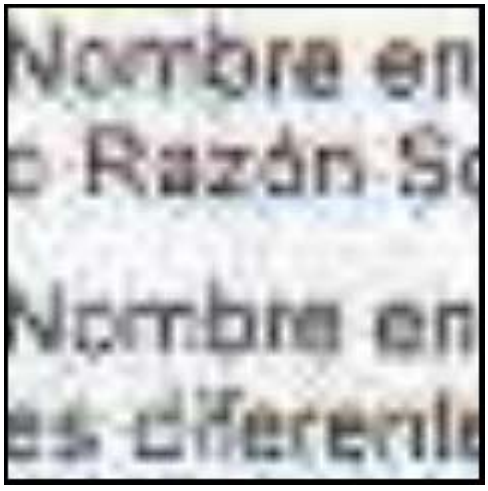
Método 0



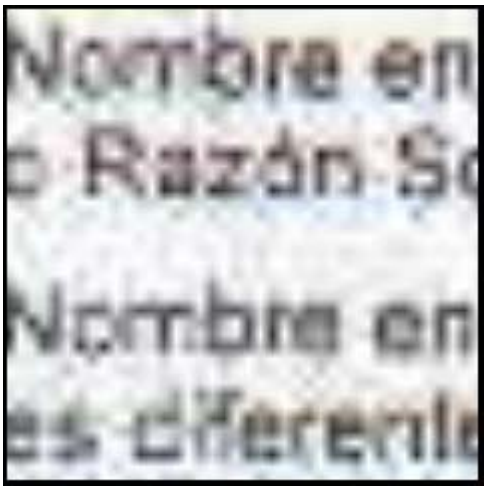
Método 1



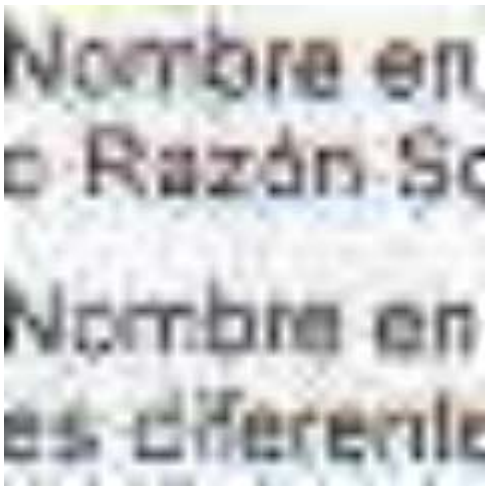
Método 2



Método 3



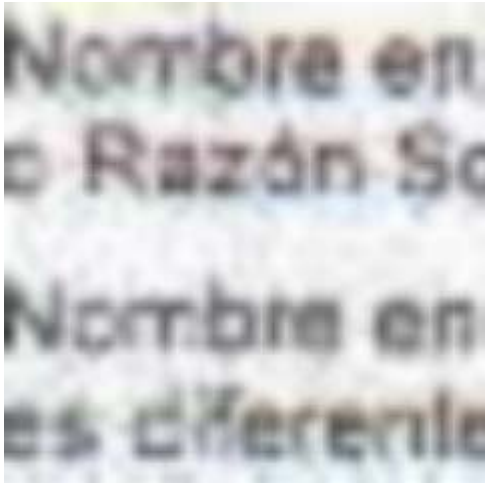
Método 4



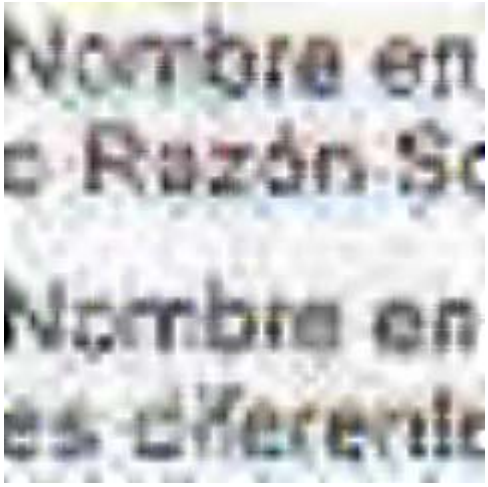
Método 5

Método 6

Método 7



Método 8



Método 9

ANEXO B. ARCHIVOS M EN MATLAB

En este apéndice se describen todas las funciones creadas en archivos M en MATLAB encargadas de realizar el procesamiento de las imágenes digitales por los diversos métodos de ampliación estudiados. Para cada una de ellas se expone su propósito, sintaxis, argumentos, el tipo de resultados que devuelve, las funciones que invoca y el código fuente en el lenguaje de programación de MATLAB. Los algoritmos han sido revisados con el objeto de incrementar su eficiencia. Adicionalmente, se tratado de seguir las recomendaciones expuestas en *Maximizing MATLAB Performance* en [10] u [11] en la elaboración de los archivos M.

NOTA: En muchas de las funciones que se describen a continuación aparecen los argumentos y variables `h`, `acumulado`, y `subtotal`, además de la función `waitbar` de MATLAB. Éstas se encargan únicamente del manejo de la barra de progreso que se muestra en pantalla mientras se está realizando el proceso de ampliado o reducción, y no intervienen en ninguna forma con los algoritmos y procedimientos matemáticos relacionados con la ampliación o reducción de imágenes. Por este motivo, la información relacionada con el uso de dichos elementos no se provee en este documento. A manera ilustrativa se menciona lo siguiente:

- `h` es una variable que contiene toda la información relacionada con el objeto "barra de progreso" de MATLAB. Dicha información constituye una estructura de datos conocida como "handle". Para mayor información, consúltese [10] u [11].
- `acumulado` es una variable que almacena la fracción de la barra de progreso que está rellena, antes de invocar a una función que continuará rellenando la barra.
- `subtotal` es una variable que contiene la fracción de la barra de progreso asignada a una función antes de ser invocada.

♦ `waitbar` es la función de MATLAB encargada de crear, presentar y modificar una barra de progreso. Para mayor información, consúltese [10] u [11].

A continuación se prosigue con la descripción de las funciones utilizadas por los diversos métodos de ampliación / reducción.

B.1. Funciones de uso general

Función `JK_zoom`

Propósito

Ampliar o reducir una región de una imagen digital.

Sintaxis

```
tiempo = jk_zoom (coord_x_inicial, coord_y_inicial,
coord_x_final, coord_y_final, n, tipo, url_origen, url_destino)
```

Argumentos

♦ `coord_x_inicial`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.

♦ `coord_y_inicial`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.

♦ `coord_x_final`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.

♦ `coord_y_final`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.

♦ `n`: Número entero positivo que indica la cantidad de ampliación o reducción.

♦ `tipo`: Número entero que indica el tipo de método a utilizar para la ampliación o reducción. Los tipos disponibles se resumen en la Tabla 8.1.

Tabla 8.1. Descripción de los tipos de método de ampliación / reducción

Tipo	Descripción
-2	Reducción con promedio
-1	Reducción con píxel central
0	Ampliado simple
1	Diagonales interpoladas con los vecinos diagonales, relleno en espiral
2	Diagonales interpoladas con los vecinos diagonales, relleno radial
3	Diagonales interpoladas con las verticales y horizontales, relleno en espiral
4	Diagonales interpoladas con las verticales y horizontales, relleno radial
5	Interpolación lineal, relleno en malla
6	Interpolación polinomial, relleno en malla
7	Interpolación con c-splines, relleno en malla
8	Interpolación con b-splines, relleno en malla
9	Interpolación polinomial adaptativa, relleno en malla

- `url_origen`: Cadena de caracteres que indica la ruta completa y nombre de archivo donde se encuentra la imagen de entrada.
- `url_destino`: Cadena de caracteres que indica la ruta completa y nombre de archivo a donde se alojará la imagen de salida.

Resultados

- `tiempo`: Cadena de caracteres que indica el número de segundos que demoró el proceso completo de ampliación o reducción. Si `tiempo = 0`, indica que el proceso no pudo concluirse debido a algún error, y el archivo de salida no se escribió.
- Imagen ampliada o reducida alojada en el archivo especificado por `url_destino`.

Comentarios

- El origen de las coordenadas se encuentra en la esquina superior izquierda de la imagen (ver Figura 8.1).
- El valor más pequeño para las coordenadas es 1.
- Si las coordenadas son cero (todas) se ampliará la imagen completa.

- Si n = 1, se devolverá la misma imagen de entrada, o la región seleccionada en tamaño original.
- Estos comentarios se aplican para todas las funciones que utilicen a `n` y a las coordenadas como argumentos de entrada.

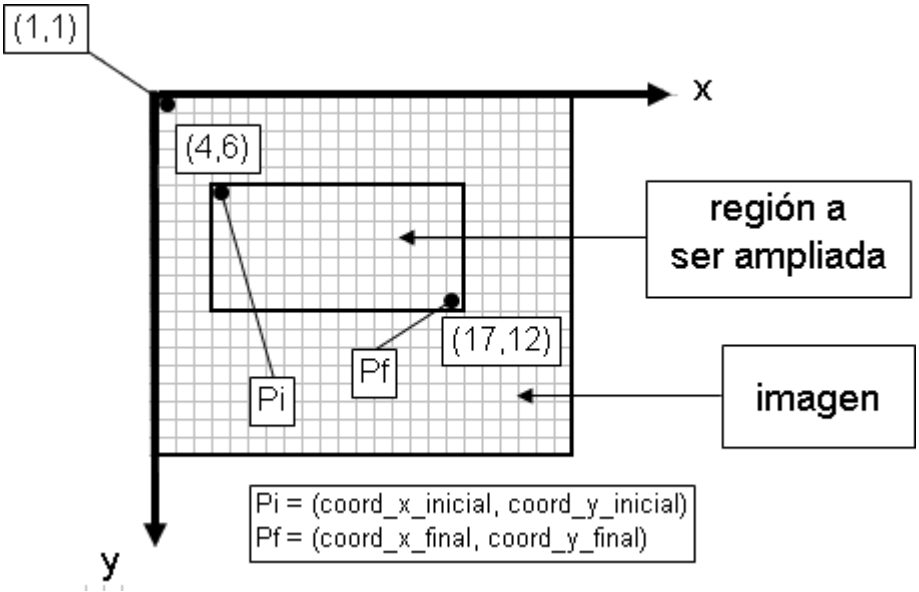


Figura 8.1. Definición de las coordenadas y de la región de interés.

Funciones empleadas

- `reducir_color`
- `ampliar_color`
- `asterisco_color`
- `polinomica_color`
- `jk_zoom_salir`

Código fuente

Listado 8.1. Función `JK_zoom`

```
function info=JK_zoom(coord_x_inicial,coord_y_inicial,coord_x_final,coord_y_final,...
   n,tipo,ruta_entrada, ruta_salida)
% JK_zoom hace una ampliación de una imagen ubicada en un archivo con extensión bmp o jpg,
% y la guarda en el archivo especificado. Sintaxis: tiempo = jk_zoom (coord_x_inicial,...
% coord_y_inicial, coord_x_final, coord_y_final, n, tipo, url_origen, url_destino)
% url_origen es una cadena que indica la ruta completa y nombre de archivo donde se
% encuentra la imagen de entrada.
% url_destino es una cadena que indica la ruta completa y nombre de archivo a donde se
% alojará la imagen de salida.
% La región de la imagen de entrada que se desea ampliar está especificada por el
% rectángulo cuya esquina superior izquierda tiene coordenadas
% (coord_x_inicial,coord_y_inicial) y cuya esquina inferior derecha tiene coordenadas
```

```
% (coord_y_final,coord_y_final).
% "n" es el número de veces que se desea ampliar la región especificada de la imagen de
% entrada. "tipo" indica el tipo de algoritmo utilizado para la ampliación.
% "info" es una cadena con el número de segundos que demoró el proceso completo. Si
% info=0, indica que el proceso fracasó.
% Si las coordenadas son cero (todas) se ampliará la imagen completa.
% Si n=1, se devolverá la misma imagen de entrada, o la región seleccionada en tamaño original.
% El argumento tipo puede tener los siguientes valores:
% tipo=-2 y -1: Se realiza una reducción de la imagen.
% tipo=0: Se realiza el ampliado simple.
% tipo=1 hasta tipo=4: Se amplía con el algoritmo del asterisco.
% tipo=5: Se amplía con interpolación lineal.
% tipo=6 y 9: Se amplía con interpolación polinomial.
% tipo 7 y 8: Se amplía con interpolación segmentaria (splines).
% Los diversos tipos se detallan a continuación:
% tipo = -2: Reducción con promedio de los píxeles vecinos.
% tipo = -1: Reducción con píxel central.
% tipo =  0: Ampliado simple.
% tipo =  1: Diagonales interpoladas con los vecinos diagonales. Relleno en espiral.
% tipo =  2: Diagonales interpoladas con los vecinos diagonales. Relleno radial.
% tipo =  3: Diagonales interpoladas con las verticales y horizontales. Relleno en espiral.
% tipo =  4: Diagonales interpoladas con las verticales y horizontales. Relleno radial.
% tipo =  5: Interpolación lineal. Relleno en malla.
% tipo =  6: Interpolación polinomial. Relleno en malla.
% tipo =  7: Interpolación con c-splines. Relleno en malla.
% tipo =  8: Interpolación con b-splines. Relleno en malla.
% tipo =  9: Interpolación polinomial adaptativa. Relleno en malla.

info='0';   % El tiempo reportado es 0 si el proceso fracasa.

% La cantidad de ampliado debe ser mayor que cero.
if n<1
   beep
   errordlg('Error: La cantidad de ampliado debe ser positiva y no nula.','Error')
   return
end
% La ruta de entrada debe ser una cadena.
if not(ischar(ruta_entrada))
   beep
   errordlg(sprintf('Error: La ruta de entrada: "%s" no es válida.',num2str(ruta_entrada)),'Error')
   return
end
% La ruta de salida debe ser una cadena.
if not(ischar(ruta_salida))
   beep
   errordlg(sprintf('Error: La ruta de salida: "%s" no es válida.',num2str(ruta_salida)),'Error')
   return
end
global salir tiempo_muerto
salir=0;
% Si salir = 1, el programa termina satisfactoriamente. Si no, es porque ha ocurrido un error.
```

```
tiempo=0;   % Si tiempo = 0, el proceso de ampliado fracasó.
tiempo_muerto=0;
% El tiempo muerto es el que transcurre mientras se encuentra desplegado el mensaje de
% confirmación de cancelación del proceso de ampliación / reducción.

% Se requieren 8 argumentos de entrada.
if nargin~=8
   beep
   errordlg('Error: El número de argumentos no es válido.','Error')
   return
end
slash_positions_out=find(ruta_salida=='\');
% Encuentra las posiciones en ruta_salida donde se encuentran los caracteres "\".

dir_salida=ruta_salida(1:slash_positions_out(end)); % Obtiene el directorio (s) de salida.
% El directorio de salida debe existir.
if exist(dir_salida)==0
   beep
   errordlg(sprintf('Error: La ruta de salida "%s" no existe.',dir_salida),'Error')
   return
end
extension=ruta_salida(end-2:end);   % Obtiene la extensión del archivo de salida.
% La extensión del archivo de salida debe ser "bmp" o "jpg".
if extension~='bmp' & extension~='jpg'
   beep
   errordlg('Error: La extensión del archivo de la imagen de salida no es válida.','Error')
   return
end
slash_positions=find(ruta_entrada=='\');
% Encuentra las posiciones en ruta_entrada donde se encuentran los caracteres "\".

try
   x=imread(ruta_entrada); % Carga la imagen de entrada.
catch
   % Reporta si la imagen de entrada no se pudo cargar con éxito.
   beep
   errordlg(sprintf('Error: No se encontró el archivo "%s" o su formato es incompatible.',...
      ruta_entrada),'Error')
   return
end
x=double(x);    % Convierte la imagen de entrada a números de punto flotante con doble presición.
% Convertir los argumentos a enteros y de doble presición.
arg(1)=fix(double(coord_x_inicial));
arg(2)=fix(double(coord_y_inicial));
arg(3)=fix(double(coord_x_final));
arg(4)=fix(double(coord_y_final));
arg(5)=fix(double(n));
arg(6)=fix(double(tipo));
% Si todas las coordenadas son 0, asignar como región de ampliado la totalidad de la
% imagen de entrada.
if arg(1:4)==0
   arg(1)=1;
```

```
   arg(2)=1;
   arg(3)=size(x,2);
   arg(4)=size(x,1);
end
% Reporta si las coordenadas están fuera de rango.
errmsg=0;
if arg(1)<1 | arg(1)>size(x,2)
   errmsg='x inicial';
elseif arg(2)<1 | arg(2)>size(x,1)
   errmsg='y inicial';
elseif arg(3)<1 | arg(3)>size(x,2)
   errmsg='x final';
elseif arg(4)<1 | arg(4)>size(x,1)
   errmsg='y final';
end
if errmsg~=0
   beep
   errordlg(sprintf('Error: Coordenada %s fuera de rango.',errmsg),'Error')
   return
end
% Reporta si el rectángulo de selección está invertido.
if arg(1)>arg(3) | arg(2)>arg(4)
   beep
   errordlg('Error: Rectángulo de selección invertido.','Error')
   return
end
try % Inicio de bloque de código con manejo de errores.
   % Selecciona mensaje para el tipo de método de ampliación.
   switch arg(6)
      case -2
         tipo_t='Reducción con promedio';
      case -1
         tipo_t='Reducción simple';
      case 0
         tipo_t='Ampliado simple';
      case 1
         tipo_t='Diagonales interpoladas, relleno en espiral';
      case 2
         tipo_t='Diagonales interpoladas, relleno radial';
      case 3
         tipo_t='Diagonales promediadas, relleno en espiral';
      case 4
         tipo_t='Diagonales promediadas, relleno radial';
      case 5
         tipo_t='Interpolación lineal, relleno en malla';
      case 6
         tipo_t='Interpolación polinomial, relleno en malla';
      case 7
         tipo_t='Interpolación con c-splines, relleno en malla';
      case 8
         tipo_t='Interpolación con b-splines, relleno en malla';
      case 9
```

```
        tipo_t='Interpolación polinomial adaptativa, relleno en malla';
    otherwise
        tipo_t='';
end
% Determina el mensaje para el tipo de acción: ampliando o reduciendo.
if arg(6)>=0
    action='Ampliando';
else
    action='Reduciendo';
end
name=sprintf('%s imagen...',action);
% Crea la barra de progreso.
h=waitbar(0,sprintf('%s "%s" x%0.f. Tipo %0.f:\n %s.',action,...
    ruta_entrada(slash_positions(size(slash_positions,2))+1:size(ruta_entrada,2)),...
    arg(5),arg(6),tipo_t),'Name',name,'CloseRequestFcn','jk_zoom_salir');
if arg(5)~=1
    % Si n = 1, se devuelve como imagen de salida la misma imagen o región de imagen
    % de entrada.

    switch arg(6)   % Selecciona el método de ampliación.
        case {-1,-2}
            t_ini=clock;    % Registra la hora del sistema.
            y=reducir_color(x,arg(2),arg(1),arg(4),arg(3),arg(5),arg(6),h,0,20/20);
            % Realiza el proceso de ampliado.

            tiempo=etime(clock,t_ini)-tiempo_muerto;
            % Calcula el tiempo transcurrido durante el proceso de ampliado.

        case 0
            t_ini=clock;    % Registra la hora del sistema.
            y=ampliar_color(x,arg(2),arg(1),arg(4),arg(3),arg(5),h,0,20/20);
            % Realiza el proceso de ampliado.

            tiempo=etime(clock,t_ini)-tiempo_muerto;
            % Calcula el tiempo transcurrido durante el proceso de ampliado.

        case {1,2,3,4}
            t_ini=clock;    % Registra la hora del sistema.
            y=asterisco_color(x,arg(2),arg(1),arg(4),arg(3),arg(5),arg(6),h,0,20/20);
            % Realiza el proceso de ampliado.

            tiempo=etime(clock,t_ini)-tiempo_muerto;
            % Calcula el tiempo transcurrido durante el proceso de ampliado.

        case {5,6,7,8,9}
            t_ini=clock;    % Registra la hora del sistema.
            y=polinomica_color(x,arg(2),arg(1),arg(4),arg(3),arg(5),arg(6),h,0,20/20);
            % Realiza el proceso de ampliado.

            tiempo=etime(clock,t_ini)-tiempo_muerto;
            % Calcula el tiempo transcurrido durante el proceso de ampliado.
```

```
        otherwise   % Si "tipo" no está en el rango de -2 a 9
           msgbox(sprintf('Error: Tipo = %0.f no reconocido o soportado',arg(6)),'Error')
           % Reporta que el tipo de ampliado no es válido.

           delete(h)
           return
      end
   else
      y=x(arg(2):arg(4),arg(1):arg(3),:);
      waitbar(20/20,h)
   end
   y=uint8(y); % Convierte matriz de imagen de salida a enteros de 8 bits.
   waitbar(20/20,h,'Guardando archivo...')
   imwrite(y,ruta_salida,extension);   % Guarda imagen de salida.
   delete(h)
   info=sprintf('%0.2f',tiempo);
catch   % Si ocurre algún error, se ejecuta lo siguiente.
   if salir==0 % O sea, si en realidad es un error.
      delete(h)
      beep, beep
      errordlg(sprintf('Error interno de programa: %s.',lasterr),'Error')
      % Reporta el mensaje de error.

   end
end
```

Función `jk_zoom_salir`

Propósito

Preguntar al usuario si desea abortar el proceso de ampliado / reducción, en cuyo caso devuelve el control del programa.

Sintaxis

```
jk_zoom_salir()
```

Comentarios

♦ En este programa se utilizan las siguientes variables globales:

- `salir`: Indicador que actúa en conjunto con `JK_zoom` para conocer si el programa finalizará debido a un error, o porque concluyó satisfactoriamente el proceso. Si salir = 1, significa que el programa ha terminado satisfactoriamente. De lo contrario ha ocurrido un error.

- `tiempo_muerto`: El tiempo muerto es el que transcurre mientras se encuentra desplegado el mensaje de confirmación de cancelación del proceso de ampliación / reducción.

Código fuente

Listado 8.2. Función `jk_zoom_salir`

```
function jk_zoom_salir()
% jk_zoom_salir() pregunta al usuario si desea abortar el proceso de ampliado / reducción,
% en cuyo caso devuelve el control del programa.

global salir tiempo_muerto
tiempo=clock;
boton_presionado=questdlg('¿Está seguro que desea cancelar el proceso de ampliado?',...
   'Abortar operación','Sí','No','No');
tiempo_muerto=tiempo_muerto+etime(clock,tiempo);
if boton_presionado=='Sí'
delete(findobj('Name','Ampliando imagen...'))
   salir=1;
end
```

Función `combinar`

Propósito

Combinar los elementos de dos matrices en una matriz.

Sintaxis

```
z = combinar(x,y)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones.
- `y`: Matriz de doble precisión de dos dimensiones.

Resultados

- `z`: Matriz de doble precisión de dos dimensiones que contiene la combinación de las matrices `x` e `y`.

Comentarios

- Las reglas de combinación para los elementos de las matrices `x` e `y` en posiciones iguales son las siguientes:
  - Cero combinado con un elemento no nulo, es ese elemento.
  - Para dos elementos iguales combinados se devuelve el mismo elemento.
  - Para dos elementos diferentes combinados se devuelve el promedio de ambos.
- `x` e `y` deben ser del mismo tamaño.

Código fuente

Listado 8.3. Función `combinar`

```
function z=combinar(x,y)
% Función que combina los elementos de dos matrices x e y en la matriz z.
% Sintaxis: z=combinar(x,y).
% 0 combinado con un elemento, es ese elemento.
% Dos elementos iguales combinados, devuelve el mismo elemento.
% Dos elementos diferentes devuelve el promedio de ambos.
% x e y deben ser del mismo tamaño.

z=zeros(size(x));
for f=1:size(x,1)   % Para las filas
   for c=1:size(x,2)   % Para las columnas
      if x(f,c)~=0
         if y(f,c)~=0
            if x(f,c)==y(f,c)
               z(f,c)=x(f,c);
            else
               z(f,c)=(x(f,c)+y(f,c))/2;
            end
         else
            z(f,c)=x(f,c);
         end
      else
         z(f,c)=y(f,c);
      end
   end
end
```

B.2. Métodos de reducción

Función `reducir_color`

Propósito

Reducir una región de una imagen digital.

Sintaxis

```
y = reducir_color(x,fila_i,col_i,fila_f,col_f,n,tipo)
```

Argumentos

- `x`: Matriz de doble precisión de dos o tres dimensiones que contiene una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.
- `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- `n`: Número entero positivo que indica la cantidad de reducción.
- `tipo`: Número entero que indica el tipo de método a utilizar para la reducción. Si tipo = -1, se coloca el píxel central de cada celda en la imagen reducida. Si tipo = -2, se coloca el promedio de los píxeles de cada celda en la imagen reducida.

Resultados

- `y`: Matriz de doble precisión que contiene la región de `x` reducida. El número de dimensiones de `y` es igual al de `x`.

Comentarios

- Si n es menor que el número de filas o de columnas de la región de la imagen original a ser reducida, n se hace igual a dicho número de filas o de columnas (el menor de ellos).

Funciones empleadas

- `reducir`

Código fuente

Listado 8.4. Función `reducir_color`

```
function y=reducir_color(x,fila_i,col_i,fila_f,col_f,n,tipo,h,acumulado,subtotal)
% y=reducir_color(x,fila_i,col_i,fila_f,col_f,n,tipo) reduce una subimagen de x (RGB).
% n es la cantidad de reducción.
% La subimagen está constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina superior derecha (fila_f,col_f).
% Si tipo = -1, se coloca el píxel central de cada celda en la imagen reducida.
% Si tipo = -2, se coloca el promedio de los píxeles de cada celda en la imagen reducida.
% Si n es menor que el número de filas o de columnas de la región de la imagen original a
% ser reducida, n se hace igual a dicho número de filas o de columnas (el menor de ellos).

if n>fila_f-fila_i+1 | n>col_f-col_i+1
   n=min(size(x,1),size(x,2));
end
subtotal_local=subtotal/size(x,3);
y=zeros(floor((fila_f-fila_i+1)/n),floor((col_f-col_i+1)/n),size(x,3));
for i=1:size(x,3)   % Para cada canal de color de la imagen.
   acumulado_local=subtotal*(i-1)/size(x,3);
   y(:,:,i)=reducir(x(:,:,i),fila_i,col_i,fila_f,col_f,n,tipo);
   waitbar(acumulado+acumulado_local,h)
end
```

Función `reducir`

Propósito

Reducir un canal de color de una región de una imagen digital.

Sintaxis

```
y = reducir(x,fila_i,col_i,fila_f,col_f,n,tipo)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.
- `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- `n`: Número entero positivo que indica la cantidad de reducción.
- `tipo`: Número entero que indica el tipo de método a utilizar para la reducción. Si tipo = -1, se coloca el píxel central de cada celda en la imagen reducida. Si tipo = -2, se coloca el promedio de los píxeles de cada celda en la imagen reducida.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene la región de `x` reducida.

Comentarios

- Si n es menor que el número de filas o de columnas de la región de la imagen original a ser reducida, n se hace igual a dicho número de filas o de columnas (el menor de ellos).

Código fuente

Listado 8.5. Función `reducir`

```
function y=reducir(x,fila_i,col_i,fila_f,col_f,n,tipo)
% y=reducir(x,fila_i,col_i,fila_f,col_f,n,tipo) reduce el tamaño de una subimagen de x
% (escala de grises).
```

```
% La subimagen está constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina inferior derecha (fila_f,col_f).
% n es la cantidad de reducción. No puede ser menor que el número de filas o de columnas
% de la región de la imagen original a ser reducida.

x=x(fila_i:fila_f,col_i:col_f);
x=double(x);
nfil=floor(size(x,1)/n); % Número de filas
ncol=floor(size(x,2)/n); % Número de columnas
y=zeros(nfil,ncol);
if tipo==-1 % Si tipo = -1, se utiliza el píxel central de cada celda.
   for f=1:nfil
      for c=1:ncol
         y(f,c)=x((f-1)*n+ceil(n/2),(c-1)*n+ceil(n/2));
      end
   end
else    % Si tipo = -2, se utiliza el promedio de todos los píxeles de la celda.
   for f=1:size(y,1)
      for c=1:size(y,2)
         y(f,c)=sum(sum(x((f-1)*n+1:f*n,(c-1)*n+1:c*n)))/n^2;
      end
   end
end
```

B.3. Ampliado simple

Función `ampliar_color`

Propósito

Ampliar una región de una imagen digital con ampliado simple.

Sintaxis

```
y = ampliar_color(x,fila_i,col_i,fila_f,col_f,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos o tres dimensiones que contiene una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.

- ♦ `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- ♦ `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- ♦ `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- ♦ `y`: Matriz de doble precisión que contiene la región de `x` ampliada. El número de dimensiones de `y` es igual al de `x`.

Funciones empleadas

- ♦ `ampliar`

Código fuente

Listado 8.6. Función `ampliar_color`

```
function y=ampliar_color(x,fila_i,col_i,fila_f,col_f,n,h,acumulado,subtotal)
% y=ampliar_color(x,fila_i,col_i,fila_f,col_f,n) amplifica una subimagen de x (RGB). n es
% la dimensión de la región cuadrada que reemplazará a cada píxel.
% La subimagen está constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina inferior derecha (fila_f,col_f).

subtotal_local=subtotal/size(x,3);
y=zeros((fila_f-fila_i+1)*n,(col_f-col_i+1)*n,size(x,3));
for i=1:size(x,3)   % Para cada canal de color de la imagen.
   acumulado_local=subtotal*(i-1)/size(x,3);
   y(:,:,i)=ampliar(x(:,:,i),fila_i,col_i,fila_f,col_f,n);
   waitbar(acumulado+acumulado_local,h)
end
```

Función `ampliar`

Propósito

Ampliar un canal de color de una región de una imagen digital con ampliado simple.

Sintaxis

```
y = ampliar(x,fila_i,col_i,fila_f,col_f,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.
- `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene la región de `x` ampliada.

Código fuente

Listado 8.7. Función `ampliar`

```
function y=ampliar(x,fila_i,col_i,fila_f,col_f,n)
% y=ampliar(x,fila_i,col_i,fila_f,col_f,n) amplifica una subimagen de x (escala de
% grises).
% La subimagen está constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina inferior derecha (fila_f,col_f).
% n es la dimensión de la región cuadrada que reemplazará a cada píxel.

x=x(fila_i:fila_f,col_i:col_f);
x=double(x);
y=zeros(size(x,1)*n,size(x,2)*n);
fx=1;
cx=1;
a=1;
```

```
for f=1:n:size(x,1)*n-n+1   % Para las filas
  for c=1:n:size(x,2)*n-n+1    % Para las columnas
    y(f:f+n-1,c:c+n-1)=x(fx,cx)*ones(n,n);
    cx=cx+1;
 end
 cx=1;
 fx=fx+1;
end
```

B.4. Algoritmo del asterisco

Función `asterisco_color`

Propósito

Ampliar una región de una imagen digital con el algoritmo del asterisco.

Sintaxis

```
y = asterisco_color(x,fila_i,col_i,fila_f,col_f,n,tipo)
```

Argumentos

- `x`: Matriz de doble precisión de dos o tres dimensiones que contiene una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.
- `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- `n`: Número entero positivo que indica la cantidad de ampliación.
- `tipo`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.2.

Tabla 8.2. Métodos de ampliación con el algoritmo del asterisco

Tipo	Descripción
1	Diagonales interpoladas con los vecinos diagonales, relleno en espiral
2	Diagonales interpoladas con los vecinos diagonales, relleno radial
3	Diagonales interpoladas con las verticales y horizontales, relleno en espiral
4	Diagonales interpoladas con las verticales y horizontales, relleno radial

Resultados

- `y`: Matriz de doble precisión que contiene la región de `x` ampliada. El número de dimensiones de `y` es igual al de `x`.

Funciones empleadas

- `asterisco`

Código fuente

Listado 8.8. Función `asterisco_color`

```
function y=asterisco_color(x,fila_i,col_i,fila_f,col_f,n,tipo,h,acumulado,subtotal)
% y=asterisco_color(x,fila_i,col_i,fila_f,n,tipo) devuelve en "y" una submatriz de "x"
% ampliada "n" veces, utilizando el algoritmo del asterisco para la ampliación.
% La submatriz está constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina inferior derecha (fila_f,col_f).
% "x" puede ser una matriz de n X M o de n X M X 3, donde n es el número de filas, M es el
% número de columnas, y 3 es el número de componentes de color en RGB.
% "tipo" indica qué variante del algoritmo se usará:
% tipo = 1: Diagonales interpoladas con los vecinos diagonales. Relleno en espiral.
% tipo = 2: Diagonales interpoladas con los vecinos diagonales. Relleno radial.
% tipo = 3: Diagonales interpoladas con las verticales y horizontales. Relleno en espiral.
% tipo = 4: Diagonales interpoladas con las verticales y horizontales. Relleno radial.

subtotal_local=subtotal/size(x,3);
y=zeros((fila_f-fila_i+1)*n,(col_f-col_i+1)*n,size(x,3));
for i=1:size(x,3)   % Para cada canal de color de la imagen
   acumulado_local=subtotal*(i-1)/size(x,3);
   y(:,:,i)=asterisco(x(:,:,i),fila_i,col_i,fila_f,col_f,n,tipo,h,acumulado_local,subtotal_local);
   waitbar(acumulado+acumulado_local,h)
end
```

Función `asterisco`

Propósito

Ampliar un canal de color de una región de una imagen digital con el algoritmo del asterisco.

Sintaxis

```
y = asterisco(x,fila_i,col_i,fila_f,col_f,n,tipo)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.
- `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- `n`: Número entero positivo que indica la cantidad de ampliación.
- `tipo`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.2.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene la región de `x` ampliada.

Funciones empleadas

- ♦ vertical
- ♦ horizontal
- ♦ diag_izq
- ♦ diag_der
- ♦ diag_interp
- ♦ rellenar_espiral
- ♦ rellenar_radial
- ♦ combinar

Código fuente

Listado 8.9. Función asterisco

```
function y=asterisco(x,fila_i,col_i,fila_f,col_f,n,tipo,hh,acumulado,subtotal)
% y=asterisco(x,fila_i,col_i,fila_f,col_f,n,tipo) devuelve en "y" una submatriz de "x"
% ampliada "n" veces, utilizando el algoritmo del asterisco para la ampliación.
% La submatriz está constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina inferior derecha (fila_f,col_f).
% "tipo" indica qué variante del algoritmo se usará:
% tipo = 1: Diagonales interpoladas con los vecinos diagonales. Relleno en espiral.
% tipo = 2: Diagonales interpoladas con los vecinos diagonales. Relleno radial.
% tipo = 3: Diagonales interpoladas con las verticales y horizontales. Relleno en espiral.
% tipo = 4: Diagonales interpoladas con las verticales y horizontales. Relleno radial.

x=x(fila_i:fila_f,col_i:col_f);
z=zeros(size(x,1)*n,size(x,2)*n);
mg=z;
z1=z;
acumulado_local=acumulado;
subtotal_local=0.14*subtotal;
v=vertical(x,n,hh,acumulado_local,subtotal_local);  % Encuentra las verticales del enrejado
waitbar(acumulado+0.14*subtotal,hh)
acumulado_local=acumulado+0.14*subtotal;
subtotal_local=0.15*subtotal;
mg(1:size(v,1),1:size(v,2))=v;
h=horizontal(x,n,hh,acumulado_local,subtotal_local);    % Encuentra las horizontales del enrejado
waitbar(acumulado+0.29*subtotal,hh)
acumulado_local=acumulado+0.29*subtotal;
subtotal_local=0.03*subtotal;
z1(1:size(h,1),1:size(h,2))=h;
mg=combinar(mg,z1); % Une las verticales y horizontales para crear el enrejado
waitbar(acumulado+0.32*subtotal,hh)
if tipo<3
   acumulado_local=acumulado+0.32*subtotal;
   subtotal_local=0.04*subtotal;
   di=diag_izq(x,n,hh,acumulado_local,subtotal_local); % Encuentra las diagonales izquierda
   waitbar(acumulado+0.36*subtotal,hh)
   acumulado_local=acumulado+0.36*subtotal;
   subtotal_local=0.06*subtotal;
   z1=z;
   z1(1:size(di,1),1:size(di,2))=di;
   mg=combinar(mg,z1); % Combina las diagonales izquierda con el enrejado
   waitbar(acumulado+0.42*subtotal,hh)
   acumulado_local=acumulado+0.42*subtotal;
   subtotal_local=0.04*subtotal;
```

```
   dd=diag_der(x,n,hh,acumulado_local,subtotal_local); % Encuentra las diagonales derecha
   waitbar(acumulado+0.46*subtotal,hh)
   acumulado_local=acumulado+0.46*subtotal;
   subtotal_local=0.06*subtotal;
   z1=z;
   z1(1:size(dd,1),1:size(dd,2))=dd;
   mg=combinar(mg,z1); % Combina las diagonales derecha con el enrejado
   waitbar(acumulado+0.52*subtotal,hh)
else
   acumulado_local=acumulado+0.32*subtotal;
   subtotal_local=0.18*subtotal;
   d=diag_interp(x,mg,n,hh,acumulado_local,subtotal_local);
   % Encuentra las diagonales interpoladas

   waitbar(acumulado+0.5*subtotal,hh)
   acumulado_local=acumulado+0.5*subtotal;
   subtotal_local=0.02*subtotal;
   mg=combinar(mg,d);  % Combina las diagonales interpoladas con el enrejado
   waitbar(acumulado+0.52*subtotal,hh)
end
if tipo==1 | tipo==3
   acumulado_local=acumulado+0.52*subtotal;
   subtotal_local=0.48*subtotal;
   mgr=rellenar_espiral(x,mg,n,hh,acumulado_local,subtotal_local); % Realiza relleno en espiral
   waitbar(acumulado+subtotal,hh)
else
   acumulado_local=acumulado+0.52*subtotal;
   subtotal_local=0.48*subtotal;
   mgr=rellenar_radial(x,mg,n,hh,acumulado_local,subtotal_local);  % Realiza relleno radial
   waitbar(acumulado+subtotal,hh)
end
y=mgr;
```

Función `vertical`

Propósito

Rellenar las verticales del enrejado con el algoritmo del asterisco.

Sintaxis

```
y = vertical(x,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene las verticales del enrejado de la región de `x` ampliada.

Código fuente

Listado 8.10. Función `vertical`

```
function y=vertical(x,n,h,acumulado,subtotal)
% y=vertical(x,n) rellena las verticales en el algoritmo del asterisco.
% x es una matriz de dos dimensiones.
% n es el número de píxeles por lado que conforman el píxel ampliado.

y=zeros(size(x)*n);
for fx=1:size(x,1)-1    % Para las filas
  for cx=1:size(x,2)   % Para las columnas
     y(fx*n-floor(n/2):(fx+1)*n-floor(n/2),cx*n-floor(n/2))=linspace(x(fx,cx),x(fx+1,cx),n+1)';
     waitbar(acumulado+subtotal*(size(x,2)*(fx-1)+cx)/((size(x,1)-1)*(size(x,2))),h)
  end
end
```

Función `horizontal`

Propósito

Rellenar las horizontales del enrejado con el algoritmo del asterisco.

Sintaxis

```
y = horizontal(x,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene las horizontales del enrejado de la región de `x` ampliada.

Código fuente

Listado 8.11. Función `horizontal`

```
function y=horizontal(x,n,h,acumulado,subtotal)
% y=horizontal(x,n) rellena las horizontales en el algoritmo del asterisco.
% x es una matriz de dos dimensiones.
% n es el número de píxeles por lado que conforman el píxel ampliado.

y=zeros(size(x)*n);
for fx=1:size(x,1)  % Para las filas
  for cx=1:size(x,2)-1 % Para las columnas
     y(fx*n-floor(n/2),cx*n-floor(n/2):(cx+1)*n-floor(n/2))=linspace(x(fx,cx),x(fx,cx+1),n+1);
     waitbar(acumulado+subtotal*((size(x,2)-1)*(fx-1)+cx)/(size(x,1)*(size(x,2)-1)),h)
  end
end
```

Función `diag_izq`

Propósito

Rellenar las diagonales izquierda con el algoritmo del asterisco, interpolando linealmente con los vecinos diagonales.

Sintaxis

```
y = diag_izq(x,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene las diagonales izquierda de la región de `x` ampliada.

Código fuente

Listado 8.12. Función `diag_izq`

```
function y=diag_izq(x,n,h,acumulado,subtotal)
% y=diag_izq(x,n) rellena las diagonales izquierda en el algoritmo del asterisco.
```

```
% x es una matriz de dos dimensiones.
% n es el número de píxeles por lado que conforman el píxel ampliado.

y=zeros(size(x)*n);
for fx=1:size(x,1)-1    % Para las filas
  for cx=1:size(x,2)-1 % Para las columnas
    indf=fx*n-floor(n/2)+1;
    indc=cx*n-floor(n/2)+1;
    valores=linspace(x(fx,cx),x(fx+1,cx+1),n+1);
    valores=valores(2:size(valores,2)-1);
    for j=1:n-1
      y(indf,indc)=valores(j);
      indf=indf+1;
      indc=indc+1;
    end
  end
  waitbar(acumulado+subtotal*fx/(size(x,1)-1),h)
end
```

Función `diag_der`

Propósito

Rellenar las diagonales derecha con el algoritmo del asterisco, interpolando linealmente con los vecinos diagonales.

Sintaxis

```
y = diag_der(x,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene las diagonales derecha de la región de `x` ampliada.

Código fuente

Listado 8.13. Función `diag_der`

```
function y=diag_der(x,n,h,acumulado,subtotal)
% y=diag_der(x,n) rellena las diagonales derecha en el algoritmo del asterisco.
% x es una matriz de dos dimensiones.
% n es el número de píxeles por lado que conforman el píxel ampliado.

y=zeros(size(x)*n);
for fx=1:size(x,1)-1    % Para las filas
  for cx=1:size(x,2)-1 % Para las columnas
    indf=fx*n-floor(n/2)+1;
    indc=(cx+1)*n-floor(n/2)-1;
    valores=linspace(x(fx,cx+1),x(fx+1,cx),n+1);
    valores=valores(2:size(valores,2)-1);
    for j=1:n-1
      y(indf,indc)=valores(j);
      indf=indf+1;
      indc=indc-1;
    end
  end
  waitbar(acumulado+subtotal*fx/(size(x,1)-1),h)
end
```

Función `diag_interp`

Propósito

Rellenar las diagonales con el algoritmo del asterisco, promediando las interpolaciones lineales de filas y columnas.

Sintaxis

```
y = diag_interp(x,xa,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `xa`: Matriz a la que se le ha aplicado `vertical.M` y `horizontal.M`.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene las diagonales de la región de `x` ampliada.

Código fuente

Listado 8.14. Función `diag_interp`

```
function y=diag_interp(x,xa,n,h,acumulado,subtotal)
% y=diag_interp(x,xa,n) rellena las diagonales en el algoritmo del asterisco, promediando
% las interpolaciones lineales de filas y columnas.
% x es una matriz de dos dimensiones.
% xa es la matriz a la que se le ha aplicado vertical.M y horizontal.M.
% n es el número de píxeles por lado que conforman el píxel ampliado.
% n puede ser par o impar.

y=xa;
for fx=1:size(x,1)-1    % Para las filas
  for cx=1:size(x,2)-1 % Para las columnas
     if mod(n,2)~=0   % Si la cantidad de ampliado es impar
        fi=fx*n-(n-1)/2;
        ff=(fx+1)*n-(n-1)/2;
        ci=cx*n-(n-1)/2;
        cf=(cx+1)*n-(n-1)/2;
        for j=1:(n-1)/2
           vert1=linspace(xa(fi,ci+j),xa(ff,ci+j),n+1);
           vert2=linspace(xa(fi,cf-j),xa(ff,cf-j),n+1);
           horiz1=linspace(xa(fi+j,ci),xa(fi+j,cf),n+1);
           horiz2=linspace(xa(ff-j,ci),xa(ff-j,cf),n+1);
           y(fi+j,ci+j)=(vert1(j+1)+horiz1(j+1))/2;
           y(ff-j,ci+j)=(vert1(n+1-j)+horiz2(j+1))/2;
           y(fi+j,cf-j)=(vert2(j+1)+horiz1(n+1-j))/2;
           y(ff-j,cf-j)=(vert2(n+1-j)+horiz2(n+1-j))/2;
        end
     else % Si la cantidad de ampliado es par
        fi=fx*n-n/2;
        ff=(fx+1)*n-n/2;
        ci=cx*n-n/2;
        cf=(cx+1)*n-n/2;
        for j=1:n/2-1
           vert1=linspace(xa(fi,ci+j),xa(ff,ci+j),n+1);
           vert2=linspace(xa(fi,cf-j),xa(ff,cf-j),n+1);
           horiz1=linspace(xa(fi+j,ci),xa(fi+j,cf),n+1);
           horiz2=linspace(xa(ff-j,ci),xa(ff-j,cf),n+1);
           y(fi+j,ci+j)=(vert1(j+1)+horiz1(j+1))/2;
           y(ff-j,ci+j)=(vert1(n+1-j)+horiz2(j+1))/2;
           y(fi+j,cf-j)=(vert2(j+1)+horiz1(n+1-j))/2;
           y(ff-j,cf-j)=(vert2(n+1-j)+horiz2(n+1-j))/2;
        end
        vert=linspace(xa(fi,ci+n/2-1+1),xa(ff,ci+n/2-1+1),n+1);
```

```
        horiz=linspace(xa(fi+n/2-1+1,ci),xa(fi+n/2-1+1,cf),n+1);
        y(fi+n/2-1+1,ci+n/2-1+1)=(vert(n/2)+horiz(n/2))/2;
      end
      waitbar(acumulado+subtotal*((size(x,2)-1)*(fx-1)+cx)/((size(x,1)-1)*(size(x,2)-1)),h)
    end
end
```

Función `rellenar_espiral`

Propósito

Rellenar los píxeles intermedios del enrejado con el algoritmo del asterisco, siguiendo una forma en espiral.

Sintaxis

```
y = rellenar_espiral(x,xa,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `xa`: Matriz a la que se le ha aplicado `vertical.M` y `horizontal.M`.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene los píxeles intermedios del enrejado de la región de `x` ampliada.

Código fuente

Listado 8.15. Función `rellenar_espiral`

```
function y=rellenar_espiral(x,xa,n,h,acumulado,subtotal)
% y=rellenar_espiral(x,xa,n) rellena los píxeles intermedios en el algoritmo del
% asterisco, siguiendo una forma espiral.
% x es la matriz de dos dimensiones original.
% xa es la matriz ampliada, luego de haber sido procesada por vertical.M, horizontal.M,
% diag_izq.M y diag_der.M.
% n es el número de píxeles por lado que conforman el píxel ampliado.

y=xa;
```

```
if mod(n,2)==0  % Si la cantidad de ampliado es par
   for fx=1:size(x,1)-1
      for cx=1:size(x,2)-1
         %Para rellenar semitriángulo superior izquierdo
         fi=fx*n-n/2;
         indc=cx*n-n/2+2;
         indf=fi+2;
         for j=1:n/2-1
            y(fi:indf,indc)=linspace(y(fi,indc),y(indf,indc),j+2)';
            indf=indf+1;
            indc=indc+1;
         end
         %Para rellenar semitriángulo superior derecho
         indf=indf-2;
         for j=1:n/2-2
            y(fi:indf,indc)=linspace(y(fi,indc),y(indf,indc),n/2-2-j+3)';
            indf=indf-1;
            indc=indc+1;
         end
         %Para rellenar semitrángulo inferior izquierdo
         fi=(fx+1)*n-n/2;
         indc=cx*n-n/2+2;
         indf=fi-2;
         for j=1:n/2-1
            y(indf:fi,indc)=linspace(y(indf,indc),y(fi,indc),j+2)';
            indf=indf-1;
            indc=indc+1;
         end
         %Para rellenar semitrángulo inferior derecho
         indf=indf+2;
         for j=1:n/2-2
            y(indf:fi,indc)=linspace(y(indf,indc),y(fi,indc),n/2-2-j+3)';
            indf=indf+1;
            indc=indc+1;
         end
         %Para rellenar semitrangulo izquierdo superior
         indf=fx*n-n/2+2;
         ci=cx*n-n/2;
         indc=ci+2;
         for j=1:n/2-1
            y(indf,ci:indc)=linspace(y(indf,ci),y(indf,indc),j+2);
            indf=indf+1;
            indc=indc+1;
         end
         %Para rellenar semitrángulo izquierdo inferior
         indc=indc-2;
         for j=1:n/2-2
            y(indf,ci:indc)=linspace(y(indf,ci),y(indf,indc),n/2-2-j+3);
            indf=indf+1;
            indc=indc-1;
         end
         %Para rellenar semitrángulo derecho superior
```

```
         indf=fx*n-n/2+2;
         ci=(cx+1)*n-n/2;
         indc=ci-2;
         for j=1:n/2-1
            y(indf,indc:ci)=linspace(y(indf,indc),y(indf,ci),j+2);
            indf=indf+1;
            indc=indc-1;
         end
         %Para rellenar semitrángulo derecho inferior
         indc=indc+2;
         for j=1:n/2-2
            y(indf,indc:ci)=linspace(y(indf,indc),y(indf,ci),n/2-2-j+3);
            indf=indf+1;
            indc=indc+1;
         end
         waitbar(acumulado+subtotal*((size(x,2)-1)*(fx)+cx)/((size(x,1)-1)*(size(x,2)-1)),h)
      end
   end
else    % Si la cantidad de ampliado es impar
   for fx=1:size(x,1)-1
      for cx=1:size(x,2)-1
         %Para rellenar semitriángulo superior izquierdo
         fi=fx*n-(n-1)/2;
         indc=cx*n-(n-1)/2+2;
         indf=fi+2;
         for j=1:(n-1)/2-1
            y(fi:indf,indc)=linspace(y(fi,indc),y(indf,indc),j+2)';
            indf=indf+1;
            indc=indc+1;
         end
         %Para rellenar semitriángulo superior derecho
         indf=indf-1;
         for j=1:(n-1)/2-1
            y(fi:indf,indc)=linspace(y(fi,indc),y(indf,indc),(n-1)/2-1-j+3)';
            indf=indf-1;
            indc=indc+1;
         end
         %Para rellenar semitrángulo inferior izquierdo
         fi=(fx+1)*n-(n-1)/2;
         indc=cx*n-(n-1)/2+2;
         indf=fi-2;
         for j=1:(n-1)/2-1
            y(indf:fi,indc)=linspace(y(indf,indc),y(fi,indc),j+2)';
            indf=indf-1;
            indc=indc+1;
         end
         %Para rellenar semitrángulo inferior derecho
         indf=indf+1;
         for j=1:(n-1)/2-1
            y(indf:fi,indc)=linspace(y(indf,indc),y(fi,indc),(n-1)/2-1-j+3)';
            indf=indf+1;
            indc=indc+1;
```

```
        end
        %Para rellenar semitrángulo izquierdo superior
        indf=fx*n-(n-1)/2+2;
        ci=cx*n-(n-1)/2;
        indc=ci+2;
        for j=1:(n-1)/2-1
           y(indf,ci:indc)=linspace(y(indf,ci),y(indf,indc),j+2);
           indf=indf+1;
           indc=indc+1;
        end
        %Para rellenar semitrángulo izquierdo inferior
        indc=indc-1;
        for j=1:(n-1)/2-1
           y(indf,ci:indc)=linspace(y(indf,ci),y(indf,indc),(n-1)/2-1-j+3);
           indf=indf+1;
           indc=indc-1;
        end
        %Para rellenar semitrángulo derecho superior
        indf=fx*n-(n-1)/2+2;
        ci=(cx+1)*n-(n-1)/2;
        indc=ci-2;
        for j=1:(n-1)/2-1
           y(indf,indc:ci)=linspace(y(indf,indc),y(indf,ci),j+2);
           indf=indf+1;
           indc=indc-1;
        end
        %Para rellenar semitrángulo derecho inferior
        indc=indc+1;
        for j=1:(n-1)/2-1
           y(indf,indc:ci)=linspace(y(indf,indc),y(indf,ci),(n-1)/2-1-j+3);
           indf=indf+1;
           indc=indc+1;
        end
        waitbar(acumulado+subtotal*((size(x,2)-1)*(fx)+cx)/((size(x,1)-1)*(size(x,2)-1)),h)
     end
  end
end
```

Función `rellenar_radial`

Propósito

Rellenar los píxeles intermedios del enrejado con el algoritmo del asterisco, siguiendo una forma radial.

Sintaxis

```
y = rellenar_radial(x,xa,n)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `xa`: Matriz a la que se le ha aplicado `vertical.M` y `horizontal.M`.
- `n`: Número entero positivo que indica la cantidad de ampliación.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene los píxeles intermedios del enrejado de la región de `x` ampliada.

Código fuente

Listado 8.16. Función `rellenar_radial`

```
function y=rellenar_radial(x,xa,n,h,acumulado,subtotal)
% y=rellenar_radial(x,xa,n) rellena los píxeles intermedios en el algoritmo del asterisco,
% siguiendo una forma radial.
% x es la matriz de dos dimensiones original.
% xa es la matriz ampliada, luego de haber sido procesada por vertical.M, horizontal.M,
% diag_izq.M y diag_der.M.
% n es el número de píxeles por lado que conforman el píxel ampliado.

y=xa;
if mod(n,2)==0  % Si la cantidad de ampliado es par
   for fx=1:size(x,1)-1
      for cx=1:size(x,2)-1
         %Para rellenar triángulo superior
         fi=fx*n-n/2;
         ci=cx*n-n/2;
         cf=(cx+1)*n-n/2;
         for j=1:n/2-1
            y(fi+j,ci+j:cf-j)=linspace(y(fi+j,ci+j),y(fi+j,cf-j),n-1-2*(j-1));
         end
         %Para rellenar triángulo inferior
         fi=fi+j+1;
         for j=1:n/2-1
            y(fi+j,ci+n/2-j:cf-n/2+j)=linspace(y(fi+j,ci+n/2-j),y(fi+j,cf-n/2+j),2*j+1);
         end
         %Para rellenar triángulo izquierdo
         fi=fx*n-n/2;
         ff=(fx+1)*n-n/2;
         for j=1:n/2-1
            y(fi+j:ff-j,ci+j)=linspace(y(fi+j,ci+j),y(ff-j,ci+j),n-1-2*(j-1))';
         end
```

```
        %Para rellenar triángulo derecho
        ci=ci+j+1;
        for j=1:n/2-1
           y(fi+n/2-j:ff-n/2+j,ci+j)=linspace(y(fi+n/2-j,ci+j),y(ff-n/2+j,ci+j),2*j+1)';
        end
        waitbar(acumulado+subtotal*((size(x,2)-1)*(fx-1)+cx)/((size(x,1)-1)*(size(x,2)-1)),h)
     end
  end
else   % Si la cantidad de ampliado es impar
  for fx=1:size(x,1)-1
     for cx=1:size(x,2)-1
        %Para rellenar triángulo superior
        fi=fx*n-(n-1)/2;
        ci=cx*n-(n-1)/2;
        cf=(cx+1)*n-(n-1)/2;
        for j=1:(n-1)/2-1
           y(fi+j,ci+j:cf-j)=linspace(y(fi+j,ci+j),y(fi+j,cf-j),n-1-2*(j-1));
        end
        %Para rellenar triángulo inferior
        fi=fi+j+2;
        for j=1:(n-1)/2-1
           y(fi+j,ci+(n-1)/2-j:cf-(n-1)/2+j)=linspace(y(fi+j,ci+(n-1)/2-j),y(fi+j,cf-(n-1)/2+j),2*j+2);
        end
        %Para rellenar triángulo izquierdo
        fi=fx*n-(n-1)/2;
        ff=(fx+1)*n-(n-1)/2;
        for j=1:(n-1)/2-1
           y(fi+j:ff-j,ci+j)=linspace(y(fi+j,ci+j),y(ff-j,ci+j),n-1-2*(j-1))';
        end
        %Para rellenar triángulo derecho
        ci=ci+j+2;
        for j=1:(n-1)/2-1
           y(fi+(n-1)/2-j:ff-(n-1)/2+j,ci+j)=linspace(y(fi+(n-1)/2-j,ci+j),y(ff-(n-1)/2+j,ci+j),2*j+2)';
        end
        waitbar(acumulado+subtotal*((size(x,2)-1)*(fx-1)+cx)/((size(x,1)-1)*(size(x,2)-1)),h)
     end
  end
end
```

B.5. Métodos polinómicos

Función `polinomica_color`

Propósito

Ampliar una región de una imagen digital utilizando algún tipo de interpolación polinómica.

Sintaxis

```
y = polinomica_color(x,fila_i,col_i,fila_f,col_f,n,método)
```

Argumentos

- `x`: Matriz de doble precisión de dos o tres dimensiones que contiene una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.
- `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- `n`: Número entero positivo que indica la cantidad de ampliación.
- `método`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.3.

Tabla 8.3. Métodos de ampliación que utilizan interpolación polinómica

Tipo	Descripción
5	Interpolación lineal, relleno en malla
6	Interpolación polinomial, relleno en malla
7	Interpolación con c-splines, relleno en malla
8	Interpolación con b-splines, relleno en malla
9	Interpolación polinomial adaptativa, relleno en malla

Resultados

- `y`: Matriz de doble precisión que contiene la región de `x` ampliada. El número de dimensiones de `y` es igual al de `x`.

Comentarios

- Los valores para el argumento `método` mostrados en la Tabla 8.3 son válidos también para las funciones `polinomica`, `polinomica_vertical`, `polinomica_horizontal`, y `polinomica_rellenar_promedio`.

Funciones empleadas

- `polinomica`

Código fuente

Listado 8.17. Función `polinomica_color`

```
function y=polinomica_color(x,fila_i,col_i,fila_f,col_f,n,metodo,h,acumulado,subtotal)
% y=polinomica_color(x,fila_i,col_i,fila_f,n,método) devuelve en "y" una submatriz de "x"
% ampliada "n" veces, utilizando la interpolación especificada por "metodo" para la
% ampliación.
% La submatriz esta constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina inferior derecha (fila_f,col_f).
% "x" puede ser una matriz de n X M o de n X M X 3, donde n es el número de filas, M es el
% número de columnas, y 3 es el número de componentes de color en RGB.
% Los valores para "método" son los siguientes:
% método = 5: interpolación lineal.
% método = 6: interpolación polinomial.
% método = 7: interpolación con c-splines.
% método = 8: interpolación con b-splines.
% método = 9: interpolación polinomial adaptativa.

subtotal_local=subtotal/size(x,3);
y=zeros((fila_f-fila_i+1)*n,(col_f-col_i+1)*n,size(x,3));
for i=1:size(x,3)   % Para cada canal de color de la imagen
   acumulado_local=subtotal*(i-1)/size(x,3);
   y(:,:,i)=polinomica(x(:,:,i),fila_i,col_i,fila_f,col_f,n,metodo,h,acumulado_local,subtotal_local);
   waitbar(acumulado+acumulado_local,h)
end
```

Función `polinomica`

Propósito

Ampliar un canal de color de una región de una imagen digital con algún método de interpolación polinómica.

Sintaxis

```
y = polinomica(x,fila_i,col_i,fila_f,col_f,n,método)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una imagen digital en escala de grises o en RGB, respectivamente.
- `col_i`: Número entero positivo que representa la posición de columna donde inicia la región a ser procesada en una imagen.
- `fila_i`: Número entero positivo que representa la posición de fila donde inicia la región a ser procesada en una imagen.
- `col_f`: Número entero positivo que representa la posición de columna donde termina la región a ser procesada en una imagen.
- `fila_f`: Número entero positivo que representa la posición de fila donde termina la región a ser procesada en una imagen.
- `n`: Número entero positivo que indica la cantidad de ampliación.
- `método`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.3.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene la región de `x` ampliada.

Funciones empleadas

- `polinomica_vertical`
- `polinomica_horizontal`
- `polinomica_rellenar_malla`
- `combinar`

Código fuente

Listado 8.18. Función `polinomica`

```
function y=polinomica(x,fila_i,col_i,fila_f,col_f,n,metodo,hh,acumulado,subtotal)
```

```
% y=polinomica(x,fila_i,col_i,fila_f,n,método) devuelve en "y" una submatriz de "x"
% ampliada "n" veces, utilizando la interpolación especificada por "metodo" para la
% ampliación.
% La submatriz esta constituida por la región rectangular definida por los puntos de la
% esquina superior izquierda (fila_i,col_i) y la esquina inferior derecha (fila_f,col_f).
% Los valores para "método" son los siguientes:
% método = 5: interpolación lineal.
% método = 6: interpolación polinomial.
% método = 7: interpolación con c-splines.
% método = 8: interpolación con b-splines.
% método = 9: interpolación polinomial adaptativa.

x=x(fila_i:fila_f,col_i:col_f);
acumulado_local=acumulado;
subtotal_local=0.03*subtotal;
v=polinomica_vertical(x,n,metodo,hh,acumulado_local,subtotal_local);
% Encuentra las verticales del enrejado

waitbar(acumulado+0.03*subtotal,hh)
acumulado_local=acumulado+0.03*subtotal;
subtotal_local=0.03*subtotal;
h=polinomica_horizontal(x,n,metodo,hh,acumulado_local,subtotal_local);
% Encuentra las horizontales del enrejado

waitbar(acumulado+0.06*subtotal,hh)
acumulado_local=acumulado+0.06*subtotal;
subtotal_local=0.03*subtotal;
xa=combinar(v,h);   % Combina las verticales y horizontales para formar el enrejado
waitbar(acumulado+0.09*subtotal,hh)
acumulado_local=acumulado+0.09*subtotal;
subtotal_local=0.91*subtotal;
y=polinomica_rellenar_malla(x,xa,n,metodo,hh,acumulado_local,subtotal_local);
% Realiza el relleno en malla sobre el enrejado

waitbar(acumulado+subtotal,hh)
```

Función `polinomica_vertical`

Propósito

Rellenar las verticales del enrejado utilizando algún método de interpolación polinómica.

Sintaxis

```
y = polinomica_vertical(x,n,método)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `n`: Número entero positivo que indica la cantidad de ampliación.
- `método`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.3.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene las verticales del enrejado de la región de `x` ampliada.

Funciones empleadas

- `interpolar`

Código fuente

Listado 8.19. Función `polinomica_vertical`

```
function y=polinomica_vertical(x,n,metodo,h,acumulado,subtotal)
% y=polinomica_vertical(x,n,método) rellena las verticales utilizando interpolación
% polinomial. x es una matriz de dos dimensiones.
% n es el número de píxeles por lado que conforman el píxel ampliado.
% n puede ser par o impar.
% Los valores para "método" son los siguientes:
% método = 5: interpolación lineal.
% método = 6: interpolación polinomial.
% método = 7: interpolación con c-splines.
% método = 8: interpolación con b-splines.
% método = 9: interpolación polinomial adaptativa.

y=zeros(size(x,1)*n,size(x,2)*n);
val_x=1:size(x,1);
val_xa=linspace(0,size(x,1)+1,(size(x,1)+1)*n+1);
val_xa=val_xa(floor(n/2)+2:end-ceil(n/2));
for cx=1:size(x,2)  % Para las columnas
   val_y=x(val_x,cx)';
   y(:,cx*n-floor(n/2))=interpolar(val_x,val_y,val_xa,metodo)';
   waitbar(acumulado+subtotal*cx/size(x,2),h)
end
```

Función `polinomica_horizontal`

Propósito

Rellenar las horizontales del enrejado utilizando algún método de interpolación polinómica.

Sintaxis

```
y = polinomica_horizontal(x,n,método)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `n`: Número entero positivo que indica la cantidad de ampliación.
- `método`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.3.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene las horizontales del enrejado de la región de `x` ampliada.

Funciones empleadas

- `interpolar`

Código fuente

Listado 8.20. Función `polinomica_horizontal`

```
function y=polinomica_horizontal(x,n,metodo,h,acumulado,subtotal)
% y=polinomica_horizontal(x,n,método) rellena las horizontales utilizando interpolación
% polinomial. x es una matriz de dos dimensiones.
% n es el número de píxeles por lado que conforman el píxel ampliado.
% n puede ser par o impar.
% Los valores para "método" son los siguientes:
% método = 5: interpolación lineal.
% método = 6: interpolación polinomial.
% método = 7: interpolación con c-splines.
```

```
% método = 8: interpolación con b-splines.
% método = 9: interpolación polinomial adaptativa.

y=zeros(size(x,1)*n,size(x,2)*n);
val_x=1:size(x,2);
val_xa=linspace(0,size(x,2)+1,(size(x,2)+1)*n+1);
val_xa=val_xa(floor(n/2)+2:end-ceil(n/2));
for fx=1:size(x,1)  % Para las filas
   val_y=x(fx,val_x);
   y(fx*n-floor(n/2),:)=interpolar(val_x,val_y,val_xa,metodo);
   waitbar(acumulado+subtotal*fx/size(x,1),h)
end
```

Función `polinomica_rellenar_malla`

Propósito

Rellenar los píxeles intermedios del enrejado utilizando algún método de interpolación polinómica, siguiendo una forma en malla.

Sintaxis

```
y = polinomica_rellenar_malla(x,xa,n,método)
```

Argumentos

- `x`: Matriz de doble precisión de dos dimensiones que contiene un canal de color de una región de una imagen digital en escala de grises o en RGB, respectivamente.
- `xa`: Matriz a la que se le ha aplicado `polinomica_vertical.M` y `polinomica_horizontal.M`.
- `n`: Número entero positivo que indica la cantidad de ampliación.
- `método`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.3.

Resultados

- `y`: Matriz de doble precisión de dos dimensiones que contiene los píxeles intermedios del enrejado de la región de `x` ampliada.

Funciones empleadas

♦ interpolar

Código fuente

Listado 8.21. Función `polinomica_rellenar_malla`

```
function y=polinomica_rellenar_malla(x,xa,n,metodo,h,acumulado,subtotal)
% y=polinomica_rellenar_malla(x,xa,n,método) rellena los píxeles intermedios en xa,
% promediando la interpolación de las filas y columnas. La interpolación se realiza con el
% método especificado por "método".
% x es una matriz de dos dimensiones.
% n es el número de píxeles por lado que conforman el píxel ampliado.
% n puede ser par o impar.
% Los valores para "método" son los siguientes:
% método = 5: interpolación lineal.
% método = 6: interpolación polinomial.
% método = 7: interpolación con c-splines.
% método = 8: interpolación con b-splines.
% método = 9: interpolación polinomial adaptativa.

verticales=zeros(size(x,1)*n,size(x,2)*n);
horizontales=verticales;
pos_col_xa=[1:size(x,2)]*n-floor(n/2);
val_x=1:size(x,2);
val_xa=linspace(0,size(x,2)+1,(size(x,2)+1)*n+1);
val_xa=val_xa(floor(n/2)+2:end-ceil(n/2));
val_y=zeros(size(val_x));
for fxa=1:size(xa,1)    % Para las filas
   for i=1:size(x,2)
      val_y(i)=xa(fxa,pos_col_xa(i));
   end
   horizontales(fxa,:)=interpolar(val_x,val_y,val_xa,metodo);
   waitbar(acumulado+0.5*subtotal*fxa/size(xa,1),h)
end
pos_fil_xa=[1:size(x,1)]*n-floor(n/2);
val_x=1:size(x,1);
val_xa=linspace(0,size(x,1)+1,(size(x,1)+1)*n+1);
val_xa=val_xa(floor(n/2)+2:end-ceil(n/2));
val_y=zeros(size(val_x));
for cxa=1:size(xa,2)    % Para las columnas
   for i=1:size(x,1)
      val_y(i)=xa(pos_fil_xa(i),cxa);
   end
   verticales(:,cxa)=interpolar(val_x,val_y,val_xa,metodo)';
   waitbar(acumulado+0.5*subtotal*(1+cxa/size(xa,2)),h)
end
y=(horizontales+verticales)/2;

% Para eliminar la retícula
```

```
if metodo=='b-splines'
   if n~=2
      for f=1:size(x,1)
         y(ceil(n/2)+(f-1)*n,:)=(y(ceil(n/2)+(f-1)*n-1,:)+y(ceil(n/2)+(f-1)*n+1,:))/2;
      end
      for c=1:size(x,2)
         y(:,ceil(n/2)+(c-1)*n)=(y(:,ceil(n/2)+(c-1)*n-1)+y(:,ceil(n/2)+(c-1)*n+1))/2;
      end
   end
end
```

Función `interpolar`

Propósito

Asignar valores utilizando algún método de interpolación polinómica.

Sintaxis

```
yi = interpolar(x,y,xi,método)
```

Argumentos

- `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.
- `y`: Vector fila que contiene los valores conocidos de la función o funciones polinómicas utilizadas para la interpolación.
- `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.
- `método`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.3.

Resultados

- `yi`: Vector fila que contiene los valores interpolados.

Comentarios

- Los valores de `x` y de `xi` deben estar en orden ascendente, igualmente espaciados, y deben ser todos mayores que cero.

Funciones empleadas

- ♦ lineal
- ♦ lagrange
- ♦ c_splines
- ♦ b_splines
- ♦ smart_lagrange

Código fuente

Listado 8.22. Función `interpolar`

```
function yi=interpolar(x,y,xi,metodo)
% yi=interpolar(x,y,xi,metodo) interpola con el "método" especificado para encontrar "yi",
% los cuales son los valores de la función subyacente "y" en los puntos dados por el
% vector "xi". El vector "x" especifica los puntos para los cuales se proporciona "y".
% Los valores de "x" y de "xi" deben estar en orden ascendente, igualmente espaciados, y
% deben ser todos mayores que cero.
% Los valores para "método" son los siguientes:
% método = 5: interpolación lineal.
% método = 6: interpolación polinomial.
% método = 7: interpolación con c-splines.
% método = 8: interpolación con b-splines.
% método = 9: interpolación polinomial adaptativa.

switch metodo
   case 5
      yi=lineal(x,y,xi);
   case 6
      yi=lagrange(x,y,xi);
   case 7
      yi=c_splines(x,y,xi);
   case 8
      yi=b_splines(x,y,xi);
   case 9
      yi=smart_lagrange(x,y,xi);
end
```

Función `lineal`

Propósito

Asignar valores utilizando interpolación lineal.

Sintaxis

```
yi = lineal(x,y,xi)
```

Argumentos

- `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.
- `y`: Vector fila que contiene los valores conocidos de las funciones lineales utilizadas para la interpolación.
- `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.
- `método`: Número entero que indica el tipo de método a utilizar para la ampliación. Los tipos disponibles se resumen en la Tabla 8.3.

Resultados

- `yi`: Vector fila que contiene los valores interpolados.

Comentarios

- Los valores de `x` y de `xi` deben estar en orden ascendente, igualmente espaciados, y deben ser todos mayores que cero.

Código fuente

Listado 8.23. Función `lineal`

```
function yi=lineal(x,y,xi)
% yi=lineal(x,y,xi) asigna con interpolación lineal los valores de "yi", los cuales son
% los valores de la función subyacente "y" en los puntos dados por el vector "xi". El
% vector "x" especifica los puntos para los cuales se proporciona "y".
% Los valores de "x" y de "xi" deben estar en orden ascendente, igualmente espaciados, y
% deben ser todos mayores que cero.

% Extrapolar linealmente valor de "y" a la izquierda y a la derecha.
y(2:end+1)=y;
y(1)=y(2)-(y(3)-y(2));
y(end+1)=y(end)+(y(end)-y(end-1));

xi=xi+1;    % Evitar indices iguales a cero.

x1=ceil(xi-1);
x2=ceil(xi);
y1=y(x1);
y2=y(x2);
yi=y1+(y2-y1).*(xi-x1)./(x2-x1);
```

Función `lagrange`

Propósito

Asignar valores utilizando interpolación polinomial de Lagrange.

Sintaxis

```
yi = lagrange(x,y,xi,no_fix)
```

Argumentos

- `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.
- `y`: Vector fila que contiene los valores conocidos de la función polinómica utilizada para la interpolación.
- `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.
- `no_fix`: Bandera que indica si se deben "centrar" los datos de `x` e `y` antes de interpolar. Si se pasa este argumento, independientemente de su valor, no se realizará el "centrado" de los datos.

Resultados

- `yi`: Vector fila que contiene los valores interpolados.

Comentarios

- El "centrado" de los datos se realiza repitiendo al inicio y al final de `y` un número igual de puntos como valores en haya en `y`.
- El "centrado" se realiza para aminorar el efecto de las oscilaciones en el polinomio interpolado.
- `x` y `xi` deben estar ordenadas ascendentemente.

Código fuente

Listado 8.24. Función `lagrange`

```
function fi=lagrange(x,f,xi,no_fix)
```

```
% Interpolación de datos mediante la interpolación de Lagrange.
% Sintaxis: yi=lagrange(x,y,xi,no_fix)
%   x,y -> tabla de datos en forma de arreglo.
%   xi  -> arreglo de abscisas para las cuales se calculara el valor de "y".
%   yi  -> arreglo de valores de "y" calculados mediante interpolacion de Lagrange.
% Si no se provee no_fix, se repite f(1) size(f,2) veces a la izquierda, y f(2) size(f,2)
% veces a la derecha, para aminorar el efecto de las oscilaciones.
% "x" y "xi" deben estar ordenadas ascendentemente.

% PARA EVITAR PICOS Y VALLES DEBIDO A LA OSCILACION
if nargin==3
   % Añadir datos al inicio y al final de f
   izq=ones(1,size(f,2))*f(1);
   der=ones(1,size(f,2))*f(end);
   f=[izq,f,der];
   % Agregar datos al inicio y al final de x
   delta=x(2)-x(1);
   izq=x(1)-size(x,2)*delta:delta:x(1)-delta;
   der=x(end)+delta:delta:x(end)+size(x,2)*delta;
   x=[izq,x,der];
end

fi=zeros(size(xi));
np1=length(f);
for i=1:np1
   z=ones(size(xi));
   for j=1:np1
      if i~=j, z=z.*(xi-x(j))/(x(i)-x(j)); end
   end
   fi=fi+z*f(i);
end
```

Función `c_splines`

Propósito

Asignar valores utilizando interpolación cúbica segmentaria (c-splines).

Sintaxis

```
yi = c_splines(x,y,xi)
```

Argumentos

♦ `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.

- ♦ `y`: Vector fila que contiene los valores conocidos de las funciones cúbicas segmentarias utilizadas para la interpolación.
- ♦ `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.

Resultados

- ♦ `yi`: Vector fila que contiene los valores interpolados.

Comentarios

- ♦ Los valores de `x` y de `xi` deben estar en orden ascendente, igualmente espaciados, y deben ser todos mayores que cero.
- ♦ Los puntos extremos se duplican para permitir la interpolación de valores de `xi` fuera del rango de `x`.

Código fuente

Listado 8.25. Función `c_splines`

```
function yi=c_splines(x,y,xi)
% yi=c_splines(x,y,xi) interpola con c-splines para encontrar "yi", los cuales son los
% valores de la función subyacente "y" en los puntos dados por el vector "xi". El vector
% "x" especifica los puntos para los cuales se proporciona "y". Los puntos extremos se
% duplican para permitir la interpolación del primer y último segmento de datos.
% Los valores de "x" y de "xi" deben estar en orden ascendente, igualmente espaciados, y
% deben ser todos mayores que cero.

n=size(x,2);
% Para duplicar el primero y último valores de "y".
x(2:n+1)=x;
x(1)=x(2)-(x(3)-x(2));
x(n+2)=x(n+1)+(x(n+1)-x(n));
y(2:n+1)=y;
y(1)=y(2);
y(n+2)=y(n+1);
n=n+2;
x=x+1;
xi=xi+1;

N=n-2;
% Cálculo de las hi
h=zeros(1,size(x,2)-1);
for i=1:size(x,2)-1
   h(i)=x(i+1)-x(i);
end
```

```
% Calcular las segundas derivadas
a(1)=0;
a(2:N-1)=h(2:N-1);
a(N)=h(N)-h(N+1)^2/h(N);
b(1)=3*h(1)+2*h(2)+h(1)^2/h(2);
b(2:N-1)=2*h(2:N-1)+2*h(3:N);
b(N)=2*h(N)+3*h(N+1)+h(N+1)^2/h(N);
c(1)=h(2)-h(1)^2/h(2);
c(2:N-1)=h(3:N);
d(1:N)=6*(1./h(1:N).*y(1:N)-(1./h(1:N)+1./h(2:N+1)).*y(2:N+1)+1./h(2:N+1).*y(3:N+2));
ypp=tri_diag(a,b,c,d,N);
ypp(2:N+1)=ypp;
ypp(1)=(1+h(1)/h(2))*ypp(2)-h(1)/h(2)*ypp(3);
ypp(n)=(1+h(n-1)/h(n-2))*ypp(n-1)-h(n-1)/h(n-2)*ypp(n-2);
% Calcular las funciones cúbicas segmentarias
yi=zeros(size(xi));
for c=1:size(xi,2)
   if xi(c)<=x(1)
      i=1;
   elseif xi(c)>x(n-1)
      i=n-1;
   else
      i=floor(xi(c));
   end
   t=xi(c)-x(i);
   yi(c)=y(i)+((y(i+1)-y(i))/h(i)-...
      (ypp(i+1)+2*ypp(i))/6*h(i))*t+ypp(i)/2*t^2+(ypp(i+1)-ypp(i))/6/h(i)*t^3;
end
```

Función `trid_diag`

Propósito

Resolver un sistema tridiagonal.

Sintaxis

```
f = tri_diag(a,b,c,d,n)
```

Argumentos

- `a`: Vector fila que contiene los valores de la diagonal inferior.
- `b`: Vector fila que contiene los valores de la diagonal central.
- `c`: Vector fila que contiene los valores de la diagonal superior.

- `d`: Vector fila que contiene los valores del vector columna de términos independientes.
- `n`: Número de incógnitas.

Resultados

- `f`: Vector fila que contiene los valores del vector columna de incógnitas.

Comentarios

- Un sistema tridiagonal tiene la forma de la ecuación (E.1) (ANEXO E), que se repite a continuación:

$$\begin{bmatrix} B_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & A_i & B_i & C_i & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & A_{N-1} & B_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & A_N & B_N \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_i \\ \vdots \\ \phi_{N-1} \\ \phi_N \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_i \\ \vdots \\ D_{N-1} \\ D_N \end{bmatrix}$$

En donde los valores de A_i, `Bi` y `Ci` constituyen las diagonales inferior, central y superior, respectivamente.

Código fuente

Listado 8.26. Función `tri_diag`

```
function f=tri_diag(a,b,c,d,n)
% f=tri_diag(A,B,C,D,N) resuelve un sistema tridiagonal, de la forma
%
% [ B1  C1             ]    [ phi_1 ]    [ D1 ]
% [ A2  B2  C2          ]    [ phi_2 ]    [ D2 ]
% [    A3  B3  C3       ] *  [ phi_3 ] = [ D3 ]
% [       .. .. ..     ]    [ ..... ]    [ .. ]
% [          .. .. CN-1 ]    [ ..... ]    [ .. ]
% [          AN  BN  CN ]    [ phi_N ]    [ D5 ]
%
% en donde "f" es un vector FILA que contiene los valores de phi_i.

for i=2:n
```

```
   r=a(i)/b(i-1);
   b(i)=b(i)-r*c(i-1);
   d(i)=d(i)-r*d(i-1);
end
d(n)=d(n)/b(n);
for i=n-1:-1:1
   d(i)=(d(i)-c(i)*d(i+1))/b(i);
end
f=d;
```

Función `b_splines`

Propósito

Asignar valores utilizando interpolación con b-splines.

Sintaxis

```
yi = b_splines(x,y,xi)
```

Argumentos

- `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.
- `y`: Vector fila que contiene los datos que darán origen a las funciones b-spline utilizadas para la interpolación.
- `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.

Resultados

- `yi`: Vector fila que contiene los valores interpolados.

Comentarios

- Los valores de `x` y de `xi` deben estar en orden ascendente, igualmente espaciados, y deben ser todos mayores que cero.
- Los puntos extremos se duplican para permitir la interpolación de valores de `xi` fuera del rango de `x`.

Código fuente

Listado 8.27. Función `b_splines`

```
function yi=b_splines(x,y,xi)
% yi=b_splines(x,y,xi) interpola con b-splines para encontrar "yi", los cuales son los
% valores de la función subyacente "y" en los puntos dados por el vector "xi". El vector
% "x" especifica los puntos para los cuales se proporciona "y". Los puntos extremos se
% duplican para permitir la interpolación del primer y último segmento de datos.
% Los valores de "x" y de "xi" deben estar en orden ascendente, igualmente espaciados, y
% deben ser todos mayores que cero.

n=size(x,2);
% Para duplicar el primero y último valores de "y".
x(2:n+1)=x;
x(1)=x(2)-(x(3)-x(2));
x(n+2)=x(n+1)+(x(n+1)-x(n));
y(2:n+1)=y;
y(1)=y(2);
y(n+2)=y(n+1);
n=n+2;

yi=zeros(size(xi));
for c=1:size(xi,2)
   if xi(c)<=x(2)
      i=2;
   elseif xi(c)>x(n-2)
      i=n-2;
   else
      i=find(xi(c)<=x(1:end));
      i=i(1)-1;
   end
   t=xi(c)-x(i);
   yi(c)=1/6*((1-t)^3*y(i-1)+(3*t^3-6*t^2+4)*y(i)+(-3*t^3+3*t^2+3*t+1)*y(i+1)+t^3*y(i+2));
end
```

Función `smart_lagrange`

Propósito

Asignar valores utilizando interpolación polinomial de Lagrange adaptativa.

Sintaxis

```
yi = smart_lagrange(x,y,xi)
```

Argumentos

- `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.
- `y`: Vector fila que contiene los valores conocidos de las funciones polinómicas utilizadas para la interpolación.
- `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.

Resultados

- `yi`: Vector fila que contiene los valores interpolados.

Comentarios

- La interpolación de lagrange adaptativa asigna múltiples polinomios a una serie de datos, de manera que en cualquier punto el valor interpolado no sobrepase los valores de datos dados más cercanos a la izquierda y a la derecha del punto en cuestión en una cantidad mayor a "delta".
- "delta" es constante e igual a 1.
- `x` y `xi` deben estar ordenadas ascendentemente.

Funciones empleadas

- `lagrange2`
- `revisar_delta`

Código fuente

Listado 8.28. Función `smart_lagrange`

```
function yi=smart_lagrange(x,y,xi)
% Interpolación de datos mediante la interpolación de Lagrange adaptativa.
% Sintaxis: yi=smart_lagrange(x,y,xi)
%   x,y -> tabla de datos en forma de arreglo.
%   xi  -> arreglo de abscisas para las cuales se calculara el valor de "y".
%   yi  -> arreglo de valores de "y" calculados mediante interpolación de Lagrange
%   adaptativa.
% La interpolación de lagrange adaptativa asigna múltiples polinomios a una serie de
% datos, de manera que en cualquier punto el valor interpolado no sobrepase los valores de
% datos dados más cercanos a la izquierda y a la derecha del punto en cuestión en una
```

```
% cantidad mayor a "delta".
% El algoritmo esta optimizado para valores contiguos repetidos de "y".
% "x" y "xi" deben estar ordenados ascendentemente.
% delta = 1.

delta=1;
no_completado=1;
yi=[];
yi_t_ant=[];
pos_ini=1;
pos_fin=2;
posiciones_xi=[1];
for i=2:size(x,2)-1
   pos=find(x(i)<=xi);
   posiciones_xi=[posiciones_xi,pos(1)];
end
posiciones_xi=[posiciones_xi,size(xi,2)];
while pos_fin<size(y,2)
   last_equal=find([diff(y(pos_ini:end)),1]);
   if last_equal(1)>2
      pos_fin=pos_ini+last_equal(1)-1;
      yi_t=lagrange2(x(pos_ini:pos_fin),y(pos_ini:pos_fin),...
         xi(posiciones_xi(pos_ini):posiciones_xi(pos_fin)));
      pos_ini=pos_fin;
      if pos_fin==size(y,2)
         yi=[yi,yi_t];
         no_completado=0;
         break
      else
         yi=[yi,yi_t(1:end-1)];
      end
   else
      x_t=x(pos_ini:pos_fin+1);
      y_t=y(pos_ini:pos_fin+1);
      xi_t=xi(posiciones_xi(pos_ini):posiciones_xi(pos_fin+1));
      yi_t=lagrange(x_t,y_t,xi_t,1);
      aumentar_orden=revisar_delta(x_t,y_t,xi_t,yi_t,delta);
      if aumentar_orden==0
         if pos_fin-pos_ini~=1
            yi=[yi,yi_t_ant(1:end-1)];
         else
            yi_t=lagrange2(x(pos_ini:pos_fin),y(pos_ini:pos_fin),...
               xi(posiciones_xi(pos_ini):posiciones_xi(pos_fin)));
            if pos_fin==size(y,2)-1
               yi=[yi,yi_t];
            else
               yi=[yi,yi_t(1:end-1)];
            end
         end
         pos_ini=pos_fin;
         no_completado=1;
      else
```

```
        yi_t_ant=yi_t;
      end
   end
   pos_fin=pos_fin+1;
end
if no_completado
   yi=[yi,lagrange2(x(pos_ini:pos_fin),y(pos_ini:pos_fin),...
         xi(posiciones_xi(pos_ini):posiciones_xi(pos_fin)))];
end
yi=yi(1:size(xi,2));
```

Función `lagrange2`

Propósito

Asignar valores utilizando interpolación polinomial de Lagrange.

Sintaxis

```
yi = lagrange2(x,y,xi)
```

Argumentos

- `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.
- `y`: Vector fila que contiene los valores conocidos de la función polinómica utilizada para la interpolación.
- `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.

Resultados

- `yi`: Vector fila que contiene los valores interpolados.

Comentarios

- La diferencia entre `lagrange` y `lagrange2` es que la primera permite el "centrado" de los datos, mientras que la segunda no realiza dicho proceso. Además, el algoritmo de `lagrange2` está optimizado para cuando todos los valores de `y` son iguales.
- `x` y `xi` deben estar ordenadas ascendentemente.

Código fuente

Listado 8.29. Función `lagrange2`

```
function fi=lagrange2(x,f,xi)
% Interpolación de datos mediante la interpolación de Lagrange.
% Sintaxis: yi=lagrange2(x,y,xi)
%   x,y -> tabla de datos en forma de arreglo.
%   xi  -> arreglo de abscisas para las cuales se calculara el valor de "y".
%   yi  -> arreglo de valores de "y" calculados mediante interpolación de Lagrange.
% El algoritmo esta optimizado para cuando todos los valores de  "y" son iguales.
% "x" y "xi" deben estar ordenadas ascendentemente.

if not(todos_iguales(f))
   fi=zeros(size(xi));
   np1=length(f);
   for i=1:np1
      z=ones(size(xi));
      for j=1:np1
         if i~=j, z=z.*(xi-x(j))/(x(i)-x(j)); end
      end
      fi=fi+z*f(i);
   end
else
   fi=ones(1,size(xi,2))*f(1);
end

function son_iguales=todos_iguales(M)
% Devuelve 1 si todos los elementos de un vector fila son iguales.
% Caso contrario, devuelve 0.
son_iguales=1;
for i=1:size(M,2)-1
   if M(i)~=M(i+1)
      son_iguales=0;
      return
   end
end
```

Función `revisar_delta`

Propósito

Determinar si un conjunto de datos dados sobrepasan el rango de aceptación permitido para la interpolación polinomial adaptativa.

Sintaxis

```
ok = revisar_delta(x,y,xi,yi,delta)
```

Argumentos

- `x`: Vector fila que especifica los puntos para los cuales se proporciona `y`.
- `y`: Vector fila que contiene los valores conocidos de la función polinómica utilizada para la interpolación.
- `xi`: Vector fila que especifica los puntos para los que se desea interpolar valores.
- `yi`: Vector fila que contiene los valores interpolados.
- `delta`: Valor máximo en el que un valor de `yi` puede alejarse de los valores de `y` más próximos a la izquierda y a la derecha.

Resultados

- `ok`: Estatus de la verificación. `ok` = 1 si para todas las posiciones de `xi` los valores de `yi` no sobrepasan en una cantidad delta a los valores de `y` para las posiciones de x más cercanas a la izquierda y a la derecha de la posición en `xi`. En otro caso, `ok` = 0.

Código fuente

Listado 8.30. Función `revisar_delta`

```
function ok=revisar_delta(x,y,xi,yi,delta)
% ok=revisar_delta(x,y,xi,yi,delta) devuelve ok = 1 si para todas las posiciones de xi los
% valores de yi no sobrepasan en una cantidad delta a los valores de y para las posiciones
% de x más cercanas a la izquierda y a la derecha de la posición en xi.
% En otro caso, devuelve ok = 0.

ok=1;
% Encontrar posiciones en xi
posiciones=[];
for i=1:size(x,2)
   pos=find(x(i)<=xi);
   posiciones=[posiciones,pos(1)];
end

for i=1:size(x,2)-1
   if (yi(posiciones(i):posiciones(i+1))<=max(y(i),y(i+1))+delta) &...
         (yi(posiciones(i):posiciones(i+1))>=min(y(i),y(i+1))-delta)
   else
      ok=0;
      break
   end
```

```
end
```

ANEXO C. BIBLIOTECA ENLAZADA DINÁMICAMENTE[27]

En este apéndice se describe el procedimiento seguido por los autores del Trabajo de Graduación para la construcción de la biblioteca enlazada dinámicamente (dinamically linked library o DLL) de las funciones en archivos M creadas en MATLAB descritas en el ANEXO B, utilizando para ello la caja de herramientas MATLAB COM Builder versión 1.0 en MATLAB versión 6.5 distribución 13, de The MathWorks Inc. También se describe brevemente el uso de la biblioteca en un proyecto de Visual Basic. Para mayor información, consúltese [10] u [11].

C.1. Elementos de un Proyecto de COM Builder

Un proyecto consiste de todos los elementos necesarios para construir una aplicación portátil utilizando MATLAB COM Builder. Los componentes del COM Builder son objetos COM accesibles a través de Visual Basic, C++, o cualquier otro lenguaje que soporte COM. COM es un acrónimo de *Component Object Model*, o Modelo del Objeto Componente, el cual constituye un estándar binario de Microsoft para la interoperabilidad entre objetos. Cada objeto COM provee una o más clases al entorno de programación de Visual Basic. Cada clase contiene una serie de funciones llamadas *métodos*, correspondientes a las funciones originales de MATLAB incluidas en el proyecto del componente.

- *Clases*. Cuando se crea un componente, se debe proveer adicionalmente uno o más nombres de clases. El nombre del componente representa el nombre del archivo DLL a ser creado. Un nombre de clase denota el nombre de la clase que realiza una llamada a un método específico en tiempo de ejecución. La relación entre el nombre de un componente y el nombre de una clase, y cuáles métodos

[27] Adaptado de *MATLAB COM Builder: Building a Deployable Application* en [10].

(funciones de MATLAB) intervienen en una clase particular, son puramente organizacionales. Como regla general, cuando se compilan muchas funciones de MATLAB, es útil determinar un esquema de categorías de funciones y crear una clase separada para cada categoría. El nombre de cada clase debe ser descriptivo acerca del propósito de la clase.

♦ *Versiones*. Los componentes de MATLAB COM Builder también soportan un mecanismo simple de manejo de versiones. Un número de versión se añade a un componente dado. Este número se agrega automáticamente en el nombre del archivo DLL y en el Registro de Información del Sistema. Como regla general, la primera versión de un componente es 1.0 (el valor por defecto si no se escoge ninguno). Los cambios hechos al componente antes de su realización conservan el mismo número de versión. Después de su construcción, se debe cambiar el número de versión para todos los cambios subsecuentes, de manera que se puedan manejar con facilidad las versiones nuevas y anteriores. El sistema ve como distintas a las clases en diferentes versiones del mismo componente, aún si éstas tienen el mismo nombre.

C.2. Creación del DLL

Los pasos seguidos para la creación del DLL en MATLAB son los siguientes:

1. Configurar el compilador de C/C++ a ser utilizado por MATLAB COM Builder ejecutando la instrucción `mbuild -setup` desde la línea de comandos de MATLAB. El sistema pregunta al usuario si desea que `mbuild` localice los compiladores instalados actualmente en la computadora, a lo que debe responderse "`y`" (sí). Posteriormente se muestra en pantalla el listado de compiladores disponibles. MATLAB está provisto de un compilador llamado `Lcc`; sin embargo, los únicos compiladores que soportan la creación de objetos COM son Borland C++ Builder (versiones 3.0, 4.0, 5.0, y 6.0) y Microsoft Visual C/C++ (versiones 5.0, 6.0, y 7.0), siendo necesario contar con alguno de dichos

programas para la creación del DLL. Finalmente, el sistema pide confirmar al usuario su selección, y procede a la configuración del compilador.

2. Invocar la interfaz gráfica de usuario de MATLAB COM Builder, ya sea por medio del teclado desde la línea de comandos de MATLAB digitando la instrucción `comtool`, o desde el menú inicio de MATLAB o desde el "Launch Pad", seleccionando la aplicación MATLAB COM Builder. Aparecerá la ventana mostrada en la Figura 8.2.

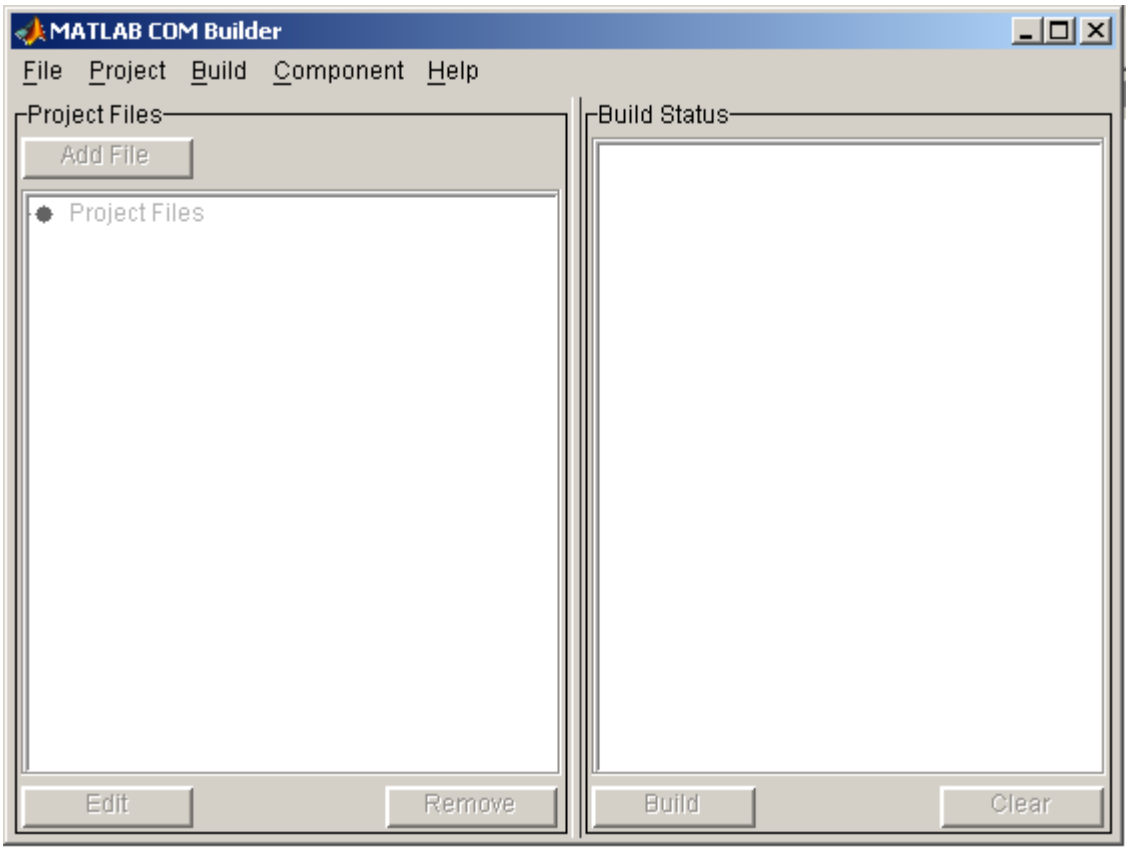


Figura 8.2. Interfaz Gráfica de Usuario de MATLAB COM Builder 1.0.

3. Crear un nuevo proyecto seleccionando `New Project` del menú `File`. Aparecerá la ventana de propiedades del nuevo proyecto (`New Project Settings`) mostrada en la Figura 8.3.

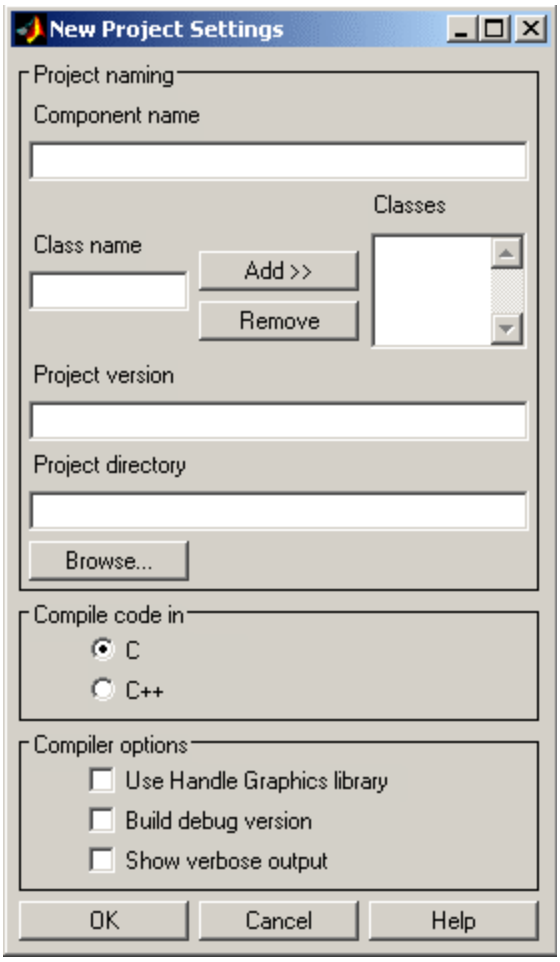


Figura 8.3. Cuadro de diálogo de configuración de las propiedades del proyecto.

La información requerida en este cuadro de diálogo puede clasificarse en tres categorías:

- Propiedades relacionadas con la identificación del proyecto
- Tipo de compilador utilizado
- Opciones de compilación

Estas categorías se describen a continuación.

♦ *Identificación del proyecto*. Se compone de la siguiente información:

- `Component name`: Nombre del componente. Denota el nombre del DLL creado más adelante en el proceso. Después de que se ingresa el nombre del componente, automáticamente se ingresa un nombre de clase idéntico al del

componente. Se puede cambiar este nombre por algo más descriptivo. A pesar de que el nombre del componente y de la clase pueden ser iguales, el nombre del componente no puede ser idéntico al de ninguna función de archivo M añadidas al proyecto.

- `Class name`: Nombre de la clase. Para añadir una clase al componente, debe ingresarse el nombre de la clase en el campo `Class name` y hacer clic en `Add>>`. La clase añadida aparece ahora en la lista `Classes`.
- `Project version`: Versión del proyecto. El valor por omisión es 1.0.
- `Project directory`: Directorio del proyecto. Especifica dónde se guardarán los archivos de proyecto y de construcción cuando se compilan y empaquetan los proyectos. El nombre del proyecto se genera automáticamente a partir del nombre del directorio actual y del nombre del componente.

NOTA: El usuario puede aceptar la ruta de proyecto generada automáticamente o escoger otra de su predilección. Una vez que se haga clic en el botón `OK`, se guarda esta ruta. Si más tarde se decide mover el proyecto o cambiar algo en su ruta, es necesario rehacer el proceso entero de especificación del proyecto, incluyendo la adición de archivos, y reespecificar la ruta del directorio del proyecto.

♦ *Tipo de compilador*. MATLAB COM Builder soporta dos tipos de compiladores para la creación del DLL: C y C++. Los componentes escritos en C proporcionan un mejor desempeño, mientras que los escritos en C++ son más legibles, permitiendo que la modificación del código generado sea más fácil de ser necesario. Los archivos generados relacionados con la interface del objeto COM están siempre en C++, independientemente de cuál opción se escoja.

♦ *Opciones de compilación*.

- *Use Handle Graphics library*: Usar biblioteca para manejo de gráficos. Si las funciones a compilar contienen llamadas a objetos gráficos generados por MATLAB, debe incluirse la biblioteca de gráficos de MATLAB de C/C++ (MATLAB C/C++ Graphics library) seleccionando esta opción.

- *Build debug version*: También se puede crear una versión depurable de las funciones compiladas, la cual permite lo siguiente:
    - Rastrear un error, puesto que cualquier error reportado muestra el nombre del archivo M y el número de línea donde ocurrió el error. Sin esta característica, los errores se reportan sin ninguna indicación de su ubicación en el código de MATLAB.
    - Permite una depuración completa utilizando el depurador de Visual Studio (Visual Studio debugger).

4. Completar la información siguiente en el cuadro de diálogo `New Project Settings`:
    - `Component name`: Zoom
    - `Class name`: Zoom
    - `Project version`: 1.0
    - `Compile code in`: C
    - `Use Handle Graphics library`: seleccionado

Una vez que se confirman estas opciones haciendo clic en el botón `OK`, éstas se vuelven parte del espacio de trabajo del proyecto y se guardan en el archivo de proyecto, junto con los nombres de todos los archivos M que se añadan subsecuentemente al mismo. Entonces el archivo de proyecto se guarda automáticamente al directorio del proyecto, con el nombre `<nombre del componente>.cbl`.

5. Luego de creado el proyecto, deben añadirse los archivos M que conformarán las funciones disponibles en el DLL, haciendo clic en el botón `Add File` o seleccionando `Add File` en el menú `Project`. Se puede agregar solamente un archivo M a la vez. En este punto, deben agregarse los archivos `JK_zoom.m` y `jk_zoom_salir`, descritos en el ANEXO B.

NOTA: El nombre de los archivos añadidos al proyecto no puede ser igual al nombre de ninguna de las funciones existentes en la biblioteca de funciones precompiladas[28].

NOTA: Las funciones de archivos M añadidas al proyecto estarán disponibles para su utilización en Visual Basic por medio del DLL que se creará. No es necesario añadir las funciones invocadas por las funciones añadidas, porque MATLAB COM Builder las compilará automáticamente con el DLL[29]. Sin embargo, dichas funciones no estarán disponibles al usuario si no se agregan manualmente al proyecto.

6. Luego de haber configurado las propiedades del proyecto y añadido las funciones de archivo M deseadas, es el momento de construir el DLL, seleccionando `COM Object` del menú `Build` o haciendo clic en el botón `Build`, lo que invoca al MATLAB Compiler, escribiendo los archivos fuente intermedios a `<directorio del proyecto>\src\` y los archivos de salida necesarios para la distribución a `<directorio del proyecto>\distrib\`, en donde se localizará el archivo `<nombre del componente_versión>.dll`.

C.3. Empaquetado y distribución del DLL

Una vez que se han compilado satisfactoriamente las funciones incluidas en el DLL, y se ha probado el objeto COM. desde Visual Basic, por ejemplo, se está listo para empaquetar el componente para su distribución a los usuarios finales.

Para ello, selecciónese `Package Component` del menú `Component` de MATLAB COM Builder para crear el ejecutable auto-extraíble conteniendo los siguientes archivos:

---

[28] Dicha biblioteca contiene las funciones propietarias de MATLAB necesarias para ejecutar los procesos matemáticos o gráficos.

[29] Siempre y cuando la invocación de dichas funciones se realice en forma explícita.

Tabla 8.4. Archivos que componen el instalador de la biblioteca

Archivo	Propósito
_install.bat	Guión seguido por el ejecutable auto-extraíble
<nombre del componente_versión>.dll	Componente compilado
mglinstaller.exe	Instalador de la biblioteca matemática y de gráficos de MATLAB
mwcomutil.dll	Biblioteca de utilidades de COM Builder
mwregsvr.exe	Ejecutable que registra los DLL en las computadoras destino

El ejecutable auto-extraíble tiene por nombre <nombre del componente>.exe.

Al correr el instalador en la computadora destino se ejecutan los siguientes pasos:

1. mglinstaller instalará la biblioteca matemática y de gráficos de MATLAB (MATLAB C/C++ y MATLAB Graphics).

NOTA: Al finalizar todo el proceso de instalación, el usuario deberá agregar a la ruta de su aplicación el directorio <aplicación>\bin\win32\ que creó el programa mglinstaller. (<aplicación> representa el directorio raíz de la aplicación desarrollada, el lugar donde la aplicación reside en el sistema).

2. mwregsvr registrará las bibliotecas mwcomutil.dll y <nombre del componente_versión>.dll.

Estos pasos deben repetirse en cada computadora donde desee instalarse el DLL creado.

C.4. Uso del DLL en Visual Basic

Los pasos a seguir para poder utilizar una biblioteca creada con MATLAB COM Builder en un proyecto de Visual Basic se describen a continuación.

1. Instalación del componente

En primer lugar, debe instalarse la biblioteca en el sistema operativo. Esto se realiza ejecutando el instalador creado en la sección anterior (anexo C.3) y siguiendo las instrucciones de instalación. Si el DLL se creó con MATLAB COM Builder en la misma computadora en donde se pretende usar la biblioteca, no es necesario instalarlo, puesto que el proceso de creación descrito en el anexo C.2 lo instala automáticamente al finalizar.

2. Adición de la biblioteca

Una vez instalado el DLL, debe agregársele al proyecto de Visual Basic en el que será utilizado. Para ello, es necesario incluir la referencia a la biblioteca en tiempo de diseño, seleccionando `Referencias` del menú `Proyecto`. Aparecerá entonces el cuadro de diálogo `Referencias`, en el que debe chequearse el nombre del componente y hacer clic en `Aceptar`. El nombre de los componentes creados con MATLAB COM Builder tienen la forma "`<Nombre del DLL> <versión> Type Library`". En el caso de este Trabajo de Graduación, el nombre del componente creado es "`Zoom 1.0 Type Library`".

3. Declaración del objeto

El siguiente paso consiste en la declaración de la variable de Visual Basic que contendrá al objeto proveniente de la clase de la biblioteca a utilizar. Debe declararse un objeto para cada clase empleada. La sintaxis para la declaración es la siguiente[30]:

{Dim | ReDim | Static | Private | Public} *variable* As [New] *<Nombre del DLL>.clase*

[30] Para mayor información acerca de la sintaxis de declaración de variables de objeto en Visual Basic, véase [12].

Aquí, "*variable*" indica el nombre de la variable empleada para referirse a la clase "*clase*" de la biblioteca dentro del proyecto en Visual Basic. El nombre de la variable lo decide el usuario, y puede ser cualquier nombre válido de Visual Basic. Por otro lado, el nombre de la clase no es antojadizo, sino que se define en el proceso de creación del DLL.

En nuestro caso, la biblioteca `Zoom.dll` se compone solamente de una clase utilizable –la clase "`Zoom`"; la variable utilizada es "`Zoom`", y la declaración es como sigue:

```
Private Zoom As Zoom.Zoom
```

4. Creación del objeto

El siguiente paso consiste en la creación de una nueva instancia de la clase a utilizar, por medio de la instrucción `Set` de Visual Basic, tal como se indica en la siguiente sintaxis:

```
Set variable = New <Nombre del DLL>.<Nombre de la clase>
```

en donde *variable* indica el nombre de la variable objeto declarada en el paso anterior. Para nuestro caso,

```
Set Zoom = New Zoom.Zoom
```

Esta asignación debe ser hecha antes de intentar utilizar la biblioteca; por ejemplo, podría colocarse dentro del procedimiento `Form_Load`.

5. Uso del objeto

Ahora ya se está en condiciones de utilizar la biblioteca. La sintaxis para el uso en Visual Basic de funciones compiladas dentro de archivos DLL creados con MATLAB COM Builder es la que se muestra a continuación.

```
Call variable.<Nombre de la función>(<Número de argumentos de
salida>, salida_1, ..., salida_n, argumento_1, ..., argumento_n)
```

en donde *variable* representa nuevamente el nombre elegido en Visual Basic para contener a la clase del objeto comprendido en la biblioteca. <Nombre de la función> se refiere a la función creada en MATLAB compilada dentro del DLL. Dicho nombre debe ser idéntico al del archivo M correspondiente que constituye la biblioteca. Como puede observarse en la sintaxis anterior, debe especificarse el número de argumentos de salida, es decir, la cantidad de variables que contendrán los resultados devueltos por la función. También deben especificarse los nombres de dichas variables de salida (salida_1 hasta salida_n, siendo n el número de argumentos de salida especificado). Finalmente, deben incluirse los argumentos de la función (argumento_1 hasta argumento_n) en caso de que la función lo requiera. Por ejemplo, para nuestro caso, la sintaxis sería la siguiente:

```
Call Zoom.JK_zoom(1, tiempo,argumento_1, ..., argumento_n)
```

En este caso, la función JK_zoom sólo devuelve un resultado, el cual se almacenará en la variable tiempo en Visual Basic. Los argumentos para esta función se describen en el anexo B.1.

Las funciones disponibles en la biblioteca aparecen automáticamente en el editor de código de Visual Basic al teclear el nombre del DLL seguido de un punto (tal como sucede con las propiedades de un objeto en dicho lenguaje de programación). En la lista de funciones que se muestra automáticamente se encuentran todas las funciones que fueron agregadas al proyecto de MATLAB COM Builder en el proceso de creación del DLL (sección C.2 de este anexo), además de todas las variables globales que se hayan declarado en los archivos M compilados, y una variable especial llamada MWFlags, empleada por la

biblioteca `MWComUtil` de MATLAB para el correcto funcionamiento del DLL[31].

[31] La biblioteca `MWComUtil` de MATLAB se instala automáticamente al instalar una biblioteca creada con MATLAB COM Builder.

ANEXO D. APLICACIÓN EN VISUAL BASIC

Para observar los resultados obtenidos a través de los procesos matemáticos realizados por MATLAB por medio del DLL (ANEXO C) al aplicar los diversos métodos de ampliación estudiados, se desarrolló una aplicación de entorno gráfico, con un ambiente amigable, sencillo y fácil de utilizar.

Este anexo será enfocado a explicar cómo debe utilizar la aplicación el usuario, así como también a analizar de cada una de las funciones utilizadas en Visual Basic, con miras a que dicha aplicación sea empleada y mejorada por trabajos posteriores, partiendo de las bases que aquí se presentan.

D.1. Manual del usuario

A continuación se detalla la forma en la que debemos interactuar con la aplicación, desde el proceso de instalación hasta el momento de cargar una imagen y aplicarle los diversos métodos para obtener la ampliación de la misma.

D.1.1. Proceso de instalación

Para instalar la aplicación en una máquina cliente, que no necesariamente cuenta con MATLAB o Visual Basic instalados en el sistema, se proporciona una archivo de nombre `ZoomVB.msi`, el cual se encarga de desempaquetar todos los archivos necesarios para poder utilizar la aplicación. Si la aplicación ya ha sido instalada anteriormente, el instalador detectará los recursos que han sido agregados y nos preguntará si deseamos desinstalar o arreglar algún archivo defectuoso. El programa de instalación es un asistente que nos llevará paso a paso por todo el proceso que se describe a continuación:

1. Buscamos en el CD de instalación el archivo `ZoomVb.msi` y hacemos doble clic sobre él; esto nos desplegará una ventana de bienvenida.

Figura 8.4. Ventana de bienvenida del instalador de la aplicación.

2. En cada una de las pantallas del asistente se tiene la opción de cancelar la instalación, con lo que se desinstala todo lo que hasta ese momento se haya copiado a la computadora. Para seguir con el proceso se hará clic en el botón "`Next`", lo que desplegará una pantalla en la que se podrá elegir en qué directorio se desea instalar la aplicación. La ruta por omisión es "`C:\Archivos de programa\ZoomVB\`". Si no deseamos emplear esta dirección, podemos cambiarla utilizando el botón "`Browse`" para buscar así la que deseemos. Al final, en la ruta que haya sido especificada se copiarán todos los archivos que el programa necesita para ser ejecutado.

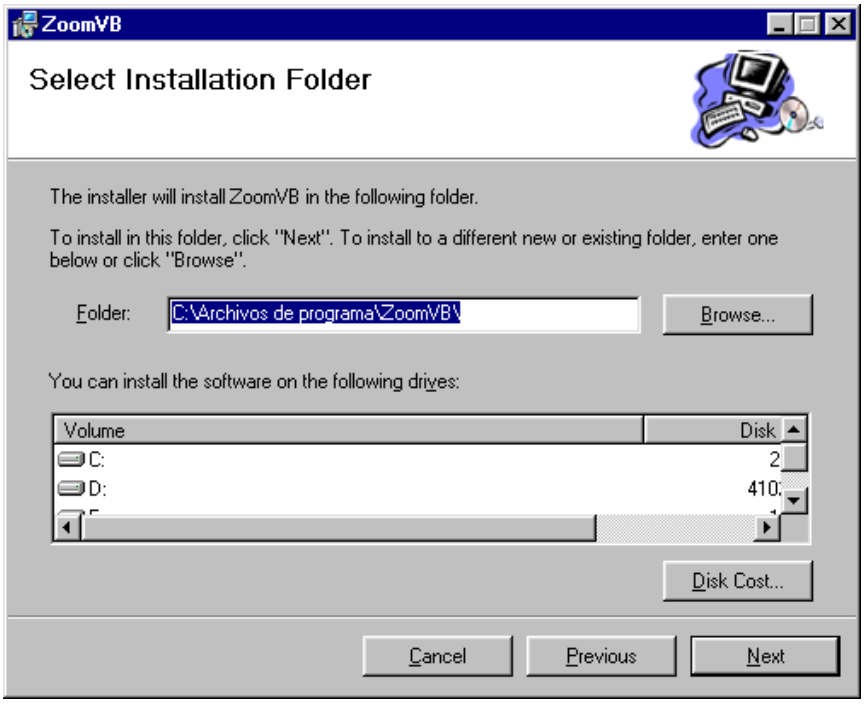


Figura 8.5. Ventana de selección de carpeta de instalación.

3. Una vez seleccionada la ruta, hacemos clic en el botón "`Next`", el cual nos envía a una ventana de confirmación. Para confirmar haremos clic en el botón "`Next`".

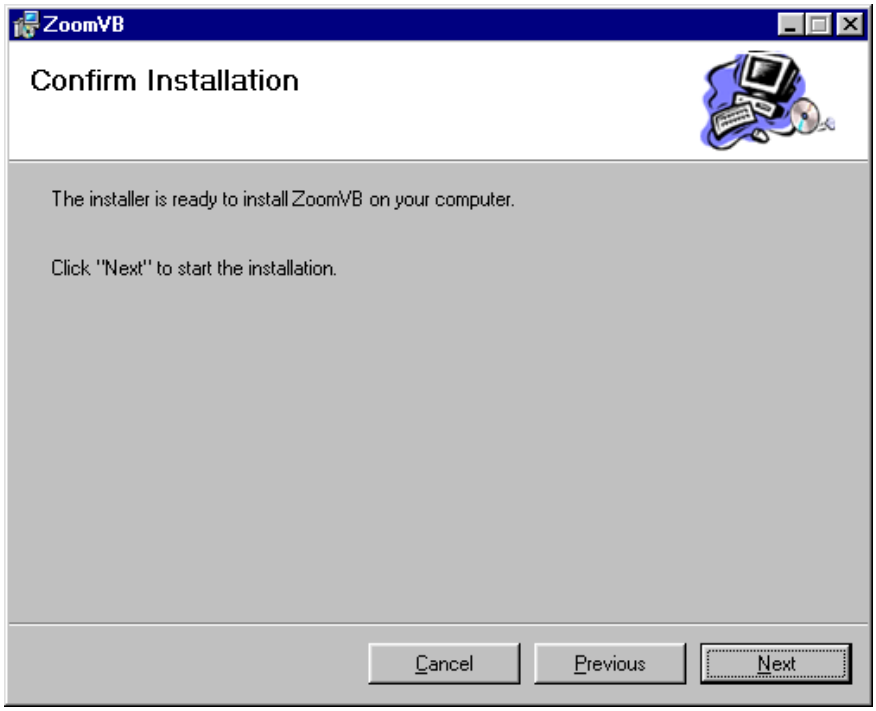


Figura 8.6. Ventana de confirmación de la instalación.

4. Luego de la confirmación, se procede a copiar todos los archivos necesarios para poder ejecutar el programa. Si en todo el proceso no existió ningún problema, al final se mostrará una ventana indicando que la aplicación fue instalada satisfactoriamente. Posteriormente hacemos clic en "`Close`" para terminar con el proceso.

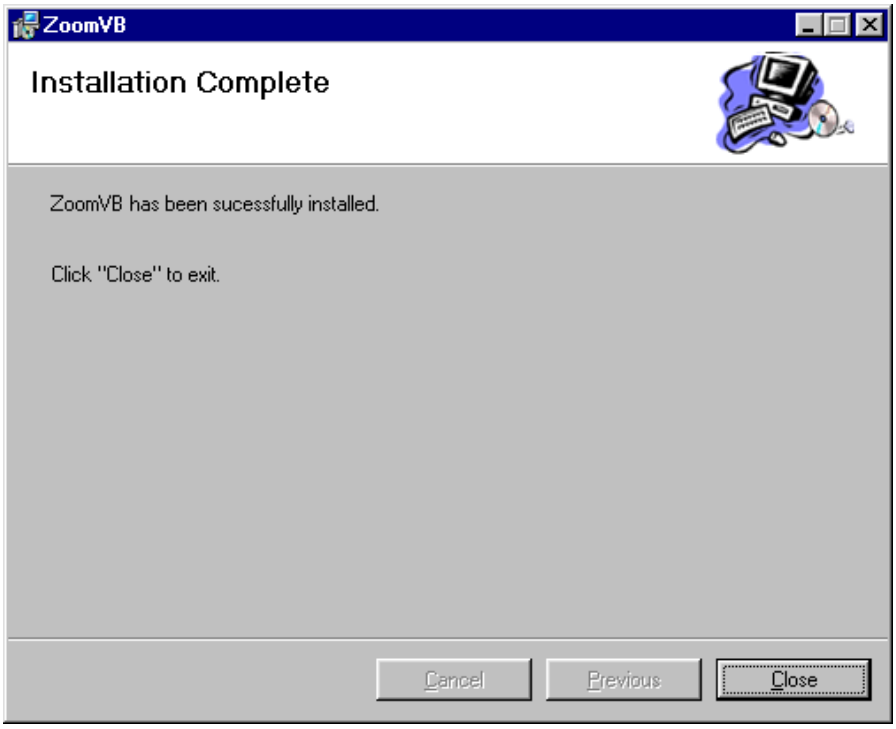


Figura 8.7. Ventana de finalización de la instalación.

5. Una vez copiados los archivos se procede a registrar las bibliotecas (DLL), necesarias para interactuar con MATLAB. Para hacer esto abrimos la ventana de comandos de DOS y nos colocamos en la ruta que se eligió para instalar la aplicación. En este ejemplo se utilizará la ruta por defecto.

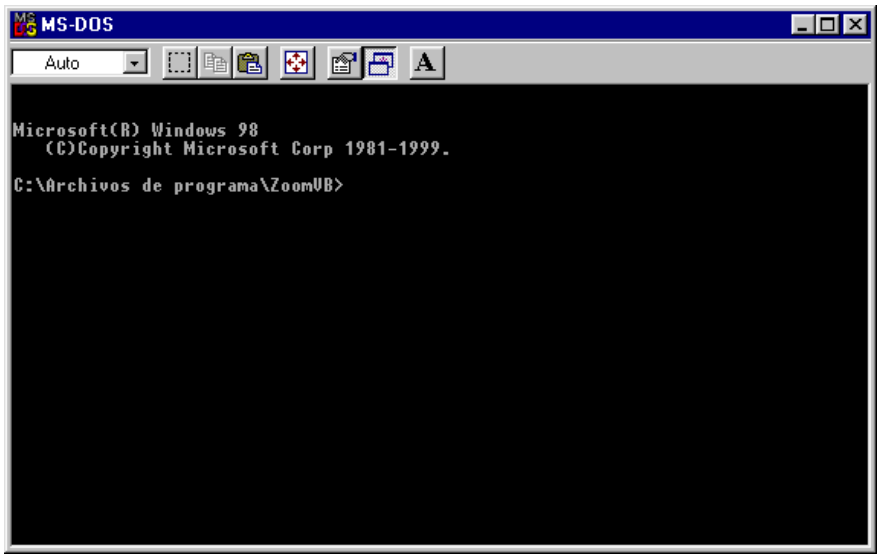


Figura 8.8. Ventana de comandos de DOS.

6. Estando posicionados en esa ruta procederemos a registrar las bibliotecas siguientes: `mwcomutil.dll` y `zoom_1_0.dll`. Utilizaremos el comando `mwregsvr`[32] y como argumento el nombre del archivo a registrar, tal como sigue:

```
mwregsvr mwcomutil.dll
mwregsvr zoom_1_0.dll
```

Al hacerlo se nos desplegará un mensaje informando que el archivo DLL fue registrado satisfactoriamente.

[32] El comando `mwregsvr` fue desarrollado por The MathWorks Inc. y viene incluido en el paquete de instalación de los DLL creados con MATLAB COM Builder.

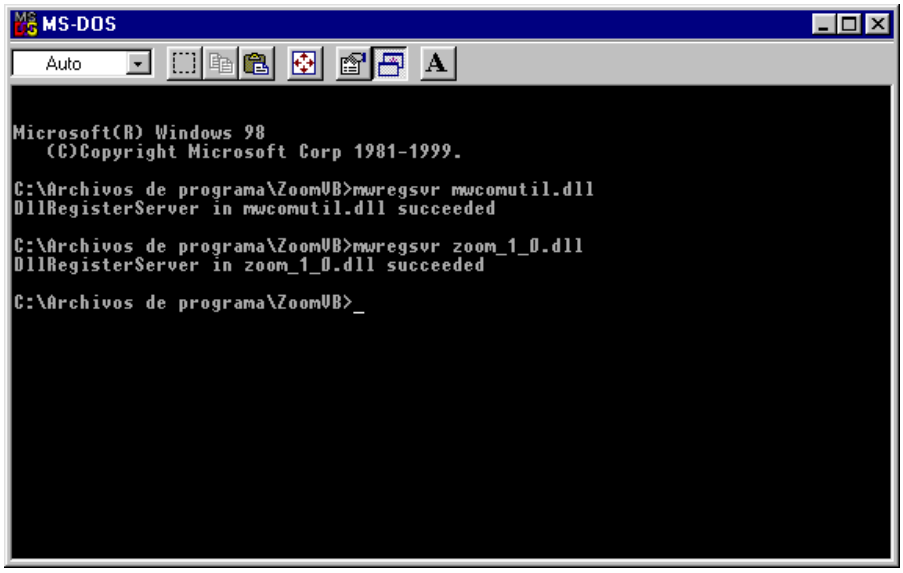


Figura 8.9. Mensaje de instalación de bibliotecas finalizada.

Luego de realizados todos estos pasos, la aplicación está lista para ser utilizada. Podemos ejecutarla a través del Acceso Directo que presenta el icono de una lupa en el menú Inicio.

D.1.2. Proceso de desinstalación

Para desinstalar de la computadora a la aplicación, se utiliza la herramienta del Panel de Control `Agregar o quitar programas`, con la que buscamos la aplicación en cuestión (`ZoomVB`) y procedemos a desinstalarla. Se nos presenta una ventana en donde se nos pregunta si deseamos borrar toda la aplicación o si queremos reparar algún archivo, en caso de que se haya creado un nuevo instalador con referencias a archivos más recientes.

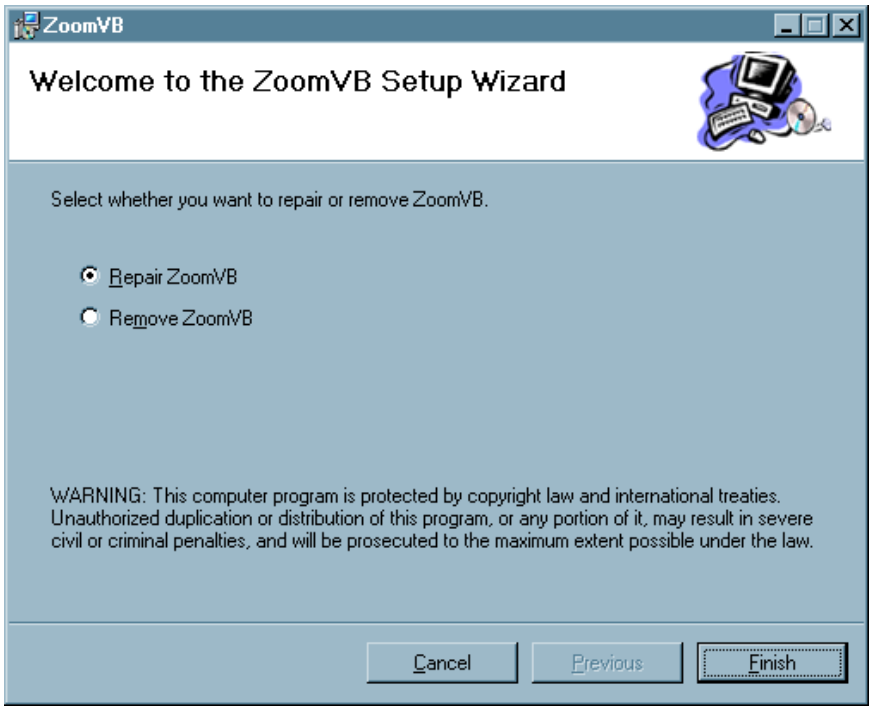


Figura 8.10. Asistente para la desinstalación de la aplicación.

Seleccionamos la opción deseada y presionamos el botón "`Finish`", con lo que se procede a realizar la acción indicada.

D.1.3. Forma de empleo de la aplicación

Una vez realizados los pasos para instalar la aplicación, podemos comenzar a utilizarla. Dos formas de acceder a ella son desde el menú Inicio a través del Acceso Directo "`ZoomVB.exe`" o dirigiéndose a la carpeta "`C:\Archivos de programa\ZoomVB`" y ejecutando el archivo "`ZoomVB.exe`". Luego de ello se carga la aplicación (Figura 8.11), cuyos controles se describen a continuación.

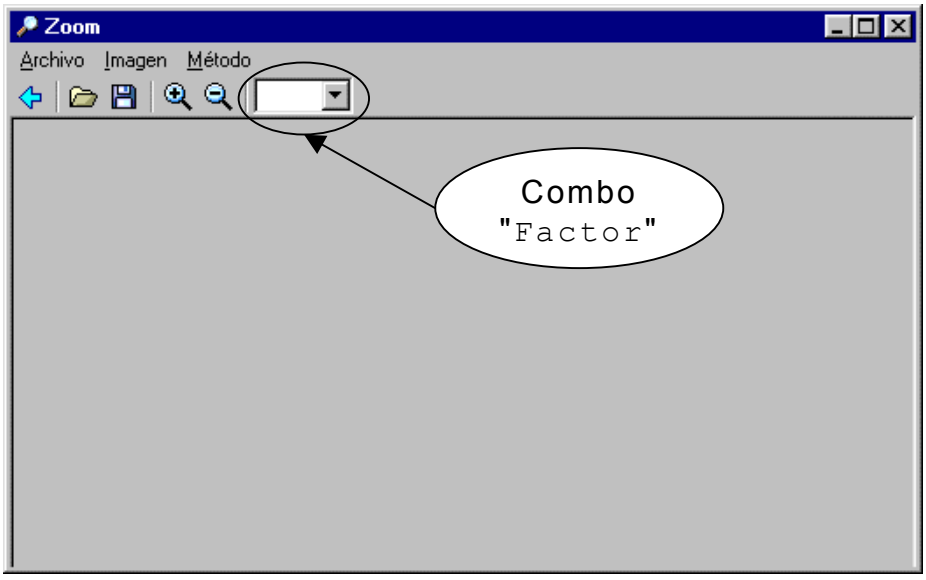


Figura 8.11. Interfaz gráfica de usuario de la aplicación.

Menú Archivo

Figura 8.12. Menú `Archivo` de la aplicación.

♦ *Abrir*. Presenta un cuadro de diálogo con el que se puede escoger la imagen que se desea ampliar, la cual puede ser de formato bmp o jpg.

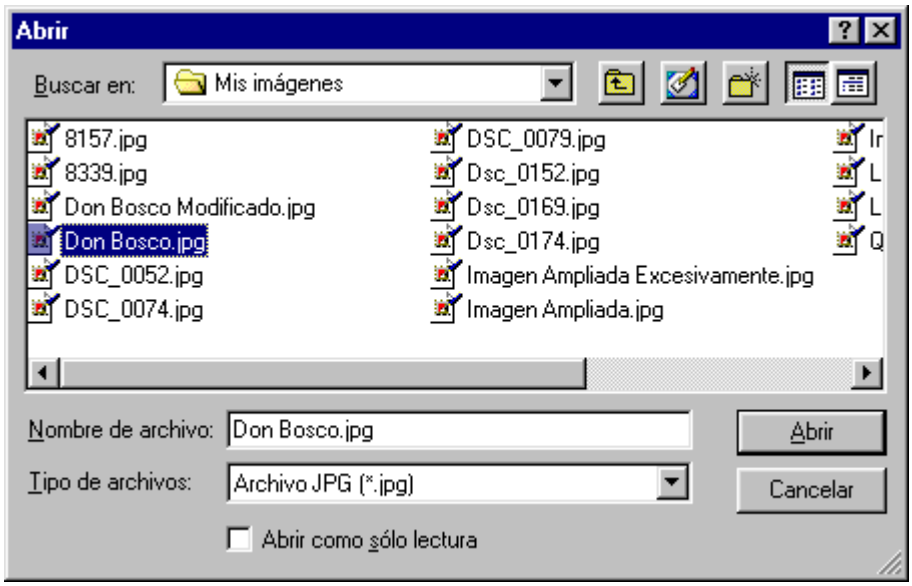


Figura 8.13. Cuadro de diálogo `Abrir` de la aplicación.

- *Guardar*. Guarda la ampliación ó reducción de la imagen, sobrescribiendo directamente la imagen original sin ningún tipo de pregunta o advertencia.
- *Guardar como...* Presenta un cuadro de dialogo por medio del cual se puede decidir en qué ruta se guardará y cómo deseamos que se llame la imagen ampliada. Puede asignársele un nombre diferente para conservar la imagen original, o puede sobrescribirse esta última, previo a un mensaje de confirmación en el que se pregunta si se desea o no sobrescribir la imagen original.

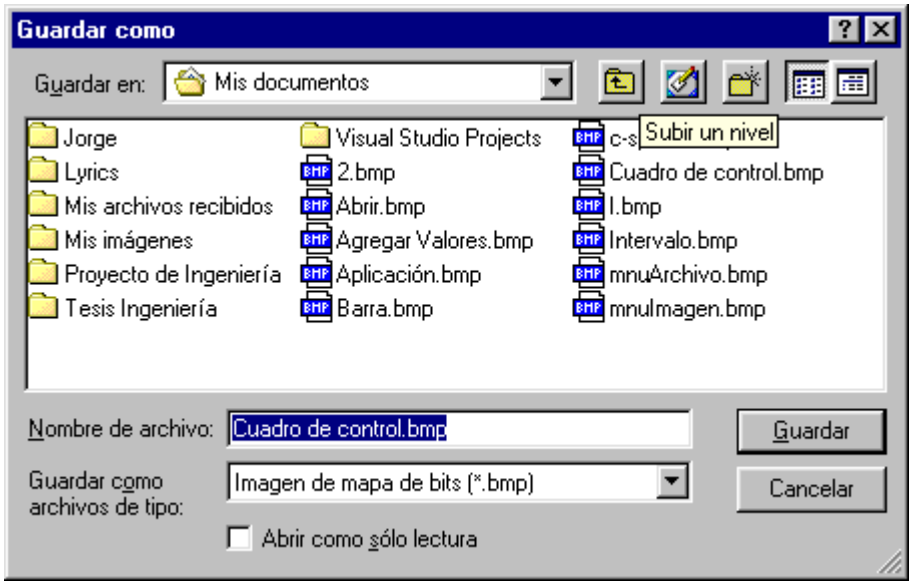


Figura 8.14. Cuadro de diálogo `Guardar como...` de la aplicación.

♦ *Salir*. Cierra la aplicación.

Menú Imagen

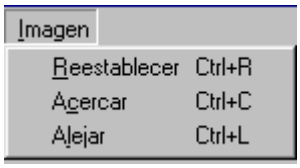


Figura 8.15. Menú `Imagen` de la aplicación.

♦ *Restablecer*. Si la imagen ha sido ampliada o reducida varias veces y no se obtiene la información requerida de la imagen, ésta puede ser restablecida a la imagen original para poder aplicarle otros métodos u otros factores de ampliado.

♦ *Acercar*. Realiza la función de ampliar en un factor de dos la imagen, cada vez que se selecciona esta opción.

♦ *Alejar*. Realiza la función de reducir en un factor dos la imagen, cada vez que se selecciona esta opción.

Menú Método

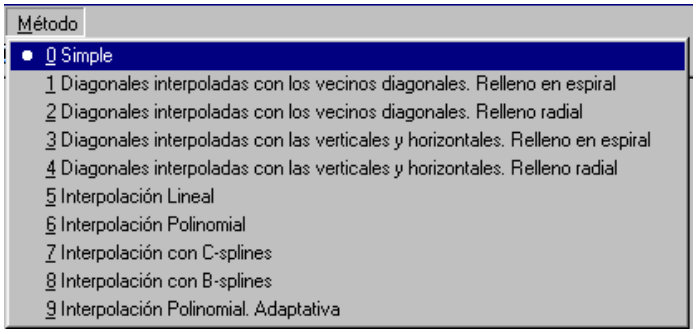


Figura 8.16. Menú `Método` de la aplicación.

Este menú despliega la lista de todos los métodos disponibles para ampliar una imagen, presentando la marca de un punto a la izquierda del método que actualmente está siendo utilizado. Por omisión, si el usuario no ha seleccionado ninguno se utilizará el "Método simple".

Para una forma de empleo más rápida se ha creado una barra de herramientas con la que también se puede tener acceso a las opciones

mencionadas anteriormente. Dicha barra consta de los elementos que se muestran en la Tabla 8.5.

Tabla 8.5. Elementos de la barra de tareas de la aplicación

Icono	Nombre	Menú equivalente
	Restablecer imagen	Imagen / Restablecer
	Abrir archivo	Archivo / Abrir
	Guardar archivo	Archivo / Guardar
	Acercar imagen	Imagen / Acercar
	Alejar imagen	Imagen / Alejar
	Combo Factor	No disponible

El combo "`Factor`" se utiliza para escoger la cantidad de ampliación.

Formas de ampliar una imagen

La aplicación maneja tres modalidades diferentes para efectuar el ampliado a una imagen, las cuales se describen en seguida.

Por medio de un recuadro

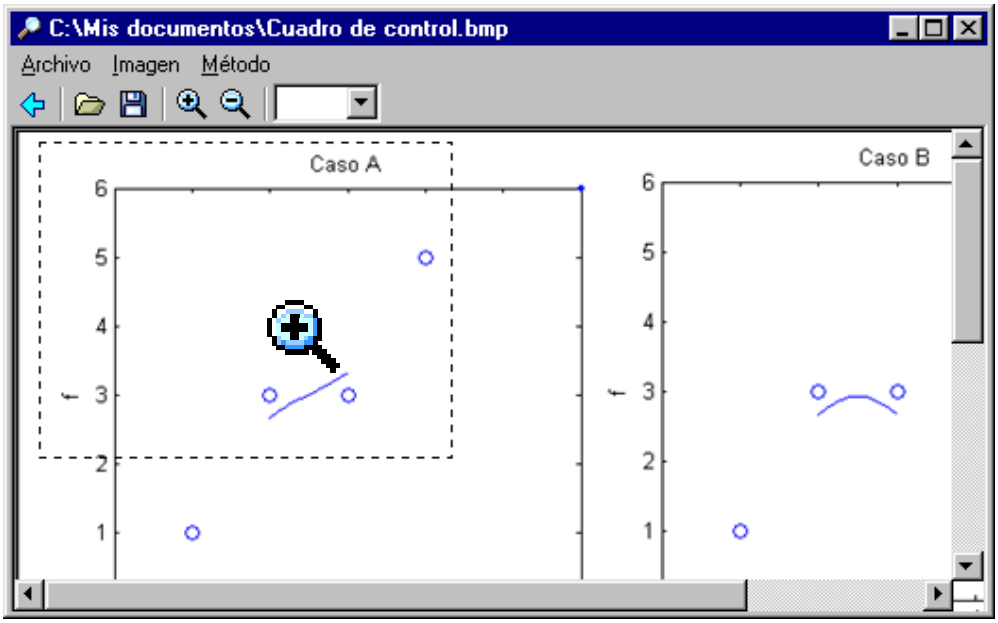


Figura 8.17. Ampliación por medio de un recuadro.

Esta modalidad es de utilidad cuando el interés se centra en pequeñas áreas dentro de la imagen, siendo innecesario realizar una ampliación de la totalidad de la misma.

Para lograr esto, hacemos clic en la imagen en una esquina del área que deseamos ampliar, y manteniendo presionado el botón izquierdo del ratón, lo movemos hacia arriba o hacia abajo, y hacia la derecha o hacia la izquierda, de manera que se genere un rectángulo de líneas discontinuas que encierre el área de interés. Una vez obtenida el área en cuestión soltamos el botón izquierdo del ratón y colocamos el puntero encima del recuadro. Al hacer esto, el puntero del ratón cambiará a una forma de lupa (), la que nos indica que podemos iniciar el proceso de ampliado haciendo clic con el botón izquierdo del ratón. El ampliado se realiza con el método que tengamos seleccionado (`Menú Método`) en ese momento. La cantidad de ampliación para la imagen es fija e igual a cinco (de forma predeterminada) por lo que el valor del combo "`Factor`" es indiferente. Si el recuadro seleccionado no es el deseado, puede hacerse clic con el botón derecho del ratón en

cualquier parte de la imagen y el recuadro desaparecerá, permitiendo la generación de un recuadro nuevo.

Por medio de las opciones `Acercar` y `Alejar`

Esto se logra utilizando las opciones `Acercar` o `Alejar` del `Menú Imagen` o los iconos con pequeñas lupas de la barra de herramientas. Al utilizar estas opciones lo que se hace es aumentar o disminuir en un factor de dos la imagen que actualmente se esta observando. A diferencia de la modalidad anterior, en este caso sí se amplia toda la imagen.

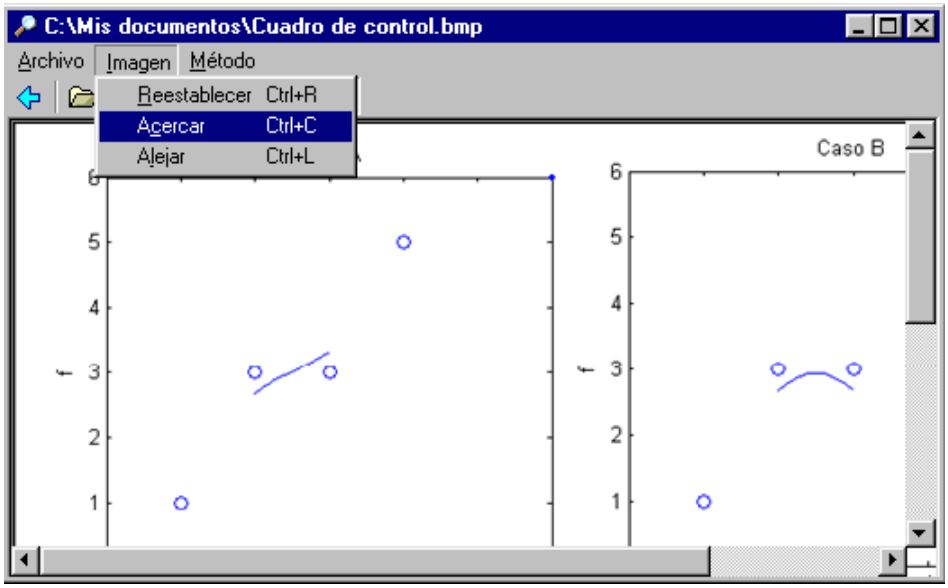


Figura 8.18. Ampliación por medio de las opciones `Acercar` y `Alejar`.

Por medio del combo "Factor"

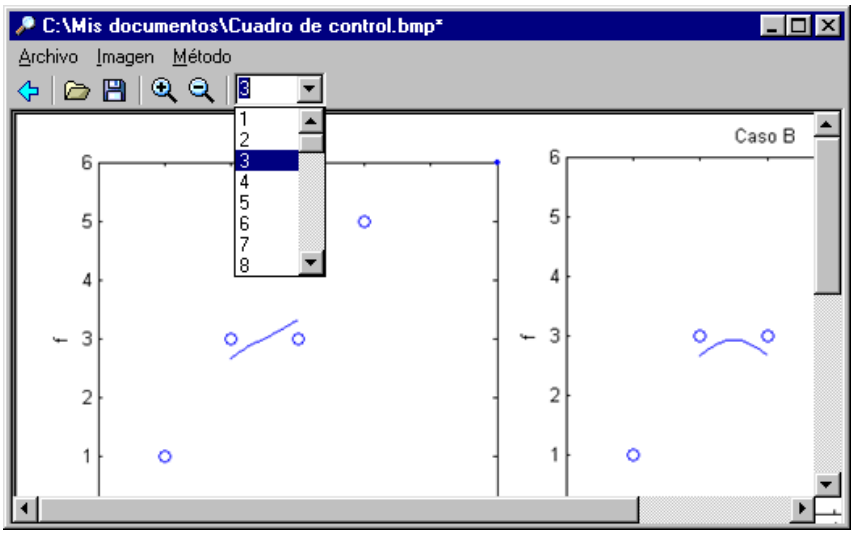


Figura 8.19. Ampliación por medio del combo `Factor`.

Al seleccionar el método mediante el `Menú Método` podemos después indicarle a la aplicación cuánto se desea ampliar la imagen mediante el combo "`Factor`". El proceso de ampliado se inicia al momento de seleccionar una cantidad en este combo. A diferencia de las otras modalidades de ampliación, en este caso sí se puede especificar el factor, mientras que con las otras modalidades se emplean valores fijos para realizar dicha tarea.

D.2. Manual del programador

La aplicación fue desarrollada con dos herramientas de programación. Los algoritmos encargados de realizar los procesos matemáticos fueron elaborados por medio de MATLAB 6.5 (véase el ANEXO B), mientras que el entorno gráfico se implementó con Microsoft Visual Basic 6.0. Se utilizaron algunos componentes comunes presentes en Visual Basic, como lo son `Microsoft Common Dialog Control 6.0` y `Microsoft Windows Common Controls 6.0`.

La aplicación es bastante sencilla ya que no requiere de la adición de muchos controles al formulario; la única referencia externa a Visual

Basic necesaria es una biblioteca que se instala junto con la aplicación al ejecutar el archivo `ZoomVB.exe`, y que se encuentra en la carpeta de Windows que se especificó a la hora de la instalación al registrar los archivos `mwcomutil.dll` y `zoom_1_0.dll` (véase la sección D.1.1 de este anexo). Dicha biblioteca se denomina `Zoom 1.0 Type Library`. Para poder utilizar el proyecto en Visual Basic es necesario hacer referencia a esta biblioteca, por medio del menú `Project / References`. En el cuadro de diálogo que aparece se recorre toda la lista y cuando se localiza la biblioteca, se selecciona y se hace clic en "`Aceptar`" para que quede registrado el cambio. Si la biblioteca no aparece en el listado, puede buscarse en el directorio donde se instaló la aplicación y seleccionar el archivo `zoom_1_0.dll`.

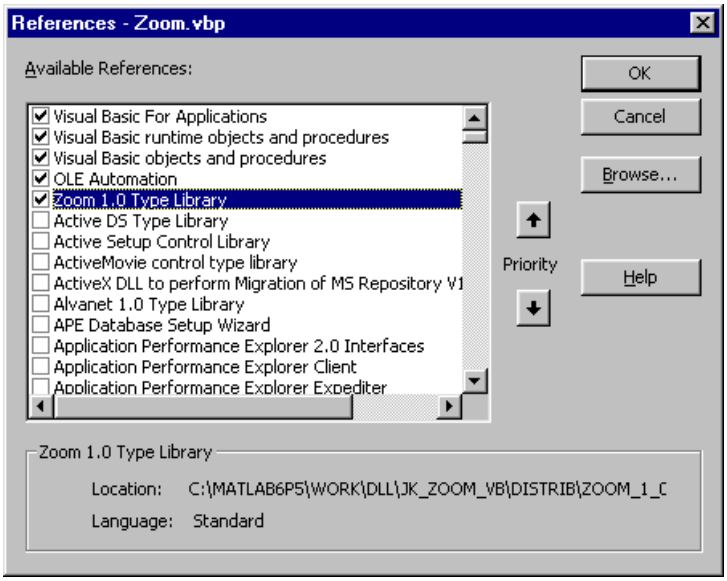


Figura 8.20. Cuadro de diálogo `Referencias`.

Para poder utilizar dicha biblioteca, en la sección General de Visual Basic se declara una variable que herede las propiedades y métodos de la misma, de esta forma:

```
Private <Nombre Variable> As New Zoom.Zoom
```

Con esto, el objeto `<Nombre Variable>` está listo para utilizar la función `JK_zoom` que es la que se encarga de ejecutar los procesos creados en MATLAB.

Una vez obtenidos todos los elementos necesarios se procede a crear el formulario principal de la aplicación.

Los objetos utilizados son los siguientes:

- `ComboBox`
- `PictureBox`
- `Vertical ScrollBar`
- `Horizontal ScrollBar`
- `ToolBar`
- `CommonDialog`
- `ImageList`

Estos objetos se han distribuido en el formulario de la siguiente manera:

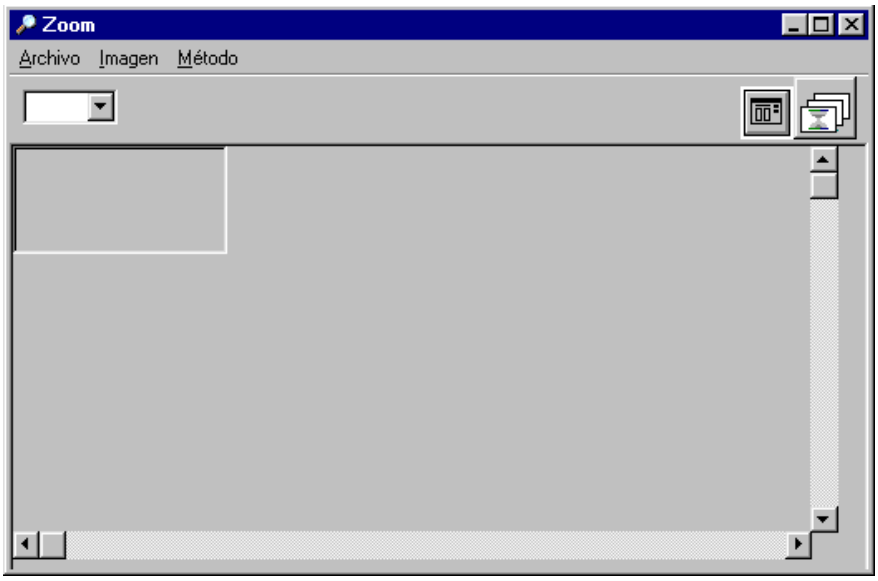


Figura 8.21. Formulario de la aplicación.

Con los objetos colocados en el formulario se procede a asignarles un nombre intuitivo a cada uno de ellos y poder así identificarlos en el código fuente. A continuación de detallan los nombres de los objetos y su propósito.

- `cmbFactor`. `ComboBox` utilizado para presentar los diversos valores posibles de la cantidad de ampliación para una imagen.

- ♦ `tbrImagen`. `ToolBar` que contiene todos los botones que se utilizan para realizar alguna modificación a la imagen.
- ♦ `imlImagen`. `ImageList` que contiene todos los iconos que se muestran en la barra de herramientas.
- ♦ `picImagen`. `PictureBox` contenedor de la imagen que se desea ampliar.
- ♦ `picImagen_Contenedor`. `PictureBox` que se utiliza para contener al objeto `picImagen`, ello debido a que cuando se colocaba el objeto `Picture` sobre el formulario ésta dejaba espacios de otro color, lo cual no era aceptable.
- ♦ `CdlImagen`. `CommonDialog` que se despliega al momento de abrir un archivo de imagen y cuando se selecciona "`Guardar como...`".

La mayoría de opciones se pueden realizar mediante el uso de la barra de herramientas, la cual es creada en tiempo de ejecución, lo que significa que las imágenes de cada uno de los botones se cargan hasta el momento en el que la aplicación se ejecuta. Esto se logra con el siguiente código dentro del evento `Load` del formulario:

Listado 8.31. Carga de la barra de herramientas en tiempo de ejecución

```
With imlImagen
      .ImageHeight = 16
      .ImageWidth = 16

      'Imágenes para la Barra de Herramientas
      Set mimgImagen = .ListImages.Add(, "Reestablecer", _
            LoadPicture(App.Path +          "\Reestablecer.ico"))
      Set mimgImagen = .ListImages.Add(, "Abrir", LoadPicture(App.Path + "\Abrir.ico"))
      Set mimgImagen = .ListImages.Add(, "Guardar", LoadPicture(App.Path + "\Guardar.ico"))
      Set mimgImagen = .ListImages.Add(, "Acercar", LoadPicture(App.Path + "\Acercar.ico"))
      Set mimgImagen = .ListImages.Add(, "Alejar", LoadPicture(App.Path + "\Alejar.ico"))

End With
```

Los botones serán colocados en la barra de tareas mediante la función `PrepararBotones`, la cual formatea cada uno de los botones asignándoles posición y un nombre que se utiliza en el evento `clic` de `tbrImagen`, para realizar mediante el uso de una función la acción que se desea.

D.2.1. Funciones utilizadas

A continuación se detallan las funciones más importantes involucradas en el proceso de ampliado.

Función `JK_zoom`[33]

Propósito

Trasladar a MATLAB los argumentos necesarios para que se amplíe una imagen determinada.

Sintaxis

```
Call (Nombre Variable).JK_zoom(nargout As Long, info,
coord_x_inicial, coord_y_inicial, coord_x_final, coord_y_final,
n, tipo, ruta_entrada, ruta_salida)
```

Argumentos

- `nargout`: Número de argumentos de salida de la función. Este argumento siempre es igual a 1.
- `info`: Variable en la que se almacena una cadena de caracteres que indica el tiempo transcurrido durante el proceso de ampliado.
- `coord_x_inicial`: Coordenada "x" de la esquina superior izquierda del recuadro de una imagen (véase la modalidad de ampliado con recuadro en la sección D.1.3). Si no se utiliza un recuadro se toma toda la imagen y los valores tanto para x como para y son de cero.
- `coord_y_inicial`: Coordenada "y" de la esquina superior izquierda del recuadro de una imagen.
- `coord_x_final`: Coordenada "x" de la esquina inferior derecha del recuadro de una imagen.

[33] Para mayor información sobre el uso en Visual Basic de la biblioteca creada con MATLAB COM Builder, consúltese el ANEXO C.

- `coord_y_final`: Coordenada "y" de la esquina inferior derecha del recuadro de una imagen.
- `n`: Cantidad de ampliación. Este valor se calcula dependiendo de la modalidad de ampliado seleccionada.
- `tipo`: Método que se utilizará para realizar el ampliado.
- `ruta_entrada`: Ruta completa (directorio más nombre de archivo) de la imagen a ser ampliada.
- `ruta_salida`: Ruta completa (directorio más nombre de archivo) donde se guardará la imagen obtenida al ampliar o disminuir la imagen original.

Resultados

Al invocar a esta función se ejecuta el proceso de ampliado de la imagen mediante las funciones creadas en MATLAB, trasladándose de una aplicación a otra de una forma transparente al usuario.

Función `AmpliarImagen`

Propósito

Ampliar la imagen utilizando la función `JK_zoom`.

Sintaxis

```
AmpliarImagen(Xdown As Integer, Ydown As Integer, Xup As Integer,
Yup As Integer, Metodo As Integer, Factor As Integer, Optional
FormaAmpliado As FormaAmpliado = zFactor)
```

Argumentos

- `Xdown`: Requerido. Coordenada "x" de la esquina superior izquierda del recuadro de una imagen; se obtiene del evento `Mouse_Down`.
- `Ydown`: Requerido. Coordenada "y" de la esquina superior izquierda del recuadro de una imagen; se obtiene del evento `Mouse_Down`.

- `Xup`: Requerido. Coordenada "x" de la esquina inferior derecha del recuadro de una imagen; se obtiene del evento `Mouse_Up`.
- `Yup`: Requerido. Coordenada "y" de la esquina inferior derecha del recuadro de una imagen; se obtiene del evento `Mouse_Up`.
- `Metodo`: Requerido. Método que se utilizará para realizar el ampliado.
- `Factor`: Requerido. Cantidad de ampliación. Se calcula dependiendo de la modalidad de ampliado empleada.
- `FormaAmpliado`: Opcional. Enumeración de las posibles modalidades de ampliado disponibles, las cuales son: Recuadro, Acercar y Factor.

Comentarios

- Esta función realiza todos los cálculos necesarios para decidir de qué forma se realizará el ampliado, siguiendo el flujo de cómo se ha ido modificando la imagen. El valor de las coordenadas "x" e "y" devuelto por los eventos `Mouse_UP` y `Mouse_Down` depende de las unidades elegidas para la escala de los objetos gráficos. De las diversas opciones para dichas unidades (véase [12]), se decidió utilizar "twips" en lugar de píxeles, debido a que los primeros son independientes del monitor utilizado, asegurando que la colocación y proporción de los elementos de la aplicación sean los mismos en todos los sistemas de pantalla. Sin embargo, la función `JK_zoom` requiere que las coordenadas de la región a ser ampliada se proporcionen en píxeles, por lo que resulta necesario realizar una conversión de unidades. El factor de 15 utilizado para dividir a las coordenadas de la imagen cumple con este propósito.

Código fuente

Listado 8.32. Procedimiento `AmpliarImagen`

```
Private Sub AmpliarImagen(Xdown As Integer, Ydown As Integer, Xup As Integer, Yup As Integer, _
      Metodo As Integer, Factor As Integer, Optional FormaAmpliado As FormaAmpliado = zFactor)

   Dim RutaEntrada As String
   Dim RutaSalida As String
```

```
   RutaSalida = App.Path + "\Imagen_Salida.bmp"

   If Not RecuadroUsado Then
      RutaEntrada = cdlImagen.FileName
   Else
      RutaEntrada = RutaSalida
   End If

   If FactorAcumulado <> 1 Then
      Call Zoom.jk_zoom(1, salida, min(Xdown, Xup) / 15, min(Ydown, Yup) / 15, _
            max(Xup, Xdown) / 15, max(Yup, Ydown) / 15, FactorAcumulado, Metodo, _
            RutaEntrada, RutaSalida)
      Xdown = 0
      Ydown = 0
      Xup = 0
      Yup = 0
      Me.Caption = cdlImagen.FileName + "*"
      If salida <> 0 Then
         RutaSalida = App.Path + "\Imagen_Salida.bmp"
         picImagen.Picture = LoadPicture(RutaSalida)
      End If
   Else
      FactorAcumulado = 0
      If Not RecuadroUsado Then
         picImagen.Picture = LoadPicture(cdlImagen.FileName)
      Else
         picImagen.Picture = LoadPicture(App.Path + "\Imagen_Recuadro.bmp")
      End If
   End If

   ReziseImagen

   If FormaAmpliado = zRecuadro Then
      RecuadroUsado = True
      SavePicture picImagen.Picture, App.Path + "\Imagen_Recuadro.bmp"
   End If

End Sub
```

Para poder utilizar la modalidad de ampliado con recuadro se ha hecho uso de los eventos `Mouse_Down`, `Mouse_Up` y `Mouse_Move` del objeto `picImagen`, como se detalla a continuación.

Listado 8.33. Evento `Mouse Down` del objeto `picImagen`

```
Private Sub picImagen_MouseDown(Button As Integer, Shift As Integer, X As Single, y As Single)
   If picImagen.MousePointer <> vbCustom Then
      Xdown = X
      Ydown = y
   End If
   picImagen.ScaleMode = vbTwips
```

```
   If Button = 1 And picImagen.MousePointer = vbCustom Then
      FactorAcumulado = 5
      Call AmpliarImagen(Xdown, Ydown, Xup, Yup, Metodo, FactorAcumulado, zRecuadro)
      FactorAcumulado = 0
   End If

   If Button = 2 Then
      picImagen.Cls
   End If

End Sub
```

Listado 8.34. Evento `Mouse Up` del objeto `picImagen`

```
Private Sub picImagen_MouseUp(Button As Integer, Shift As Integer, X As Single, y As Single)
   Xup = X
   Yup = y
   If Xdown <> 0 Then
      picImagen.Line (Xdown, Ydown)-(Xup, Yup), , B
   End If
End Sub
```

Listado 8.35. Evento `Mouse Move` del objeto `picImagen`

```
Private Sub picImagen_MouseMove(Button As Integer, Shift As Integer, X As Single, y As Single)

   picImagen.DrawStyle = 2

   If Xdown <> 0 And Ydown <> 0 And Button = 1 Then
      picImagen.Cls
      picImagen.Line (min(Xdown, X), min(Ydown, y))-(max(X, Xdown), max(y, Ydown)), , B
   End If

   If (X >= min(Xdown, Xup) And y >= min(Ydown, Yup)) And _
     (X <= max(Xup, Xdown) And y <= max(Yup, Ydown)) Then
      picImagen.MouseIcon = LoadPicture(App.Path + "\Acercar.ico")
      picImagen.MousePointer = vbCustom
   Else
      picImagen.MousePointer = vbDefault
   End If
End Sub
```

En el `Menú Método`, se puede observar la marca de un punto en el método que actualmente se está utilizando; esto se logra mediante el uso de una API (acrónimo de Application Program Interface, o Interface de Programa de Aplicación), la cual utiliza los servicios de bajo nivel del sistema operativo. Las API han sido declaradas en el módulo `basImagen` para que puedan ser utilizadas en las demás secciones del código.

Listado 8.36. Declaración de las API

```
Declare Function GetMenu Lib "user32" (ByVal hwnd As Long) As Long
Declare Function GetSubMenu Lib "user32" (ByVal hMenu As Long, ByVal nPos As Long) As Long
Declare Function CheckMenuRadioItem Lib "user32" (ByVal hMenu As Long, ByVal un1 As Long, ByVal un2
As Long, ByVal un3 As Long, ByVal un4 As Long) As Long

Public Const MF_BYPOSITION = &H400&
```

Función `GetMenu`

Propósito

Conseguir el número único del manejador[34] del objeto Menú, definido en el formulario.

Sintaxis

```
Menu = GetMenu(hWnd)
```

Argumentos

- `hWnd`: Manejador del formulario del que se requiere el manejador del menú.

Resultados

- `Menu`: Manejador del menú.

Comentarios

- El valor que retorna es el manejador del menú. Si la ventana actual no tiene menú, el valor de retorno es `NULL`; y si el formulario es un formulario hijo, el valor de retorno es indefinido.

Función `GetSubMenu`

---

[34] El manejador es un número asignado a cada control u objeto del formulario, para poder identificarlo dentro de la aplicación.

Propósito

Conseguir el número único del manejador del Submenú activado por el ítem especificado del Menú.

Sintaxis

```
SubMenu = GetSubMenu(hMenu,nPos)
```

Argumentos

- `hMenu`: Manejador del Menú.
- `nPos`: Especifica el valor relativo del ítem activado dentro del Submenú.

Resultados

- `SubMenu`: Manejador del Submenú.

Comentarios

- Sí la función se realiza con éxito, ésta retorna el manejador del Submenú activado por el ítem del Menú. Si el ítem del Menú no activa ningún Submenú, el valor devuelto es `NULL`.

Función `CheckMenuRadioItem`

Propósito

Marca con un punto el ítem del Menú activado. Al mismo tiempo, la función desmarca todos los otros ítems que estuviesen marcados dentro del mismo grupo del Menú.

Sintaxis

```
Ret = CheckMenuRadioItem (hMenu,idFirst, idLast, idCheck, uFlags)
```

Argumentos

- `hMenu`: Manejador del Menú que contiene el grupo de ítems.

- idFirst: Posición o indicador del primer ítem en el grupo del Menú.
- idLast: Posición o indicador del último ítem en el grupo del Menú.
- idCheck: Posición o indicador del ítem a marcar.
- uFlags: Valor que especifica el significado de idFirst, idLast, idCheck. Si el parámetro es MF_BYCOMMAND, los otros parámetros indican los identificadores de cada ítem del Menú; y si es MF_BYPOSITION, los parámetros indican la posición de los ítems del Menú (para este Trabajo de Graduación se utilizó MF_BYPOSITION).

Resultados

- Ret: Si la función se ejecuta sin error el resultado es un número diferente de cero; de lo contrario es cero.

Comentarios

La función establece las banderas MFT_RADIOCHECK y MFS_CHECKED del ítem especificado por idCheck; al mismo tiempo limpia ambas banderas de los otros ítems miembros del grupo. Esta función utiliza un punto (bullet bitmap) en lugar de un cheque (check-mark bitmap) que es la opción común que presenta el editor de menús de Visual Basic.

La función creada para utilizar estas API es SetMenuRadio, la cual se define a continuación.

Función SetMenuRadio

Propósito

Función que utiliza las API anteriormente mencionadas para marcar con un punto el ítem seleccionado del Menú.

Sintaxis

```
SetMenuRadio (frm, PosicionInicial, PosicionActual,
PosicionDesde, PosicionHasta)
```

Argumentos

- `frm`: Formulario del que se obtiene el manejador para ser utilizado por `GetMenu`.
- `PosicionInicial`: Posición del grupo de Menú que contiene a los ítems a seleccionar.
- `PosicionActual`: Posición del ítem que se desea seleccionar.
- `PosicionDesde`: Posición del primer ítem del grupo que pertenece al Menú.
- `PosicionHasta`: Posición del último ítem del grupo que pertenece al Menú.

Código fuente

Listado 8.37. Función `SetMenuRadio`

```
Function SetMenuRadio(frm As Form, PosicionInicial As Integer, PosicionActual As Integer, PosicionDesde
As Integer, PosicionHasta As Integer)
   Dim Ret As Long
   Dim Menu As Long
   Dim SubMenu As Long
   Dim lhWnd As Variant
   Metodo = PosicionActual - 1
   lhWnd = frm.hwnd
   Menu = GetMenu(lhWnd)   '0 = error
   SubMenu = GetSubMenu(Menu, PosicionInicial - 1) '0 = error
   Ret = CheckMenuRadioItem(SubMenu, PosicionDesde - 1, PosicionHasta - 1, _
         PosicionActual - 1, MF_BYPOSITION)
End Function
```

D.2.2. Proceso de empaquetamiento

Una vez terminado el proyecto de Visual Basic, es necesario generar un paquete instalador capaz de copiar todos los archivos necesarios para que la aplicación pueda ejecutarse sin ningún problema en la computadora cliente.

Para generar el paquete de instalación se utilizó Visual Studio Installer 1.1. Ésta es una herramienta que le agrega a Visual Interdev

6.0 la opción de generar proyectos del tipo Installer. Esta herramienta requiere del Service Pack 5 para Visual Studio 6.0.

A continuación se detalla paso a paso el proceso de creación de un paquete de instalación.

1. En primer lugar se ejecuta Visual Studio Installer, localizado en `Menú Inicio / Programas / Microsoft Visual Studio 6.0 / Microsoft Visual Studio 6.0 Enterprise Tools / Visual Studio Installer`. Una vez abierta esta aplicación, ésta preguntará al usuario qué desea hacer. Entonces se selecciona un proyecto de instalación de Visual Basic Installer. En el campo "`Name`" se escribe el nombre que se le desea asignar al archivo empaquetador de extensión `msi`. En "`Location`" se especifica la ruta de Windows donde se alojará tanto el proyecto del instalador (extensión `sln`) como el archivo empaquetador.

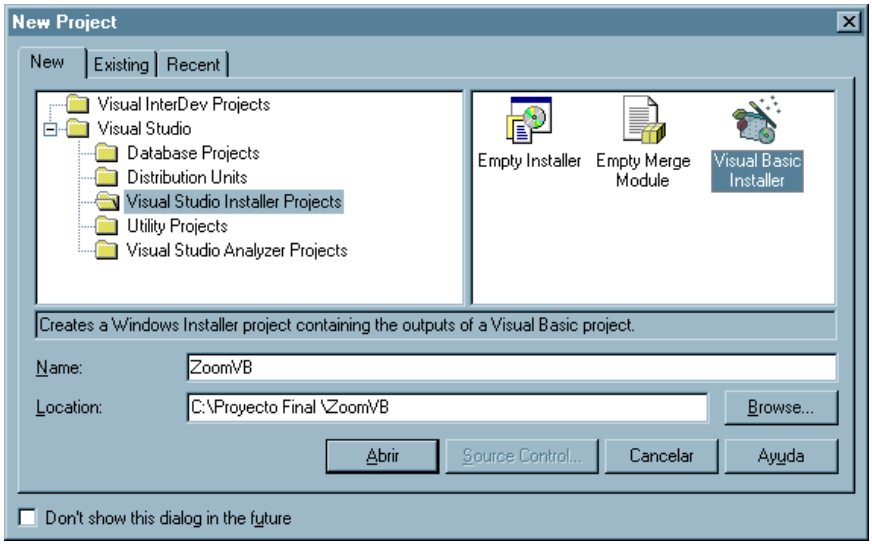


Figura 8.22. Cuadro de diálogo `New Project` de Visual Studio Installer.

2. En el cuadro de diálogo anterior se presiona el botón `Abrir` y se presentará una ventana en la que se debe indicar cuál es el proyecto al que se hará referencia en el proceso de instalación. Para que el empaquetador funcione debe existir en la misma carpeta donde se encuentra el proyecto un archivo ejecutable (extensión `exe`) compilado de la aplicación que se desea empaquetar. Como lo que se

desea es crear un asistente de instalación, cuando se le pregunta al usuario por el tipo de proyecto que desea crear, la respuesta es uno de tipo `Installer`.

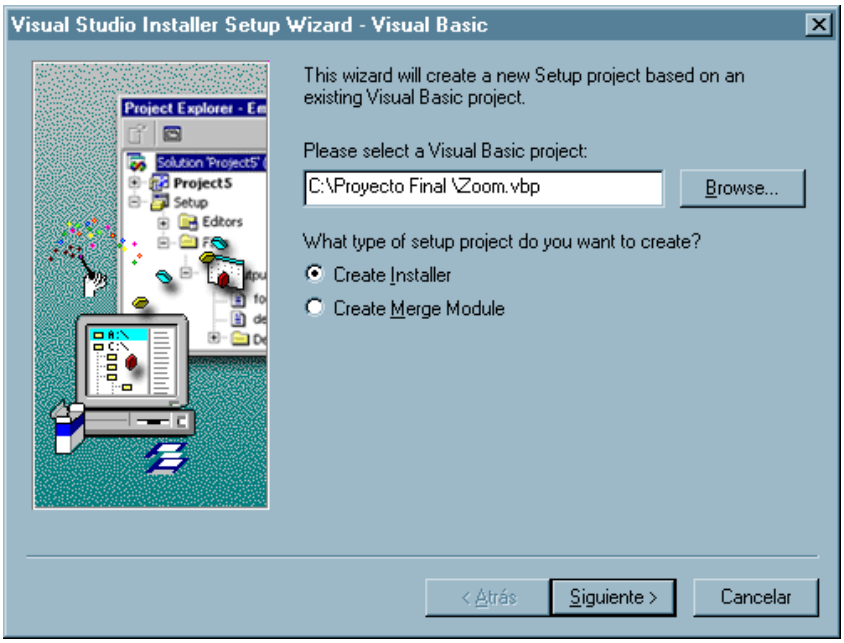


Figura 8.23. Asistente de instalación de Visual Studio Installer.

3. A continuación se presiona el botón `Siguiente`, presentándose la última pantalla antes de generar el proyecto. En esta pantalla se presenta la opción de generar un paquete que sea capaz de ser ejecutado en cualquier versión del sistema operativo Windows o sólo en la que se está creando el paquete. Para abarcar un mayor rango de Sistemas Operativos se deja la opción por omisión.

Figura 8.24. Elección del sistema operativo en Visual Studio Installer.

Al hacer clic en `Finalizar`, el programa empieza a compilar y construir el paquete, reconociendo todos los objetos y bibliotecas que se hayan utilizado en el desarrollo del proyecto (`.vbp`) creado en Visual Basic. Una vez reconocidos todos estos elementos, agrega como dependencias del proyecto instalador a dichos archivos. Como es de esperarse, en muchos casos no se reconocen todos los archivos necesarios para que la aplicación funcione, ya que las bibliotecas utilizadas hacen referencia a otros archivos fuera de la aplicación, los que deben ser agregados manualmente.

En el Explorador de Proyectos se puede observar que por el momento existen algunas dependencias entre los archivos utilizados en el proyecto (`.vbp`) y el archivo ejecutable que contiene la aplicación.

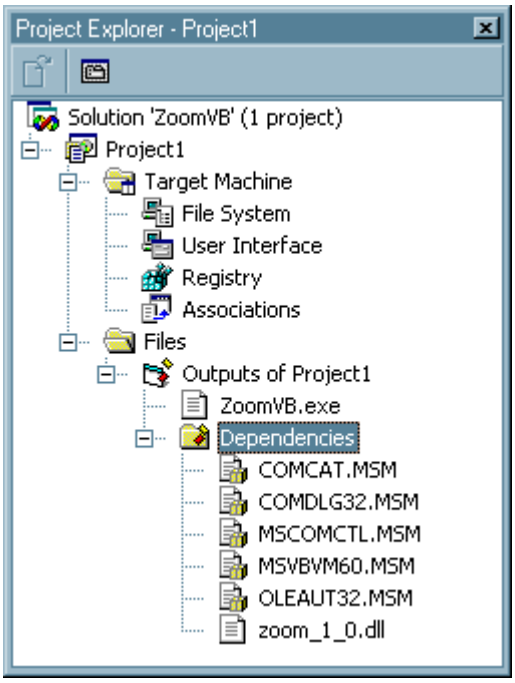


Figura 8.25. Explorador de Proyectos de Visual Studio Installer.

Estos archivos no son suficientes para que se pueda ejecutar la aplicación, por lo que se procede a agregar más archivos de dependencia, los cuales son los que MATLAB utiliza. Esto se lleva a cabo haciendo clic derecho en la carpeta `Files` del Explorador de Proyectos y seleccionando `Add / File(s)`, presentando un cuadro de diálogo en el que se pueden seleccionar los archivos a agregar.

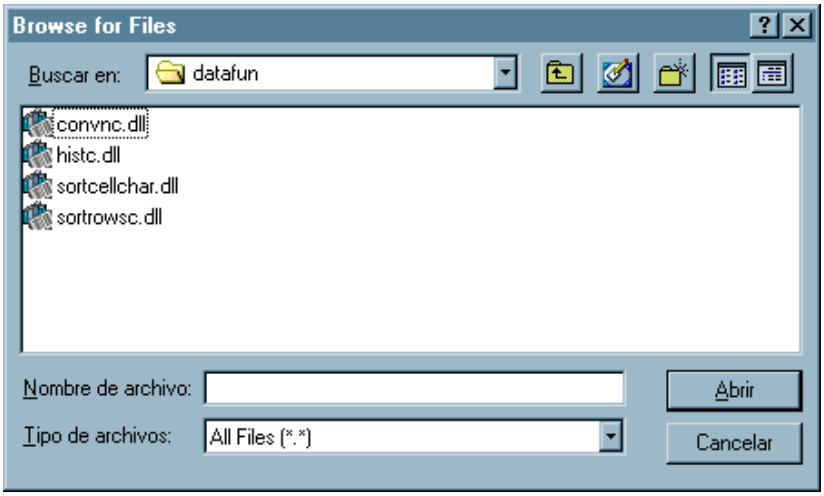


Figura 8.26. Cuadro de diálogo `Browse for Files` de Visual Studio Installer.

De esta forma se pueden agregar tantos archivos como sea necesario para que la aplicación funcione. Los archivos pueden ser de cualquier tipo, como por ejemplo `dll` y `ocx` para el uso de bibliotecas; `bmp`, `jpg`, e `ico` por si la aplicación necesita imágenes que se han utilizado como fondos o iconos de algún botón; `txt` para archivos de ayuda como un "Readme.txt", etc.

Una vez agregados estos archivos se selecciona en qué carpeta se desean copiar; por omisión se agregan a la carpeta de la aplicación (Application Folder, carpeta donde el usuario especifica que la aplicación sea instalada). En nuestro caso, está bien que los archivos se alojen en esa ruta; pero si en algún caso es necesario agregar archivos a la carpeta de Windows (Windows Folder), a la carpeta de sistema de Windows (Windows System Folder), o a alguna carpeta en especial, se pueden agregar desde el menú desplegable `File System on Target Machine` que se muestra al seleccionar `File System` del Explorador de Proyectos. Al hacer clic derecho sobre dicho menú, con la opción `Add special folder` se puede agregar una gran variedad de carpetas del sistema. O si se desea agregar una carpeta creada por el usuario dentro de una determinada carpeta de sistema, se lleva a cabo haciendo clic derecho sobre la carpeta de sistema específica y se elige la opción `Add Folder`.

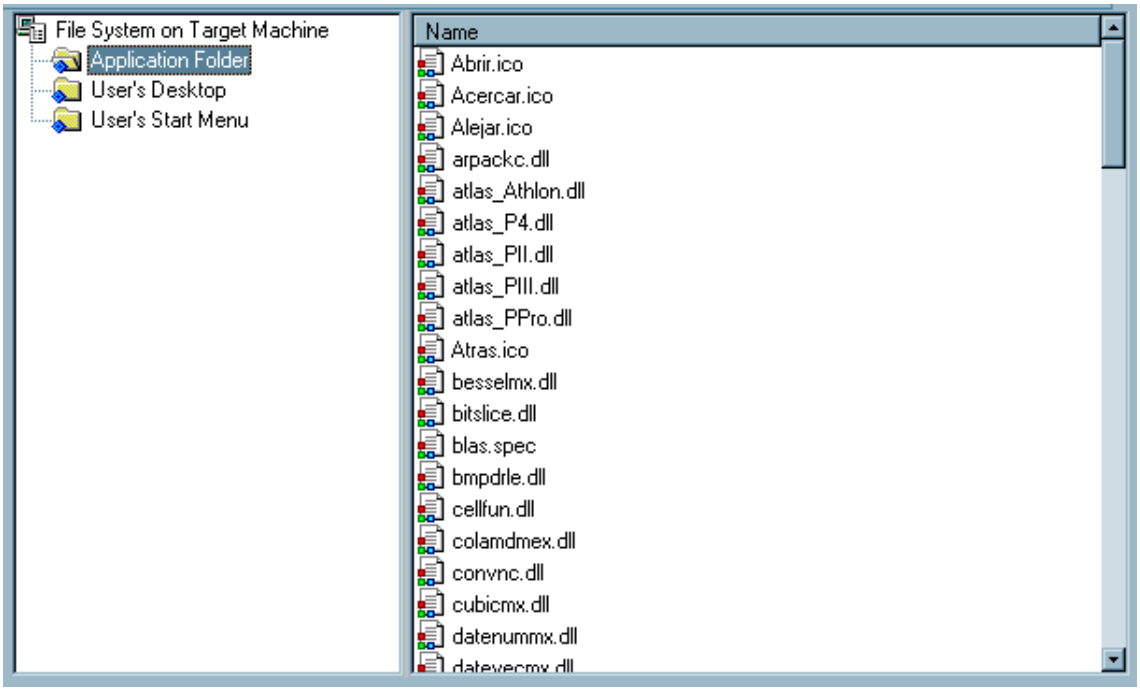


Figura 8.27. Explorador de carpetas de sistema de Visual Studio Installer.

Algunas opciones adicionales se pueden agregar en las propiedades del proyecto de instalación, como el nombre del producto, el autor, la versión, etc. Una vez teniendo todas las referencias a todos los archivos necesarios de procede a construir el paquete mediante el Menú `Build` del Menú principal y seleccionando la opción `Build` (Construir). Con esto, el programa compila y agrega los archivos necesarios y construye el instalador.

El producto final es un asistente de instalación de extensión `msi` con el que el usuario puede proceder a instalar la aplicación en una computadora. El archivo `<Nombre del instalador>.msi` se encuentra en la carpeta `<Ruta del Instaldor>\Output\Disk1`, en donde <Ruta del Instalador> es la ruta que se definió en el momento de crear el proyecto de instalación y en la que se guarda el proyecto con extensión `sln` para futuras modificaciones. Una de las ventajas de este instalador es que si la aplicación en Visual Basic se modifica y no se agrega alguna referencia adicional a las que ya se tenían, sólo se

genera un nuevo ejecutable y se sobrescribe el que el usuario tenía anteriormente, sin hacer cambios al paquete de instalación.

ANEXO E. SOLUCIÓN DE LA ECUACIÓN TRIDIAGONAL[35]

Una ecuación tridiagonal tiene la siguiente forma:

$$\begin{bmatrix} B_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & A_i & B_i & C_i & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & A_{N-1} & B_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & A_N & B_N \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_i \\ \vdots \\ \phi_{N-1} \\ \phi_N \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_i \\ \vdots \\ D_{N-1} \\ D_N \end{bmatrix} \qquad \text{(E.1)}$$

El algoritmo de resolución, llamado solución tridiagonal, es una variante de la eliminación de Gauss, y consta de los siguientes pasos:

- Inicializar las dos nuevas variables:

$$B_1' = B_1, \text{ y } \mathrm{D}_1' = D_1 \qquad \text{(E.2)}$$

- Calcular repetidamente las siguientes ecuaciones en orden de i creciente, hasta llegar a $i = N$:

$$\begin{aligned} R &= A_i / B_{i-1}' \\ B_i' &= B_i - RC_{i-1} \\ D_i' &= D_i - RD_{i-1}' \end{aligned} \qquad \text{(E.3)}$$

para $i = 2,3,\ldots,N$.

- Calcular la solución para la última incógnita con:

$$\phi_N = D_N' / B_N' \qquad \text{(E.4)}$$

- Calcular la siguiente ecuación en orden de i decreciente:

[35] Adaptado de [4].

$$\phi_i = (D'_i - C_i\phi_{i+1}) / B'_i,\ i = N-1,\ldots,2,1 \qquad \text{(E.5)}$$

En un programa para computadora no es necesario distinguir las variables primadas de las no primadas, porque ambas se almacenan en el mismo lugar en la memoria. Por tanto, el paso que genera la ecuación (E.2) no es necesario en la programación práctica. La función de archivo M de MATLAB creada para encontrar la solución de un sistema tridiagonal se nombró `tri_diag` y su código fuente se describe en el ANEXO B.

9. FUENTES DE INFORMACIÓN

A continuación se enlistan, agrupadas por tema, las fuentes de información utilizadas como referencias a lo largo del documento, así como aquellas que les fueron útiles a los autores para obtener un mejor panorama de los tópicos abordados, o que pueden ser de utilidad para el lector interesado.

Sobre ajuste de curvas

[1] CHAPRA, Steven C. & CANALE, Raymond P. (1988). *Métodos numéricos para ingenieros con aplicaciones en computadoras personales*. 1ª edición. México: McGraw–Hill / Interamericana de México, S. A. de C. V.

[2] KINCAID, David & CHENEY, Ward. (1994). *Análisis numérico. Las matemáticas del cálculo científico*. 1ª edición. EE. UU.: Addison–Wesley Iberoamericana, S. A.

[3] NAKAMURA, Shoichiro (1992). *Métodos numéricos aplicados con software*. 1ª edición. México: Prentice–Hall Hispanoamericana, S. A.

[4] NAKAMURA, Shoichiro (1997). *Análisis numérico y visualización gráfica con MATLAB*. 1ª edición. México: Prentice–Hall Hispanoamericana, S. A.

[5] UNSER, Michael (1999). *Splines. A Perfect Fit for Signal and Image Processing*. IEEE Signal Processing Magazine, November 1999.

Sobre imágenes, imágenes digitales

[6] BIBLIOTECA DE LA UNIVERSIDAD DE CORNELL / DEPARTAMENTO DE CONSERVACIÓN Y PRESERVACIÓN. (2001). Tutorial de digitalización de imágenes. Terminología básica. *Llevando la Teoría a la Práctica*. Disponible:

— Imágenes digitales:
http://www.library.cornell.edu/preservation/tutorial-spanish/intro/intro-01.html

— Resolución:

http://www.library.cornell.edu/preservation/tutorial-spanish/intro/intro-02.html
Descargado: septiembre de 2002.

[7] DE LA ESCALERA HUESO, Arturo Moreno & ARMINGOL MORENO, José María (1994). *Introducción a la visión por computador*. Leganés (Madrid): Universidad Carlos III de Madrid, Escuela Politécnica Superior, Centro de Promoción de la Formación y la Innovación, Área de Ingeniería de Sistemas y Automática. Curso 5.3: "Visión Artificial".

[8] DOMÍNGUEZ TORRES, Alejandro. *Procesamiento digital de imágenes*. Disponible:
http://www.cesu.unam.mx/iresie/revistas/perfiles/perfiles/72-html/72-06.htm
Descargado: septiembre de 2002.

[9] SÁNCHEZ MUÑOZ, Gustavo. (Mayo de 2001). *Las imágenes digitales en dos dimensiones*.
Disponible:
http://www.gusgsm.com/html/imagen.html
Descargado: septiembre de 2002.

Sobre las herramientas de desarrollo

[10] MATHWORKS, Inc. (The). (2002). Ayuda en pantalla del programa de computadora: MATLAB version 6.5.0.180913a release 13, June 18, 2002.

[11] MATHWORKS, Inc. (The). Página web oficial.
Disponible:
http://www.mathworks.com
Última visita: febrero de 2003.

[12] MICROSOFT Corporation. *MSDN Library Visual Studio 6.0*. Ayuda en pantalla del programa de computadora: Microsoft Visual Studio 6.0.
Disponible:
http://www.microsoft.com/spanish/msdn/elsalvador/default.asp
Última visita: febrero de 2003.