UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"Técnica Multigrid en la solución del problema de restauración digital"

T E S I S

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

Aristóteles Federico Núñez Juárez

DIRECTOR DE TESIS:

M.C. Luis René Marcial Castillo,

HUAJUAPAN DE LEÓN, OAX. SEPTIEMBRE DE 2007

UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"Técnica Multigrid en la solución del problema de restauración digital"

T E S I S

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

Aristóteles Federico Núñez Juárez

Jurado

M.C. Verónica Rodríguez López

Presidente

M.C. Luis René Marcial Castillo

Vocal

M.I.A Hilda Caballero Barbosa

Secretario

HUAJUAPAN DE LEÓN, OAX. SEPTIEMBRE DE 2007

Gracias a Dios por bendecirme más de lo que merezco y darme una familia excepcional, que ha realizado lo que parecía imposible por ver este sueño hecho realidad.

Gracias a mi Mamita y a mi Papá por siempre confiar en mi,y enseñarme, más que con libros, con el ejemplo.

Gracias Ale, Ana y Lis por apoyarme incondicionalmente, sin olvidar a Us y Alexei futuros profesionitas.

Finalmente gracias a la mujer de mi vida Sahadi, y a mi precioso hijo Gary, motivación de mi esfuerzo diario.

Especial agradecimiento a mi Director de tesis M.C. Luis René Marcial Castillo, a M.C. Verónica Rodríguez López, y a M.C. Hilda Caballero Barbosa, profesores sin igual, que me apoyaron y motivaron hasta el final.

Índice general

Capítulo 1

Introducción

En la actualidad mediante el procesamiento digital de imágenes es posible manipular imágenes con la computadora, es decir, la imagen es representada como un conjunto de datos (ver Figura 1.1) los cuales pueden ser expuestos a diversas operaciones principalmente para:

- Mejorar de alguna forma a la imagen.
- Para ayudar a su interpretación.
- Para extraer algún tipo de información útil de ella.

$$f = \begin{bmatrix} a_{0,0} & a_{1,0} & \cdots & a_{m,0} \\ a_{1,0} & a_{1,1} & \cdots & a_{m,1} \\ a_{0,0} & a_{1,0} & \cdots & a_{m,n} \end{bmatrix}$$

Figura 1.1: Representación matricial de una imagen de $(m+1) \times (n+1)$ píxeles.

Todo este tipo de procesamiento ha sido utilizado por diversas disciplinas, cada una con diferentes grados de éxito, entre ellas se mencionan solo algunas como:

- Medicina, por ejemplo: inspección visual automática.
- Biología, por ejemplo: Medición de características geométricas y de color de objetos.

- Ingeniería, por ejemplo: Detección de presencia de objetos, restauración de imágenes.

Diversos factores indican que este campo continuará creciendo, entre ellos se encuentran los éxitos obtenidos, la continua y exhaustiva investigación que se está llevando a cabo, la baja de costos en equipos de cómputo (cada vez más potentes) y la disponibilidad de mejor equipo para digitalizar y desplegar imágenes.

A medida que estos factores se han ido introduciendo han traido consigo problemas que es necesario ir resolviendo. Por ejemplo, equipos digitales nuevos y más potentes permiten capturar imágenes de alta resolución, se está hablando que en la actualidad una cámara digital fácilmente alcanza los siete megapixeles.

Esto trae consigo, que cada vez las imágenes son más grandes y que los algoritmos que en un principio funcionaban sin dificultad en ellas ahora son considerados lentos si no es que inaplicables. Por ello es necesario investigar y desarrollar otros algoritmos con distintos enfoques que permitan seguir con el procesamiento digital de imágenes de forma fluida y sin complicaciones.

Otro problema es que los equipos digitales al capturar las imágenes, la mayoría de las veces introducen ruido a éstas. Por lo que los resultados del procesamiento digital de estas imágenes se verán afectados. Por ejemplo, al tomar fotografías en un ambiente de poca luz, es común que las cámaras digitales introduzcan puntos negros a la imagen debido a que no tienen el tiempo de exposición correcto, y no toman la información necesaria del ambiente.

Es así como surge la idea de desarrollar y aplicar nuevos algoritmos que mejoren la calidad de la imagen. Por ello el presente trabajo se enfoca en la restauración y reconstrucción de imágenes que han sido degradadas, basándose en algún tipo de conocimiento a priori sobre el proceso de degradación.

Aún cuando este problema ya ha sido resuelto por diversos métodos, como lo son Gauss Seidel, Gradiente conjugado, Jacobi, entre otros, resulta que tienen como

principal desventaja el ser muy lentos de ahí que el objetivo principal de este trabajo es el de aplicar la Técnica Multigrid combinando Estimación Bayesiana y Campos Aleatorios Markovianos, para hacer la reconstrucción de una imagen que ha sido degradada. Con el uso de las técnicas de Estimación Bayesiana y Campos Aleatorios Markovianos se pretende que la restauración de la imagen sea más rápida.

Este trabajo se basa en el artículo de J. L. Marroquín [12], quien es una de las personas en México que se ha dedicado al estudio del Algoritmo Multigrid aplicado a diversas áreas, en este caso a las imágenes.

En el capítulo 2 se explican conceptos básicos necesarios en el área de restauración digital de imágenes y de optimización.

En el capítulo 3 se habla de Estimación Bayesiana, que proporciona el fundamento matemático necesario para establecer las probabilidades de soluciones candidatas a nuestro problema, definiendo lo que es la Inferencia Bayesiana, el Teorema de Bayes, y el Estimador Máximo a Posteriori.

El capítulo 4 trata de Campos Aleatorios Markovianos, que permite establecer sistemas de vecindades y cliques para cada uno de los píxeles de las imágenes que serán restauradas.

En el capítulo 5, se describen los métodos de interpolación, ya que uno de ellos es el utilizado por el método Multigrid, siendo necesario elegirlo de acuerdo a la dificultad y eficacia que conlleva implementarlo.

En el capítulo 6 se habla de Programación no lineal sin restricciones, donde se explica como resolver problemas que involucran la búsqueda de minimizadores de funciones objetivo, que es primordial en la solución del problema de restauración digital. Los métodos que se describen en este capítulo son el método de Jacobi, Gauss Seidel, y Gradiente. Éstos son utilizados para comprobar la eficacia del método Multigrid.

En el capítulo 7 se desarrolla el método Multigrid, tema principal de este trabajo de tesis, en donde se conjuntan los conocimientos de los capítulos previos para poder

solucionar el problema de restauración digital.

Con el programa ImageRestore, desarrollado conjuntamente en este trabajo, se realizaron diversas pruebas de las cuales se habla en el capítulo 8. Estas pruebas nos permiten dar las conclusiones sobre la efectividad y rapidez de cada uno de los métodos, y corroborar el objetivo inicial de este trabajo.

Se agregaron varios apéndices que proporcionan información complementaria que es de gran ayuda para la comprensión general de este trabajo de tesis.

En el apéndice A se proporcionan las definiciones de cada uno de los tipos de ruido que fueron aplicados a las imágenes de prueba.

En el apéndice B se muestran los algoritmos de los métodos de Programación no lineal sin restricciones, que fueron implementados en este trabajo, para poder comparar el desempeño del método Multigrid.

En el apéndice C se detalla el código de las funciones principales utilizadas en el método Multigrid. Este código se encuentra en el lenguaje de programación C#.

En el apéndice D se muestra el manual de usuario que permite conocer el programa ImageRestore.

Por último en el apéndice E se encuentra el Glosario con términos poco comunes que se ocupan a lo largo de este trabajo.

Capítulo 2

Conceptos Básicos

2.1. Representación de las imágenes

Para poder realizar el proceso de restauración de la imagen, es necesario primero contar con una representación útil de ella en la computadora.

Lo más común es utilizar una señal discreta de dos dimensiones. Matemáticamente dicha señal puede ser representada como función de dos variables independientes f [11].

Esta función puede ser vista como una matriz de $M \times N$ elementos (ver Figura 2.1), en donde cada elemento es denominado píxel.

$$f = \begin{bmatrix} a_{0,0} & a_{1,0} & \cdots & a_{m,0} \\ a_{1,0} & a_{1,1} & \cdots & a_{m,1} \\ a_{0,0} & a_{1,0} & \cdots & a_{m,n} \end{bmatrix}$$

Figura 2.1: Cada elemento en la función x, y representa un píxel.

Cada píxel representa un nivel de brillo en dicho punto. Esto puede ser de dos formas principalmente:

1. **Nivel de escala de grises.** Mediante esta representación cada píxel toma un valor $0 \leq f(x, y) \leq 255$.

2. **Nivel de color.** Para representar el valor de brillo de un píxel a color existen varios esquemas, dentro de los se encuentran:

   a) Tripletas RGB. $f(x, y) = \{R, G, B\}$ donde R es el nivel de Rojo, G el nivel de verde y B el nivel de azul, por sus siglas en inglés {Red, Green, Blue}, que son en realidad los tres colores primarios usados para sintetizar cualquiera de los 2^{24} (aproximádamente 16 millones) de colores.

   b) CMYK. Esquema que es representado por niveles de Cyan, Magenta, Amarillo y Negro, bastante útil en impresiones.

2.2. Restauración digital de imágenes

2.2.1. Definición

La restauración digital de imágenes *consiste en quitar o reducir las degradaciones en las que se incurrieron mientras la imagen digital fue obtenida. Estas degradaciones incluyen emborronamiento o empañamiento ocasionado por sistemas ópticos, movimiento de la imagen, asimismo ruido por fuentes electrónicas y fotométricas [6].*

Principalmente existen dos enfoques para restaurar una imagen [6]:

1. Si se tiene poco, o nulo, conocimiento acerca de la imagen original [1] se puede intentar modelar y caracterizar los orígenes de la degradación, e implementar un proceso diseñado para remover o reducir sus efectos. Este es un enfoque de estimaciones, ya que se intenta estimar lo que la imagen debió ser antes de que fuera degradada.

[1] Con imagen original se quiere dar a entender que se trata de la imagen que no tiene ningún tipo de degradación

2. Cuando el conocimiento "a priori" [2] de la imagen está disponible, esto puede ser más fructuoso, ya que se puede desarrollar un modelo matemático para la imagen original y luego hacer que el modelo se ajuste a la imagen observada.

En este trabajo se utilizará conocimiento a priori de la imagen, es decir se cuenta con las imágenes originales y el tipo de ruido que originó a las degradadas, aunque este conocimiento no es necesario en este caso por que para lo que se utiliza es para comprobar que tan bueno es el algoritmo que se está utilizando.

2.2.2. Modelo de degradación

El proceso de degradación está modelado como un operador (por un sistema) H, que junto a un término aditivo de ruido n actua sobre una imagen de entrada f para producir una imagen degradada g. La restauración digital de imágenes puede entenderse como el proceso de obtener de forma aproximada f, a partir de g y del conocimiento de la degradación en la forma del operador H. Se supone que el conocimiento de n se limita a una información de naturaleza estadística. La relación entrada-salida del modelo de degradación se expresa como [5]:

$$g(x, y) = H[f(x, y)] + n(x, y). \tag{2.1}$$

[2] "A priori" significa "antes de conocer el resultado de un acontecimiento, de un experimento o de una elección al azar" [3]

Capítulo 3

Estimación Bayesiana

Al trabajar con restauración de imágenes, lo que se desea obtener es una imagen sin, o con el menor ruido posible (ver el Modelo de Degradacion 2.2.2). Se puede nombrar a esta imagen objetivo, imagen f. Ahora bien, en este caso con lo que se cuenta es con una imagen g con ruido. Si se conociera la función que agregó ruido a la imagen g, podría aplicarse la función de manera inversa y se obtendría f con un poco menos de dificultad, sin embargo se desconoce la función que le agregó ruido a g, de ahí que una solución a este problema sea la de someter a esta imagen g a cambios, y de esta forma generar un gran número de imágenes $\hat{f}$ distintas. De todo este conjunto de imágenes $\hat{f}$, habrá algunas que se parezcan más a f sobresaliendo del resto, y son precisamente éstas las que nos interesan.

El conjunto finito de imágenes $\hat{f}$, es un conjunto bastante grande, por eso la importancia de utilizar Estimación Bayesiana y así obtener una función que permita estimar o aproximar, la solución que se está buscando. En otras palabras esta función de Estimación Bayesiana debe indicar las imágenes $\hat{f}$ que tienen mayor probabilidad de parecerse, o ser f, de todo el conjunto posible.

3.1. Inferencia Bayesiana

La inferencia Bayesiana *es una inferencia estadística, en la cual se usan evidencia u observaciones para actualizar o inferir nuevamente la probabilidad de que una hipótesis pueda ser verdadera. Usa una estimación numérica del grado de creencia de una hipótesis antes de que la evidencia haya sido observada y calcula una estimación numérica del grado de creencia en la hipótesis después de que la evidencia ha sido observada [15].*

La estimación estadística trata con el problema de inferir conocimiento acerca de parámetros indirectamente observables de los resultados de un experimento relacionado [1]. Es decir, el objetivo del procedimiento de estimación, es recolectar información acerca del valor del parámetro x, dada una observación de los resultados de un experimento y.

En estimación estadística se acostumbra tratar la observación como un vector aleatorio, a menudo justificado por las suposiciones acerca de las medidas de ruido aleatorio, medidas imprecisas del equipo, u otros factores. Se asume que el vector observado tiene una función de densidad de probabilidad perteneciente a una clase indexada por los parámetros $p(x|y)$.

La información que se tiene de una imagen inicial puede ser expresada por medio de una distribución de probabilidad apriori (distribución inicial).

3.2. Teorema de Bayes

El teorema de Bayes se utiliza para conocer la probabilidad a posteriori de cierta variable de interés dado un conjunto de hallazgos.

$$P(H_0|E) = \frac{P(E|H_0)P(H_0)}{P(E)}. \tag{3.1}$$

- H_0 representa una hipótesis, llamada hipótesis vacía, que fue inferida antes de

que una nueva evidencia, E, estuviera disponible.

- $P(H_0)$ es la probabilidad a priori de H_0.
- $P(E|H_0)$ es llamada la probabilidad condicional de ver la evidencia E dado que la hipótesis H_0 es verdadera. También es llamada función de verosimilitud cuando es expresado como una función de H_0 dado E.
- $P(E)$ es llamada probabilidad marginal de E, puede ser calculada como la suma del producto de todas las probabilidades de hipótesis mutuamente exclusivas y probabilidades condicionales correspondientes:

$$\sum_{H_i} P(E|H_i)P(H_i) \tag{3.2}$$

- $P(H_0|E)$ es llamada la probabilidad a posteriori de H_0 dado E.

3.3. Estimador Máximo a Posteriori (MAP)

Para definir el estimador MAP, se introduce la siguiente ecuación

$$\hat{F} = arg_F(\text{mín}\, Q(F)) = arg_F(\text{mín}(1 - P_{F|G}(F))) \tag{3.3}$$

Dicho estimador también puede escribirse como

$$\hat{F} = arg_F(\text{máx}\, P_{F|G}(F)) \tag{3.4}$$

A esta ecuación se le conoce precisamente como el estimador Máximo a Posteriori, que es muy empleado en Procesamiento de Imágenes, con este estimador se obtienen los parámetros más probables, y éstos son los que maximizan la probabilidad a posteriori.

Capítulo 4

Campos Aleatorios Markovianos

Los campos aleatorios proporcionan medidas de probabilidad sobre un dominio de definición que tenga relaciones de tipo espacial o temporal. Para este trabajo lo que importa son las relaciones de tipo espacial. Esto es porque, para poder determinar una función de estimación apropiada, que permita obtener la imagen restaurada f en el modelo de degradación (ver sección 2.2.2), es necesario, que a g, se le someta a cambios. Estos cambios generarán imágenes $\hat{f}$, y como se menciona en el capítulo de Estimación Bayesiana (ver el capítulo 3), es necesario que sólo se consideren como posibles soluciones, a aquellas imágenes que tengan una mayor probabilidad de ser f. Por ello los cambios que se le realicen a g deben tener una justificación de tipo espacial, es decir se deben realizar cambios de valores, a puntos en g, de manera que puedan aumentar la probabilidad de que g converga a f. De esta forma cuando se realice un cambio a un valor de un punto en f, no solo se estará tomando en cuenta el valor en sí, sino que también se consideran los valores de sus vecinos, de acuerdo a las propiedades de los Campos Aleatorios Markovianos.

La Propiedad de Markov *de una secuencia estocástica* $\{X_n\}_{n\geq 0}$*, implica que para todo* $n \geq 1$*,* X_n *es independiente de* $(X_k : k \notin \{n-1, n, n+1\})$*, dado* (X_{n-1}, X_{n+1}). Otra forma de escribir esto es:

$$X_n \perp (X_k : k \in \partial\{n\}). \tag{4.1}$$

Donde $\partial\{n\}$ es el conjunto de vecinos del sitio n.

Campo Aleatorio. *Sea S un conjunto finito, cuyos elementos son denotados por s y llamados sitios, y sea A un conjunto finito llamado espacio de la fase. Un campo aleatorio sobre S, con fases en A, es una colección $X = \{X(s) : s \in S\}$ de variables aleatorias $X(s)$ con valores en A.*

Para el caso de la aplicación de los campos aleatorios markovianos a las imágenes, el conjunto de posiciones donde se define el campo va a denominarse malla o rejilla y se denota por S. S representa el conjunto de píxeles en una estructura matricial en dos dimensiones como puede verse en la figura 4.1:

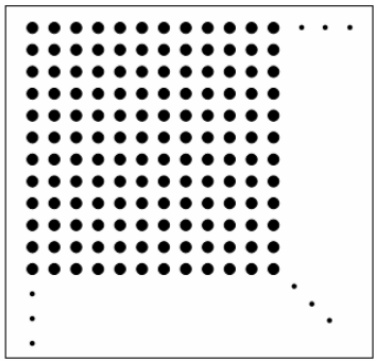


Figura 4.1: Matriz de dos dimensiones

Para cada posición $s \in S$ se define un espacio de estados A_s. Para la imagen s es un píxel y el espacio de estados A_s corresponde a los niveles de gris para cada píxel.

$\Omega = (A_s)_{s \in S}$ es el espacio de todas las configuraciones, es decir, el conjunto de todas las imágenes que se pueden definir en S. A cada configuración $x \in \Omega$ se le asigna una probabilidad $\Pi(x) \geq 0$, tal que $\sum_{x \in \Omega} \Pi(x) = 1$.

Se dice que un campo definido en la rejilla S, con espacio de configuraciones Ω y medida de probabilidad asociada Π, es un campo aleatorio o estocástico si para todo $x \in \Omega$ se cumple que $\Pi(x) > 0$, es decir, si la distribución o medida de probabilidad cumple con la condición de positividad. Por lo tanto, si el modelo de la imagen es un campo aleatorio, significa que todas las imágenes o configuraciones son posibles

(con menor o mayor probabilidad) [4].

4.1. Características locales, vecindarios y cliques

Para un campo aleatorio se puede definir un tipo de probabilidad condicionada denominada característica local del campo definida para $A \subset S$ como:

$$\Pi(X_A = x_A / X_{S\setminus A} = x_{S\setminus A}). \tag{4.2}$$

Las características locales siempre están definidas gracias a la propiedad de positividad de los campos aleatorios. Dicha propiedad, nos dice que cualquier subimagen tendrá una probabilidad con valor positivo, que depende del resto de la imagen.

Las dependencias en S son, en general locales. Es decir, que en una imagen un píxel va a depender de los píxeles cercanos. Por ello se define para cada posición $s \in S$ un conjunto $\partial(s) \subset S$, que corresponde a las posiciones de S de las que s depende. Los elementos de $\partial(s)$ se denominan vecinos de s. La colección de conjuntos $\partial = \{\partial(s) : s \in S\}$ se denomina sistema vecindario o vecindario de la cuadrícula S. Un sistema vecindario debe cumplir dos propiedades:

- Que s no sea vecino de sí mismo $s \notin \partial(s)$.
- Que si s es vecino de t, este último lo sea del primero $s \in \partial(t) \Leftrightarrow t \in \partial(s)$.

En general, los sistemas de vecindades son homogéneos. Es decir, que al conocer los vecinos de una posición s se puede saber cuales son los vecinos de otra posición t, sin más que desplazar a t el sistema de vecinos de s.

Sea $s = (i, j)$ y $t = (k, l)$, dos píxeles de la imagen tal que $t \in \partial(s)$, se define el orden c del vecindario como el menor entero que cumpla:

$$c \geq (k - i)^2 + (l - j)^2. \tag{4.3}$$

Para todos los vecinos $t = (k, l)$ de $s = (i, j)$. La imagen 4.2 muestra vecindarios de los órdenes más comunes. El punto negro representa el píxel s y el punto blanco cada uno de los píxeles t vecinos de s.

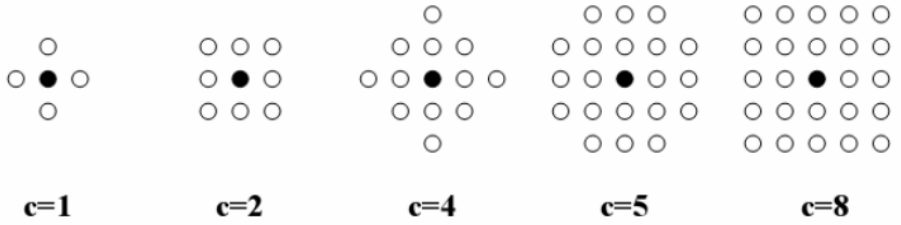


Figura 4.2: Figura que muestra vecindarios de órdenes 1, 2, 4, 5 y 8, respectivamente, en una imagen.

Puesto que la imagen definida en la rejilla S tiene dimensiones finitas, los vecindarios de los píxeles cerca del borde no pueden ser iguales que los vecindarios de los píxeles interiores. El concepto de homogeneidad no se puede cumplir para los píxeles de borde, sin embargo, esto siempre ocurre y se continua diciendo que el vecindario es homogéneo suponiendo implícitamente el efecto de los bordes.

Para el modelo de contorno se define el orden del vecindario como el menor entero que cumpla:

$$c \geq |s - t|. \tag{4.4}$$

Para todos los vecinos t de s, tal y como se muestra en la Figura 4.3.

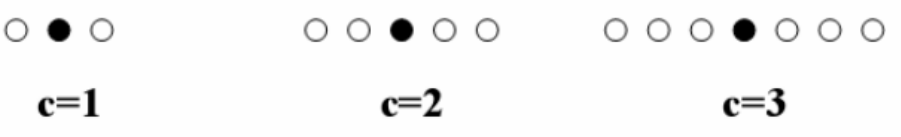


Figura 4.3: Orden del vecindario de un contorno.

Dado un sistema de vecindad ∂ en S, se dice que un subconjunto $C \subset S$ es un clique, si dos elementos cualesquiera de C (diferentes entre sí) son vecinos. Existen varios tipos de cliques, según la relación espacial entre los elementos que lo forman (ver la figura 4.4).

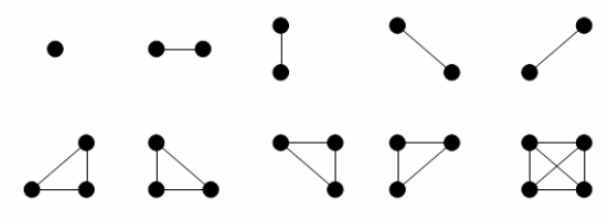


Figura 4.4: Figura que muestra los tipos de cliques para un vecindario de orden 2.

Se dice que un campo aleatorio es de Markov o MRF(Markov Random Field, por sus siglas en inglés) con respecto al vecindario ∂ si para todo $x \in \Omega$:

$$\Pi(X_s = x_s / X_r = x_r, r \neq s) = \Pi(X_s = x_s / X_r = x_r, r \in \partial(s)). \tag{4.5}$$

con $s, r \in S$.

Capítulo 5

Interpolación

Interpolación. *En el subcampo matemático del análisis numérico se denomina interpolación a la construcción de nuevos puntos dados partiendo del conocimiento de un conjunto de puntos dados discretos. En el campo de la fotografía se aplica este mismo patrón para obtener un tamaño mayor de una imagen inicial, rellenando la información faltante con datos "inventados" a partir de un algoritmo o método específico [15].*

Existen diversos tipos de métodos para realizar la interpolación. Estos métodos pueden agruparse principalmente en dos categorías, los adaptativos y los no adaptativos. Los adaptativos cambian dependiendo sobre que es lo que están interpolando (bordes definidos, texturas suaves o lisas, entre otros), mientras que los métodos no adaptativos tratan a todos los píxeles igual.

Los diferentes métodos básicamente se diferencían en la rapidez y calidad con la que producen a la nueva imagen. Algunos de estos métodos se listan a continuación.

5.1. Interpolación del vecino más próximo.

Este es el método más sencillo y básicamente lo que realiza es repetir la información con la que se cuenta, en la nueva imagen. Es decir, escala la imagen al nuevo tamaño, colocando la información disponible proporcionalmente, y rellena los píxeles

desconocidos de acuerdo a su vecino conocido más cercano (ver Figura 5.1).

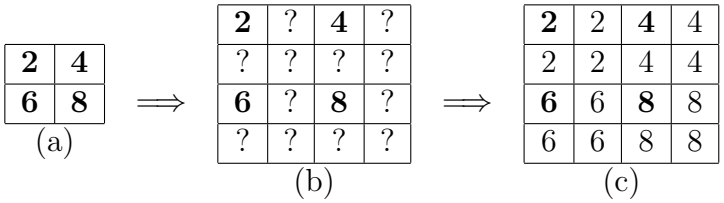


Figura 5.1: Interpolación del vecino más próximo.

En la Figura 5.1(a) se muestra la imagen original de 2×2; en la Figura 5.1(b), aparece la nueva imagen de 4×4, en donde los signos ?, indican los valores desconocidos que resultan al ampliar la imagen, y en la Figura 5.1(c) se tiene la nueva imagen después de interpolar.

5.2. Interpolación bilineal.

La interpolación bilineal al igual que el método del vecino más cercano, escala la imagen al nuevo tamaño, colocando la información disponible proporcionalmente, pero a diferencia del anterior, determina los valores de los nuevos píxeles de acuerdo al promedio pesado de los 4 píxeles que se encuentran en su cercanía.

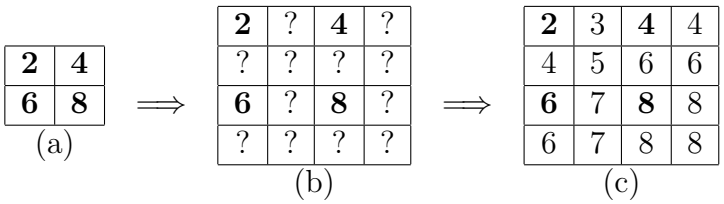
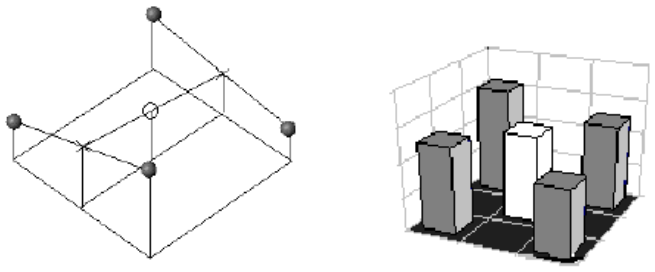


Figura 5.2: Interpolación bilineal.

En la Figura 5.2(a) se muestra la imagen original de 2×2; en la Figura 5.2(b), aparece la nueva imagen de 4×4, en donde los signos ?, indican los valores desconocidos que resultan al ampliar la imagen, y en la Figura 5.2(c) se tiene la nueva imagen

después de interpolar. Se puede apreciar que el valor del píxel que se encuentra en $(i, j) = (1, 2) = 3 = \frac{2+4}{2}$ en cambio para $(i, j) = (2, 2) = 5 = \frac{2+4+6+8}{4}$, dado que este píxel tiene 4 vecinos conocidos.

Gráficamente esta interpolación se vería como se muestra en la Figura 5.3.



(a) (b)

Figura 5.3: Interpolación bilineal.

La Figura 5.3 muestra cómo es afectado el valor del píxel del centro (columna blanca en 5.3(b)) ya que es el promedio de los cuatro píxeles adyacentes conocidos (columnas grises en 5.3(b)), que es la idea principal en la interpolación bilineal.

5.3. Interpolación bicúbica

Esta interpolación es muy parecida a la interpolación bilineal, a diferencia de que en la interpolación bicúbica se toman en consideración a 16 píxeles vecinos conocidos para obtener el valor del nuevo píxel, es decir una vecindad de 4×4, a diferencia de la bilineal que solo toma en consideración una vecindad de 2×2 píxeles conocidos.

En la Figura 5.4 se observa cómo para obtener el píxel desconocido (columna blanca), es necesario considerar a sus 16 vecinos conocidos más cercanos, que representan una matriz de 4×4 en la imagen conocida. En la figura se manejan distintos tonos de gris para los valores conocidos, debido a que los píxeles más cercanos tienen un mayor peso en los cálculos.

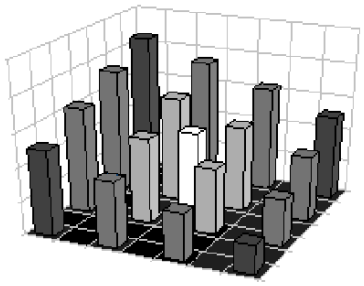


Figura 5.4: Interpolación bicúbica.

Capítulo 6

Programación no lineal sin restricciones

Hasta este momento, en los capítulos 3 y 4 se ha mencionado la necesidad de determinar una función que permita obtener la imagen restaurada $\hat{f}$ en el modelo de degradación (ver sección 2.2.2). Esta función debe ser una función real no lineal que identificamos con el nombre de función objetivo o función costo, la cual permitirá plantear un problema de optimización.

Optimización *se refiere al estudio de problemas en los cuales se busca minimizar o maximizar una función real, escogiendo sistemáticamente los valores de las variables enteras o reales desde un conjunto permitido [15].*

La optimización para problemas no lineales es importante cuando se requiere la caracterización y localización de máximos y mínimos en funciones no lineales. Existen diversos algoritmos de optimización, y dependiendo del problema en cuestión, existen métodos que son más apropiados para su solución que otros. Por ello es necesario reconocer las características del problema (las características matemáticas de la función objetivo, las restricciones y las variables de control), identificando la técnica más apropiada de solución mediante la programación no lineal sin restricciones.

6.1. Definición

Todos los métodos de programación no lineal sin restricciones tratan de resolver el problema:

$$\min_{x \in R^n} f : R^n \to R. \tag{6.1}$$

La f que se desea minimizar es una función de valores reales, y es llamada función objetivo, o función de costo, donde f es al menos dos veces diferenciable.

Minimizador local. *Sea la función $f : R^n \to R$ y Ω un subconjunto de R^n. Un punto $x^* \in \Omega$ es un minimizador local de f en Ω si existe $\varepsilon > 0$ tal que $f(x) \geq f(x^*)$ para todo $x \in \Omega \backslash \{x^*\}$ y $||x - x^*|| < \varepsilon$. En caso de que se cumpla $f(x) > f(x^*)$ entonces se dice que el punto x^* es un minimizador local estricto.*

Minimizador global. *Sea la función $f : R^n \to R$ y Ω un subconjunto de R^n. Un punto $x^* \in \Omega$ es un minimizador global de f sobre Ω si $f(x) \geq f(x^*)$ para todo $x \in \Omega \backslash \{x^*\}$ [2] (ver Figura 6.1).*

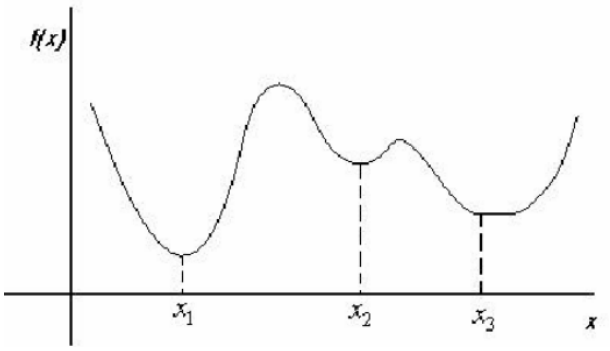


Figura 6.1: Figura donde se muestran los tipos de minimizadores. x_1 es un minimizador global, x_2 es un minimizador local estricto, x_3 es un minimizador local no estricto.

En la mayoría de los casos se tienen funciones objetivo cuyos valores se incrementan localmente en todas direcciones, y esa es la definición de un mínimo local estricto, sin embargo existen algunos puntos en donde los valores de la función permanecen localmente en la misma dirección, pero se incrementan en algunos otros

puntos, eso es lo que se le conoce como un mínimo local débil. En general existen funciones en las cuales están presentes ambos tipos de mínimos, a los que se les nombra simplemente mínimo local.

La función continua $f : R^n \rightarrow R$ se dice que es continuamente diferenciable en $x \in R^n$, si existe en una vecindad de x, y es continua en x, con i = 1,2,. . . ,n. En este caso el gradiente de f en x está definido como:

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}(x), ..., \frac{\partial f}{\partial x_n}(x)\right]^T \tag{6.2}$$

Para comprobar si un punto dado es mínimo o no es importante tener condiciones las cuales sean declaradas de manera algebraica. Estas condiciones pueden ser:

1. **Condición necesaria de primer orden.**

   Si x^* es un mínimo local y f es continuamente diferenciable en una vecindad abierta de x^*, entonces $\nabla f(x^*) = 0$.

2. **Condición necesaria de segundo orden.**

   Si x^* es un mínimo local de $f(x)$, entonces $\nabla^2 f(x^*)$ es semidefinida positiva.

3. **Condición suficiente de segundo orden.**

   Si $\nabla^2 f(x^*)$ es definida positiva, entonces x^* es un mínimo local.

Los métodos de solución de la programación no lineal se pueden clasificar de manera general en algoritmos directos e indirectos. Como ejemplo de los métodos directos están los algoritmos de gradiente, donde se busca el máximo (o el mínimo) de un problema siguiendo la mayor tasa de aumento (o disminución) de la función objetivo. En los métodos indirectos, como su nombre lo indica, no pueden resolverse directamente, sino que debe plantearse como otro problema a través del cual se determina el óptimo. Como ejemplos de estos casos están la programación cuadrática, la programación separable y la programación estocástica [13].

A continuación se describen algunos de los métodos de programación no lineal sin restricciones que fueron empleados en este trabajo.

6.2. Métodos Iterativos

Para solucionar el problema de minimizar una función objetivo arbitraria $f(x)$ de n variables independientes sin restricciones, se modela el problema como un sistema de ecuaciones lineales de la forma 6.3.

$$Ax = b \tag{6.3}$$

Donde:

- x es un vector desconocido.
- b es un vector conocido.
- A es una matriz cuadrada (ver Figura 6.2).

Este sistema de ecuaciones debe tener valores absolutos en cada fila y columna. El proceso que se utiliza para resolver el sistema de ecuaciones es iterativo, y se realiza hasta que éste converge [11].

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Figura 6.2: Representación de la ecuación 6.3.

Todos los métodos que se describen a continuación resuelven este sistema de ecuaciones, y son utilizados para comparar las soluciones del método Multigrid descrito en el capítulo 7.

6.2.1. Método de Jacobi

El método de Jacobi, por ser un método iterativo, es utilizado para resolver sistemas de ecuaciones de la forma 6.3. De esta forma considerando la i-ésima ecuación de $Ax = b$ se tiene 6.4.

$$\sum_{j=1}^{n} A_{ij} x_j = b_i \tag{6.4}$$

Resolviendo para x_i asumiendo que todos los demás valores son constantes se obtiene 6.5.

$$x_i = \frac{1}{A_{ii}} \left(b_i - \sum_{j \neq i} A_{ij} x_j \right) \tag{6.5}$$

En sí 6.5, es el método de Jacobi. De forma general puede ser visto en la k-ésima ecuación quedando como en 6.6.

$$x_i^{(k)} = \frac{1}{A_{ii}} \left(b_i - \sum_{j \neq i} A_{ij} x_j^{(k-1)} \right) \tag{6.6}$$

El algoritmo se encuentra descrito en el apéndice B.

6.2.2. Método de Gauss Seidel

Este es un método que al igual que el de Jacobi es utilizado para resolver la ecuación $Ax = b$.

Dicho método resuelve una ecuación a la vez en secuencia, y utiliza resultados previamente calculados tan pronto como éstos están disponibles [8]. Por ello la ecuación k-ésima se representa como en 6.7.

$$x_i^{(k)} = \frac{b_i - \sum_{j<i} A_{ij} x_j^{(k)} - \sum_{j>i} A_{ij} x_j^{(k-1)}}{A_{ii}} \tag{6.7}$$

Hay dos características importantes de este método que hay que recalcar. Primero el algoritmo parece secuencial, ya que cada nuevo componente de la nueva iteración

depende de todos los componentes previamente calculados, las actualizaciones no pueden realizarse simultáneamente como en el método de Jacobi. Segundo la nueva iteración $x^{(k)}$ depende del orden en el cual las ecuaciones son examinadas. Si se cambia este orden, los componentes de las nuevas iteraciones (y no solo su orden) también cambiarán. El algoritmo se encuentra descrito en el apéndice B.

6.2.3. Método de Gradiente

Existen diversos métodos de Gradiente, éstos al igual que los métodos Jacobi y Gauss Seidel, solucionan el sistema de ecuaciones $Ax = b$. Sin embargo los métodos de gradiente sólo lo pueden solucionar si la función objetivo es doblemente diferenciable. Esto significa, que la función objetivo f debe ser al menos de segundo grado. Para nuestro caso, la función objetivo cumple con dicha característica. De esta forma, que si se representa gráficamente a dicha función se observaría como la Figura 6.3.

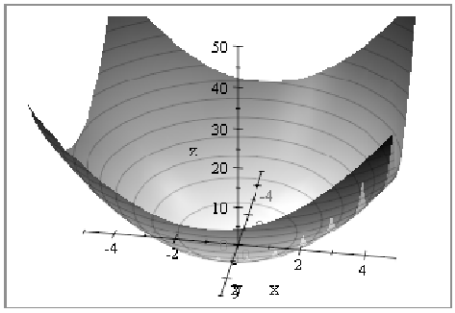


Figura 6.3: Función f de segundo grado, donde el punto mínimo de esta superficie es la solución de $Ax = b$.

A diferencia de los algoritmos anteriores, los métodos de Gradiente, realizan pequeños pasos, acercándose al punto mínimo de la función, efectuándo un cambio de dirección en cada uno de ellos. De manera que este proceso, puede expresarse mediante la ecuación 6.8.

$$x_{k+1} = x_k + \alpha_k d_k \tag{6.8}$$

Donde x_{k+1} es la solución en la k-ésima iteración, x_k es la solución en la iteración anterior, d_k es la dirección en la iteración actual, y α_k es un factor que se obtiene a partir de una búsqueda lineal.

En la Figura 6.4 se puede apreciar que el tamaño del paso se vuelve cada vez más pequeño hasta acercarse al mínimo. También puede observarse que la forma en que se aproxima es zig-zag.

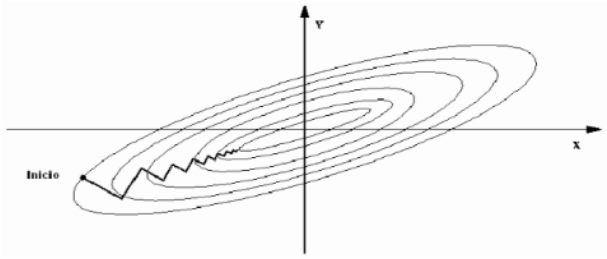


Figura 6.4: Figura que muestra la convergencia del método de Gradiente.

Existen diversos métodos de Gradiente, de los cuales se describe a continuación el método de máximo descenso, y el método de Gradiente conjugado.

Método de máximo descenso

El método de máximo descenso, conocido también como método de descenso de Gradiente, es un algoritmo de Gradiente para hallar el mínimo local más cercano de una función f. Este método inicia en un punto x_0, y tantas veces como se necesite se mueve de x_i a x_{i+1}, minimizando el gradiente en dirección del descenso.

$$d_k = -\nabla f(x_k) \tag{6.9}$$

La dirección del gradiente negativo 6.9 satisface el criterio de descenso, es decir converge, si $\nabla f(x_k) \neq 0$, siendo esto a lo que se le llama dirección de descenso en máxima escalada, y se emplea en la iteración k-ésima de la ecuación 6.8, para dar un incremento al método de descenso de gradiente.

El procedimiento para poder hallar el valor para $f(x)$ óptimo se puede ver por medio del algoritmo descrito en el apéndice B.

Método de Gradiente Conjugado

Al igual que el algoritmo de máximo descenso, el método de Gradiente conjugado es un algoritmo para hallar el mínimo local más cercano de una función de n variables donde se supone que el gradiente puede ser aplicado. Usa direcciones conjugadas en vez del gradiente local para su descenso. Si la vecindad del mínimo tiene la forma de un valle largo y estrecho, éste se alcanza en menos pasos que con el método del máximo descenso.

Es apropiado utilizar el método de Gradiente conjugado con matrices esparcidas. Si A es densa, el mejor curso de acción es probablemente factorizar a A y resolver la ecuación por sustitución hacia atrás. El tiempo utilizado en factorizar una matriz densa A, es apenas equivalente al tiempo utilizado para resolver el sistema iterativamente, y una vez que A ha sido factorizada el sistema puede ser resuelto rápidamente por sustitución hacia atrás para múltiples valores de b.

Para este método la primera iteración es la misma que para el método de descenso de Gradiente. Las iteraciones son de la forma:

- $d_k = -g_k$ para $k = 1$.
- $d_k = -g_k + \beta d_{k-1}$ para $k \geq 2$.
- $x_{k+1} = x_k + \alpha_k d_k$.

Donde β_k es un escalar, y α_k es el tamaño de paso obtenido mediante una búsqueda unidimensional.

Para el cálculo de β_k se puede utilizar una de las fórmulas siguientes [11]:

- Fletcher-Reeves (FR)

$$\beta_k^{FR} = \frac{||g_k||^2}{||g_{k-1}||^2} \tag{6.10}$$

- Polak-Ribiere (PR)

$$\beta_k^{PR} = \frac{< g_k, g_k - g_{k-1} >}{||g_{k-1}||^2} \tag{6.11}$$

- Esteness-Stiefel (ES)

$$\beta_k^{ES} = \frac{< g_k, g_k - g_{k-1} >}{< d_{k-1}, g_k - g_{k-1} >} \tag{6.12}$$

- Polak-Ribiere positivo (PR+)

$$\beta_k^{PR+} = \text{máx}\{\beta_k^{PR}, 0\} \tag{6.13}$$

Donde $< ., . >$ es el producto escalar usual y $||..||$ es la norma euclidiana.

Capítulo 7

Método Multigrid

El objetivo de esta tesis, es solucionar el problema de restauración digital. En si, se desea restaurar una imagen que ha sido degradada utilizando conocimiento apriori, partiendo del hecho de que este conocimiento no sea utilizado en el proceso de restauración. Es decir con una imagen degradada g, se desea obtener a la imagen original f (ver modelo de degradación, sección 2.2.2), mediante una función de estimación que no involucra conocimiento apriori, y que ha sido obtenida utilizando Estimación Bayesiana y Campos Aleatorios Markovianos (ver capítulos 3 y 4). Matemáticamente este proceso, se realiza de la siguiente manera [12, 7]:

Se parte del modelo de degradación, de acuerdo a la ecuación 7.1.

$$g(x, y) = H[f(x, y)] + n(x, y) \tag{7.1}$$

Despejando $n(x, y)$ de la ecuación 7.1 se obtiene la ecuación 7.2.

$$nr = gr - H[fr] \tag{7.2}$$

Donde $r = (x, y)$ significa que r, representa al píxel con coordenadas x, y en la imagen.

Si se asume que nr es ruido con una distribución gaussiana, con media cero y varianza s^2, la distribución condicional de gr, dado fr está dada por la ecuación 7.3.

$$P_{g|f}(f) = \frac{1}{K}\left(exp\left[-\sum_{r}(Hfr - gr)^2/2s^2\right]\right) \tag{7.3}$$

Donde K es una constante.

El conocimiento apriori de f puede ser modelado como un Campo Aleatorio Markoviano [7], lo cual significa que la distribución de probabilidad apriori para la imagen f está dada por:

$$P_f(f) = \frac{1}{Z}exp\left[-\sum_{C}V_C(f)\right] \tag{7.4}$$

Donde V_C son las funciones de potencial asociadas a cada clique C.

Entonces la distribución a posteriori de f es obtenida a partir de la regla de Bayes como:

$$P_{f|g}(f) = \frac{1}{Z_P}exp\left[-U(f)\right] \tag{7.5}$$

Con:

$$U(f) = \frac{1}{2s^2}\sum_{r}(Hfr - gr)^2 + \sum_{C}V_C(f) \tag{7.6}$$

El estimador MAP de la ecuación 7.5, se obtiene al minimizar 7.6.

En este trabajo de tesis, se asume que $Hfr = fr$ para toda r, y que las funciones de potencial V_C son de la forma:

$$V_C(f) = V_{<r,s>}(f) = (fr - fs)^2 \tag{7.7}$$

Entonces finalmente la ecuación a minimizar 7.6 queda de la forma:

$$U(f) = \sum_{r}(fr - gr)^2 + \lambda\sum_{<r,s>}(fr - fs)^2 \tag{7.8}$$

Donde:

$r = (x, y)$ Significa que r, representa al píxel con coordenadas x, y en la imagen.

$\lambda = 2s^2$, el cual es un parámetro positivo, que hace el efecto de "suavizado" en la imagen.

$< r, s >$ indica que la segunda sumatoria se lleva a cabo sobre los vecinos de r, de acuerdo al sistema de vecindad s que se esté utilizando.

$g(r)$ representa a la imagen original.

$f(r)$ es la imagen que se desea encontrar.

Así, para solucionar el problema planteado, es necesario minimizar la función de costo 7.8 mediante algún algoritmo conocido. Existen diversos algoritmos que pueden realizarlo, por ejemplo los mostrados en el capítulo 6. Éstos tienen como característica que pueden resolver un sistema de ecuaciones de la forma $Ax = b$, que es lo que se necesita para este caso.

Expresado matemáticamente para minimizar $U(f)$ es necesario realizar derivadas parciales con respecto a cada una de las variables (en este caso cada uno de los píxeles), e igualando a cero, y despejando, se obtiene la ecuación 7.9.

$$\min U(f).$$

$$\frac{\partial U(f)}{\partial fr} = (fr - gr) + \lambda \sum_{fs \in Nr} (fr - fs) = 0.$$

$$fr\,(1 + \lambda |Nr|) - \lambda \sum_{s \in Nr} fs = gr.$$

$$fr^{(t+1)} = \frac{gr + \lambda \sum\limits_{s \in Nr} fs^{(t)}}{1 + \lambda |Nr|}. \tag{7.9}$$

Donde:

Nr es el conjunto de los píxeles que forman la vecindad del píxel r.

$|Nr|$ indica el número de píxeles que conforman el conjunto Nr.

Se puede observar que en cada paso $(t + 1)$, se obtiene una fr, esto significa que en cada iteración se encuentra una nueva solución que se espera sea mejor que la

anterior. Para determinar cuando terminar de generar nuevas soluciones y obtener así un resultado en específico se pueden seguir dos enfoques. Uno de ellos es el de definir exactamente el número de pasos o iteraciones que se deseen desde el principio, y de esta forma observar cual ha sido el resultado después de n iteraciones. Otro es el de definir un criterio de paro en base al error que hay entre la nueva solución y la anterior, y compararlo con un factor de error deseado como se muestra en la ecuación 7.10.

$$||fr^{(t+1)} - fr^{(t)}|| < \varepsilon. \tag{7.10}$$

Entre más pequeño sea el valor de ε, significa que queremos que la diferencia de costos tienda a cero. Sin embargo un algoritmo puede tardar demasiado para lograr que suceda esto.

Para este trabajo de tesis, se ha definido un sistema de vecindad de 8, por lo cual, para un píxel en la posición (i, j) la ecuación 7.9 está dada por la ecuación 7.11.

$$f_{i,j}^{(t+1)} = \frac{g_{i,j} + \lambda \begin{pmatrix} f_{i,j-1}^{(t)} + f_{i,j+1}^{(t)} + f_{i-1,j}^{(t)} + f_{i+1,j}^{(t)} + \\ f_{i-1,j-1}^{(t)} + f_{i-1,j+1}^{(t)} + f_{i+1,j-1}^{(t)} + f_{i+1,j+1}^{(t)} \end{pmatrix}}{1 + 8\lambda}. \tag{7.11}$$

Esta ecuación puede ser resuelta por los algoritmos Jacobi, Gauss Seidel, o el de Gradiente, descritos en el capítulo 6. Al implementar cualquiera de éstos, se tiene que recorrer toda la imagen visitando a cada uno de los píxeles en ella, y cuando un píxel r es visitado el valor de $f(r)$ es reemplazado con el valor que se obtiene al resolver la ecuación 7.11. Sin embargo de esta forma los elementos de la imagen con frecuencias bajas toman mucho tiempo en ser eliminados, de ahí que la velocidad de convergencia es considerablemente lenta [12]. Para solucionar este inconveniente, es decir, acelerar la velocidad de convergencia, se puede aplicar un algoritmo conocido como método Multigrid.

Los métodos Multigrid *en análisis numérico son un grupo de algoritmos para resolver ecuaciones diferenciales utilizando básicamente una jerarquía de discretizaciones. Las aplicaciones típicas de este método son las soluciones de ecuaciones diferenciales parciales elípticas. Este método puede resolver tanto problemas lineales como no lineales.*

Este método, al igual que los anteriores, puede resolver ecuaciones de la forma $Ax = b$, pero a diferencia de ellos, utiliza una estrategia totalmente distinta resolviendo el sistema mucho más rápidamente, de ahí que se convierta en el principal objeto de estudio en este trabajo de tesis.

7.1. Descripción del método

La estrategia del método Multigrid se basa en el hecho de que la convergencia será más rápida si la solución inicial fr^0 es cercana a la solución deseada fr. Para encontrar dicha solución inicial se define un nuevo problema: hallar la solución de $A_2 f_2 = g_2$,, en donde A_2, f_2, y g_2 tienen la mitad de píxeles que el problema original, en este punto g_2 se obtiene al tomar una muestra de la imagen observada (malla2), ver figura 7.1.

Se supone que la solución del problema $A_2 f_2 = g_2$, es más fácil, dado que la matriz o malla es más pequeña, pero si no es el caso, este problema puede resolverse aplicando el mismo razonamiento recursivamente, es decir para solucionar $A_2 f_2 = g_2$, hay que definir un nuevo problema $A_3 f_3 = g_3$, que tiene la mitad de píxeles que el problema anterior, y por lo tanto debe ser más fácil resolverlo.

Lo anterior conduce a la siguiente pregunta: ¿Cómo es que la solución de una matriz más pequeña contribuye a la solución de una más grande?, y la respuesta es que se puede obtener una fr^0 muy próxima a la que se desea al interpolar la solución de la malla inmediata más pequeña. Es decir, tomando en cuenta la Fi-

gura 7.1, se puede ver que para solucionar la imagen con ruido hay que solucionar su representación en la malla1, y a su vez para solucionar la malla1, es necesario encontrar una fr^0_{malla1} que sea muy próxima a la fr_{malla1}, y al estar en esta etapa se le aplica algún algoritmo conocido como el de Gauss Seidel, y su convergencia sea mucho más rápida.

Sin embargo para hallar fr^0_{malla1} hay que solucionar el problema de la malla2 e interpolar dicho resultado obteniendo así a fr^0_{malla1}, claro que para resolver la malla2 hay que utilizar el mismo procedimiento recursivamente, hasta que se tenga una malla sumamente pequeña cuya solución no represente un costo excesivo al utilizar el método de Gauss Seidel de forma natural.

En general se definen M mallas decrecientes, donde al resolver el sistema $A_M f_M = g_M$ con el método de Gauss Seidel utilizando M iteraciones, se utiliza la solución interpolada f_M como el estado inicial para el problema correspondiente en la malla $M-1$, en donde, de igual forma una vez teniendo interpolada la solución inicial, hay que aplicar el algoritmo de Gauss Seidel con un número de iteraciones igual a $M-1$ en esta malla, así hasta que la solución en la malla1 es alcanzada aplicando finalmente una sola iteración del algoritmo Gauss Seidel en esta malla.

Dado que cada malla es reducida a la mitad en tamaño es necesario que las

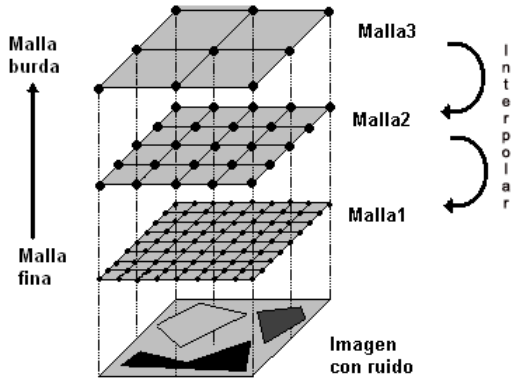


Figura 7.1: Mallas utilizadas en el esquema Multigrid. Se puede ver la diferencia de píxeles entre cada malla, aún cuando el modelo de la superficie es el mismo.

imágenes sean cuadradas y de un tamaño 2^n, donde $n > 0$.

En el método Multigrid cuando se regresa de mallas burdas a mallas finas es necesario realizar una interpolación, dado que en la malla burda se tienen menos píxeles que en la malla fina, y se desea que la malla fina sea lo más cercana a fr. Para ello existen diversos métodos de interpolación que se podrían utilizar, en este caso se ha elegido utilizar el método de interpolación bilineal, que nos ofrece una mejor calidad en la imagen que la interpolación de vecino más próximo, y es un poco menos costosa que la interpolación bicúbica. En el capítulo 5 se detallan dichos algoritmos.

7.2. Algoritmo

Multigrid recursivo(fr, gr)

Si el tamaño de fr *es igual a* 2×2

Resolver con $fr \leftarrow$ *metodo_Gauss_Seidel*(fr, gr)

Regresar

$burdafr \leftarrow reduce\_mitad(fr)$

$burdagr \leftarrow reduce\_mitad(gr)$

Aplicar Multigrid recursivo $(burdafr, burdagr)$

$fr \leftarrow$ *interpolar*$(burdafr)$

Resolver $fr \leftarrow$ *metodo_Gauss_Seidel*(fr, gr)

Regresar

Las funciones principales de la implementación de este algoritmo puede verse en el Apéndice C.

Capítulo 8

Pruebas

En esta parte se realizan pruebas con el software "ImageRestore" desarrollado en esta tesis. Estas pruebas son realizadas con distintas imágenes, y permiten comparar los resultados de aplicar los algoritmos Multigrid, Jacobi, Gauss Seidel y, Gradiente a cada una de ellas. Estas comparaciones son realizadas en tiempo y función de costo (Ecuación 7.8). Al realizar las pruebas se disponía de las imágenes originales (imágenes sin ruido), y a éstas se les añadió un porcentaje de ruido, que varía según la imagen, con la finalidad de apreciar que tan buenos son los métodos de restauración. El ruido se les agregó mediante el programa de Edición de Imágenes "Photoshop". Los ruidos que se utilizaron son: Gaussiano, Uniforme y Sal y Pimienta (ver apéndice A).

La justificación de utilizar este tipo de ruido radica en que para este trabajo de tesis la información apriori no se toma en cuenta durante la solución del problema, y es precisamente en este tipo de casos en los que el ruido $n(x, y)$ es conveniente modelarlo mediante ruido Gaussiano [14].

Para las pruebas se utilizaron imágenes en escala de grises, con extensión bmp. Este formato permite tener la imagen en RGB. Para este trabajo se mantuvieron las tripletas RGB, pero el valor para R, G y B es el mismo, lo que da la apariencia

de escala de grises a la imagen, de ahí que los algoritmos siempre trabajan con un único valor.

También es importante mencionar que para realizar las pruebas se utilizó una computadora de escritorio SONY VAIO con las siguientes características:

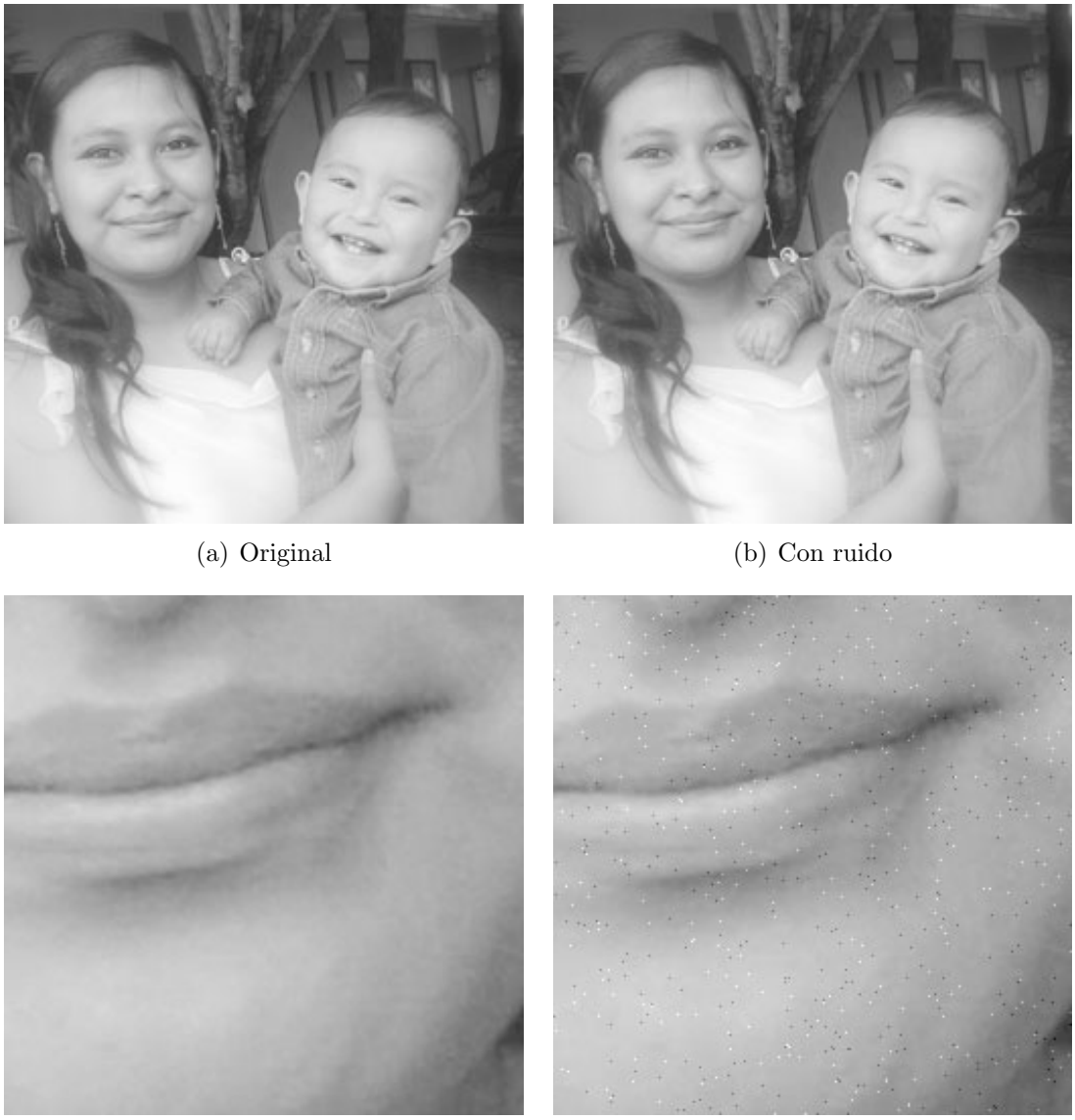
- Procesador AMD Athlon XP 2600+ (2.13 GHz).
- 512 MB en memoria RAM.
- Sistema Operativo: Windows XP Profesional.

Dado que todas las imágenes que se utilizaron en estas pruebas son demasiado grandes, una vez que se finalizó de trabajar con ellas y sólo para que tuviesen una mejor presentación en este trabajo de tesis, se utilizó un programa de edición de imágenes para hacerlas más pequeñas mediante el método de interpolación bicúbica, por ello se muestra a cada imagen y aparte se muestra una ampliación de la misma para poder observar los detalles que pasan desapercibidos a simple vista.

8.1. Imagen "SyG"

"SyG", Figura 8.1(a), es una imagen con una resolución de 2048×2048 píxeles, que se le aplicó el ruido "Sal y Pimienta" con una densidad del 10 %, produciendo la imagen "SyG con ruido", Figura 8.1(b). Como puede notarse no se aprecian diferencias en las imágenes completas, de ahí que a continuación solo se presentarán partes de éstas aumentadas al 100 %, como las mostradas en las Figuras 8.1(c) y 8.1(d).

Los resultados de las pruebas aplicadas a esta imagen utilizando $\lambda = 1.5$ son los mostrados en la Tabla 8.1. En dicha tabla, se muestra el nombre del algoritmo, el número de iteraciones que se ejecutaron, el tiempo que tardó en realizar dicho número de ejecuciones, y el valor de la función de costo resultante al final del número de



(a) Original (b) Con ruido

(c) Original 100 % (d) Con ruido 100 %

Figura 8.1: Imagen SyG.

iteraciones especificado. En esta tabla puede observarse claramente que el algoritmo Multigrid además de obtener el menor valor en la evaluación de la función de costo también lo hace en el menor tiempo, ya que ni uno de los otros algoritmos con su máximo de iteraciones computadas se acercó a este valor.

Estas no son las únicas comparaciones que pueden realizarse, aunque se aprecia que no es posible comparar los algoritmos por número de iteraciones, porque el algoritmo Multigrid no tiene iteraciones como las de los otros métodos, sino que

Algoritmo	**No. Iteraciones**	**Tiempo(seg)**	**Costo**
Multigrid	1	2.65625	55225637519.9274
GS	1	1.15625	55915129348.6217
GS	2	2.39062	55830632788.6280
GS	5	5.42187	55785701898.8974
GS	20	21.79687	55762109652.8415
GS	50	53.78125	55760987022.9878
GS	100	110.21875	55760980602.6855
Jacobi	1	1.31250	55954404631.9806
Jacobi	2	2.64062	55877162181.1571
Jacobi	5	6.48437	55808877414.4255
Jacobi	20	25.62500	55768138627.6515
Jacobi	50	63.62500	55761390062.6604
Jacobi	100	126.76562	55760985857.3704
Gradiente	1	1.76562	56207905641.3041
Gradiente	2	3.56250	56085664076.0080
Gradiente	5	8.82812	55822079281.8880
Gradiente	20	34.21875	55766014464.3195
Gradiente	50	86.39062	55761095450.2260
Gradiente	100	171.06250	55760981001.5307

Tabla 8.1: Resultados de la imagen "SyG con Ruido", utilizando $\lambda = 1.5$.

realiza distintas iteraciones dependiendo en que capa se encuentre, realizando una única iteración en su capa más fina, y en definitiva ésta no puede considerarse como una forma de comparación, por ello lo que se hizo fue tomar como base el tiempo que tarda el algoritmo Multigrid en dar su solución y de ahí tomar "instantáneas" de los otros algoritmos en un tiempo aproximado a éste y comparar los valores de la función de costo de cada uno, mostrando su imagen solución correspondiente (ver Tabla 8.2).

Las imágenes mostradas en la Figura 8.2 son las que se obtuvieron en el instante marcado en la Tabla 8.2, para cada uno de los algoritmos. Se puede ver que en algunas de ellas, como en la 8.2(c) y 8.2(d), persisten puntos de ruido que aún no

Algoritmo	Tiempo(seg)	Costo
Multigrid	2.65625	55225637519.9274
Gauss Seidel	2.39062	55830632788.6280
Jacobi	2.64062	55877162181.1571
Gradiente	3.56250	56085664076.0080

Tabla 8.2: Resultado de evaluar la función de costo de cada uno de los algoritmos, en un tiempo aproximado, al tiempo que tardó el algoritmo Multigrid.

han sido eliminados.

El factor λ (lambda) que se utilice hace la función de "suavizado" de la imagen, por ejemplo si se utiliza un valor para λ muy pequeño cualquiera de los algoritmos va a dar como resultado una imagen no muy suavizada que se parecerá más a la imagen con ruido, pero si se utilizan valores un poco más grandes se pueden obtener resultados muy agradables a la vista, aunque si se utilizan demasiado grandes puede suceder lo contrario, por ello la elección de un λ apropiado siempre es un proceso tardado y varía con cada imagen a utilizar.

8.2. Imagen "Figs"

La imagen anterior "SyG" (Figura 8.1) era muy grande, y al parecer eso beneficiaría más al algoritmo Multigrid, por ello a continuación se analiza una imagen más pequeña, se trata de "Figs" que es una imagen de 512×512 píxeles (ver Figura 8.3).

A esta imagen se le aplicó 20 % de ruido Uniforme, que es más difícil de quitar que el de Sal y Pimienta. Los resultados de las pruebas aplicadas a esta imagen utilizando $\lambda = 2.1$ son los mostrados en la Tabla 8.3.

A diferencia del análisis de la imagen "SyG" (Figura 8.1), en este caso, para poder tener una perspectiva más amplia, se comparan las mejores soluciones de cada algoritmo, y se muestran en la Tabla 8.4. En ésta se colocan las mejores soluciones de

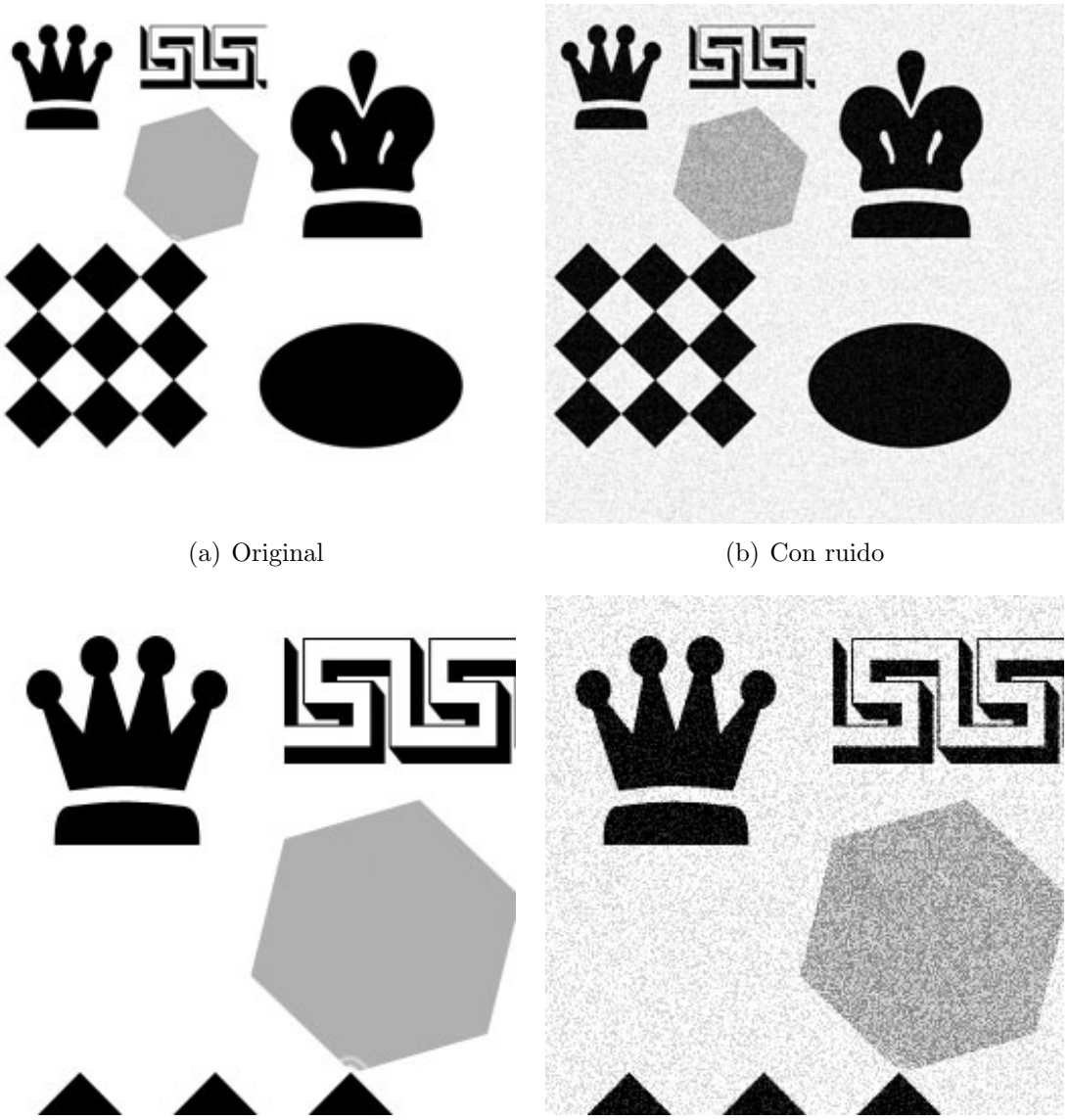
(a) Multigrid (b) Gauss Seidel

(c) Jacobi (d) Gradiente

Figura 8.2: Imagen "SyG con ruido después" de aplicarle cada uno de los algoritmos.

cada algoritmo, y el tiempo, que tardó, para obtenerlas, se puede observar claramente que la mejor solución de cada algoritmo tarda demasiado en comparación al método Multigrid.

Con estos resultados obtenidos (ver Figura 8.4), se puede observar que aún cuando la imagen sea pequeña se obtienen muy buenos resultados con el algoritmo Multigrid, y para este caso en particular se comprueba de que no siempre convergen los algoritmos con cualquier λ, que fue el caso del algoritmo de Gradiente. También



(a) Original (b) Con ruido

(c) Original 100 % (d) Con ruido 100 %

Figura 8.3: Imagen “Figs”.

puede observarse que ha disminuido el ruido considerablemente.

8.3. Imagen “Graduados”

La imagen “Graduados” (Figura 8.5) es de tamaño 2048×2048 píxeles, en realidad esta imagen no era de estas dimensiones, y por ello se agregaron píxeles blancos arriba y abajo mediante el programa de Edición de Imágenes “Photoshop”, y

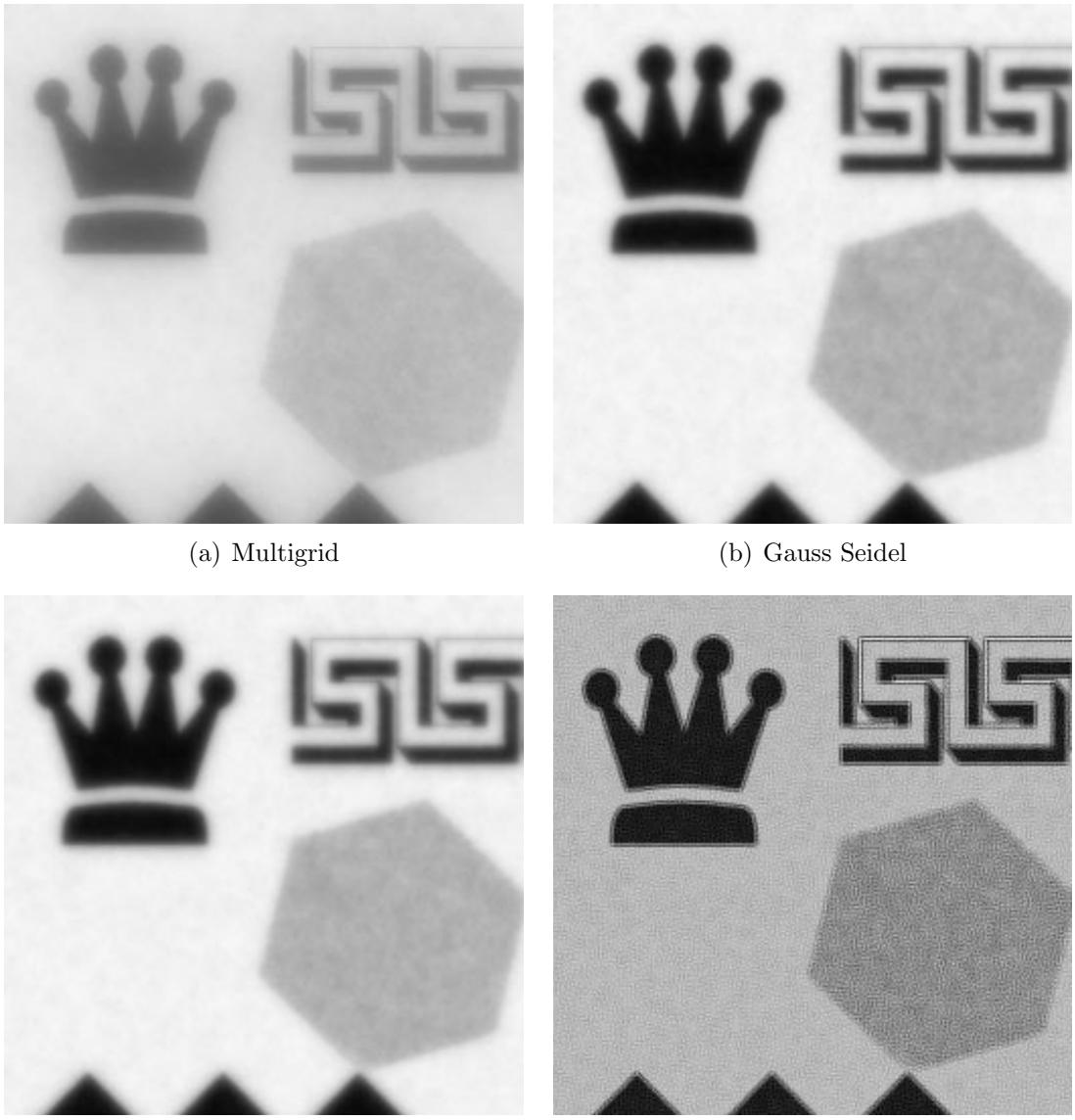
Algoritmo	**No. Iteraciones**	**Tiempo(seg)**	**Costo**
Multigrid	1	0.15625	4996631756.1726
Gauss Seidel	1	0.07812	6020260998.0574
Gauss Seidel	2	0.14062	5815830983.1846
Gauss Seidel	5	0.34375	5661846779.4723
Gauss Seidel	10	0.70312	5590546887.9895
Gauss Seidel	20	1.39062	5555476112.2445
Gauss Seidel	50	3.43750	5546454553.6053
Gauss Seidel	100	7.21875	5546300554.2675
Gauss Seidel	1000	68.26562	5546300262.4365
Gauss Seidel	2000	137.09375	5546300262.4365
Jacobi	1	0.07812	6121908097.7906
Jacobi	2	0.14062	5926671945.4444
Jacobi	5	0.35937	5746694304.8716
Jacobi	10	0.73437	5653545829.1588
Jacobi	20	1.40625	5587760677.1288
Jacobi	50	3.54687	5550456026.6696
Jacobi	100	7.07812	5546441804.2025
Jacobi	1000	70.28125	5546300262.4365
Jacobi	2000	142.60937	5546300262.4365
Gradiente	1	0.10937	7666616631.9699
Gradiente	3	0.28125	11206743043.6630
Gradiente	5	0.50000	30203560379.9165
El Gradiente no converge para este valor de λ.			

Tabla 8.3: Resultados de la imagen "Figs con ruido", utilizando $\lambda = 2.1$.

Algoritmo	**Tiempo(seg)**	**Costo**
Multigrid	0.15625	4996631756.1726
Gauss Seidel	137.09375	5546300262.4365
Jacobi	142.60937	5546300262.4365
Gradiente	0.10937	7666616631.9699

Tabla 8.4: Los mejores resultados de la función de costo obtenido en cada algoritmo.

aplicándole un ruido Uniforme al 10 % se obtiene "Graduados con ruido". Los resultados de las pruebas aplicadas a la imagen con ruido utilizando $\lambda = 1.3$ son los



(a) Multigrid (b) Gauss Seidel

(c) Jacobi (d) Gradiente

Figura 8.4: Imagen “Figs con ruido” después de aplicarle cada uno de los algoritmos.

mostrados en la Tabla 8.5.

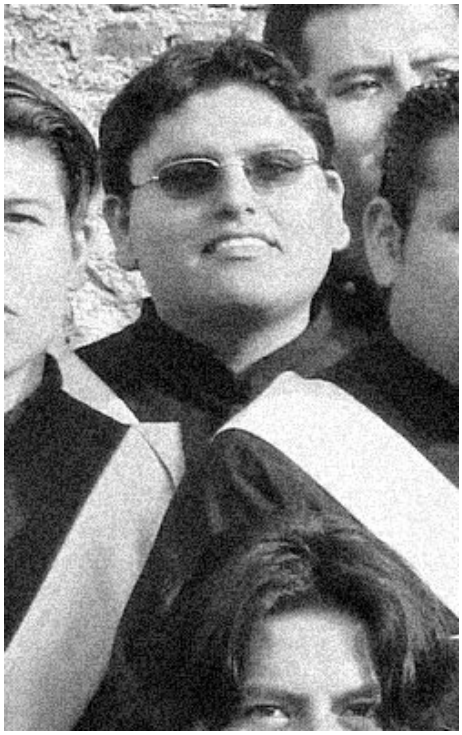
Para esta imagen se toma una “instantánea” lo más cerca posible en tiempo a la solución del método Multigrid. Obteniendo la Tabla 8.6 con sus respectivas imágenes.

En la Figura 8.6 se observa el resultado de haber aplicado cada uno de los algoritmos, en la imagen 8.6(a), se pudiera pensar que tiene mucho “suavizado”, y así es, posiblemente la imagen no es del todo agradable a la vista, pero es la que

(a) Original

(b) Con ruido

(c) Original 100 %



(d) Con ruido 100 %

Figura 8.5: Imagen “Graduados”.

tiene menor costo, que es el principal objetivo de los algoritmos.

8.4. Imagen “Ajedrez”

La imagen “Ajedrez” (ver Figura 8.7) es de tamaño 2048×2048 píxeles y se le aplicó 10 % de ruido Gaussiano para obtener “Ajedrez con ruido”. Los resultados de las pruebas aplicadas a esta imagen utilizando $\lambda = 3.1$ son los mostrados en la Tabla 8.7.

En la Figura 8.8 se pueden observar las imágenes resultado obtenidas con el menor costo computado (ver Tabla 8.8). En dicha Tabla de igual manera se puede ver que el algoritmo Multigrid supera a los demás.

Algoritmo	No. Iteraciones	Tiempo(seg)	Costo
Multigrid	1	2.64062	75291889067.5247
Gauss Seidel	1	1.09375	78397287193.1818
Gauss Seidel	2	2.20312	77790225097.2812
Gauss Seidel	5	5.48437	77457335882.3189
Gauss Seidel	20	21.67187	77303430341.8687
Gauss Seidel	50	55.48437	77298436317.2718
Gauss Seidel	100	112.79687	77298422986.6423
Jacobi	1	1.42187	78635338227.7444
Jacobi	2	2.75000	78110284057.9072
Jacobi	5	6.71875	77627062134.9420
Jacobi	20	26.73437	77338647717.9023
Jacobi	50	66.48437	77299991833.6385
Jacobi	100	136.56250	77298433702.3737
Gradiente	1	1.95312	78914867319.3749
Gradiente	2	3.75000	78295614177.9472
Gradiente	5	9.43750	77572697184.3025
Gradiente	20	37.64062	77326812432.3754
Gradiente	50	93.45312	77299157685.3746
Gradiente	100	185.84370	77298425520.2959

Tabla 8.5: Resultados de la imagen “Graduados con ruido”, utilizando $\lambda = 1.3$.

Algoritmo	Tiempo(seg)	Costo
Multigrid	2.64062	75291889067.5247
Gauss Seidel	2.20312	77790225097.2812
Jacobi	2.75000	78110284057.9072
Gradiente	3.75000	78295614177.9472

Tabla 8.6: Resultado de evaluar la función de costo de cada uno de los algoritmos, en la imagen “Graduados con ruido”, en un tiempo aproximado, al tiempo que tardó el algoritmo Multigrid.

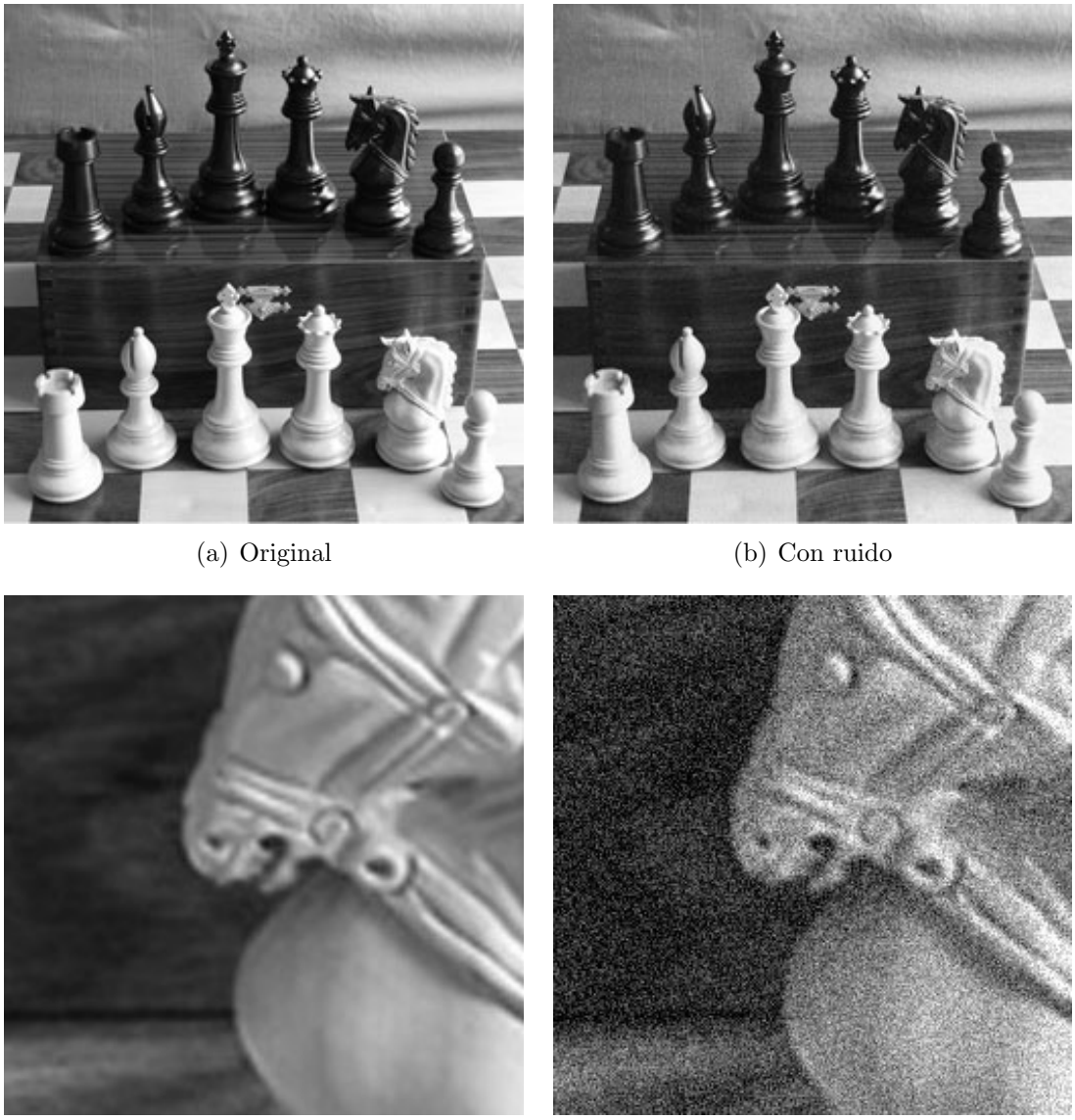
8.5. Variando λ

Hasta ahora se han hecho pruebas observando el resultado obtenido en alguna iteración o momento dado, sin embargo el valor de λ (lambda) que se ha utilizado

(a) Multigrid (b) Gauss Seidel

(c) Jacobi (d) Gradiente

Figura 8.6: Imagen “Graduados con ruido” después de aplicarle cada uno de los algoritmos.



(a) Original (b) Con ruido

(c) Original 100 % (d) Con ruido 100 %

Figura 8.7: Imagen Ajedrez.

en la función de costo (Ecuación 7.8), favorece el resultado de cada una de las imágenes. Es decir el valor escogido de λ utilizado en las pruebas anteriores no ha sido elegido de manera arbitraria, sino que se ha intentado que la imagen resultado sea más agradable a nuestra vista. Si bien el proceso de obtener una λ adecuada es tardado, al variar su valor podremos ver que los resultados siguen un patrón.

Para realizar estas pruebas se utilizó el algoritmo Multigrid en la imagen "Ajedrez con ruido", los resultados son los mostrados en la Tabla 8.9.

Algoritmo	**No. Iteraciones**	**Tiempo(seg)**	**Costo**
Multigrid	1	2.62500	31576959591.331
Gauss Seidel	1	1.09375	40031930519.139
Gauss Seidel	2	2.21875	37213276768.520
Gauss Seidel	5	5.53125	36320791418.860
Gauss Seidel	20	21.71875	35864846061.333
Gauss Seidel	50	54.81250	35803013848.193
Gauss Seidel	100	108.31250	35799618268.764
Jacobi	1	1.28125	41725415248.179
Jacobi	2	2.64062	38324975516.329
Jacobi	5	6.50000	36704821946.120
Jacobi	20	25.32812	36010568644.903
Jacobi	50	63.50000	35835768983.818
Jacobi	100	126.25000	35802780747.476
Gradiente	1	1.71875	286336902929.037
Gradiente	2	3.78125	1327942014701.960
El gradiente no converge para este valor de λ.			

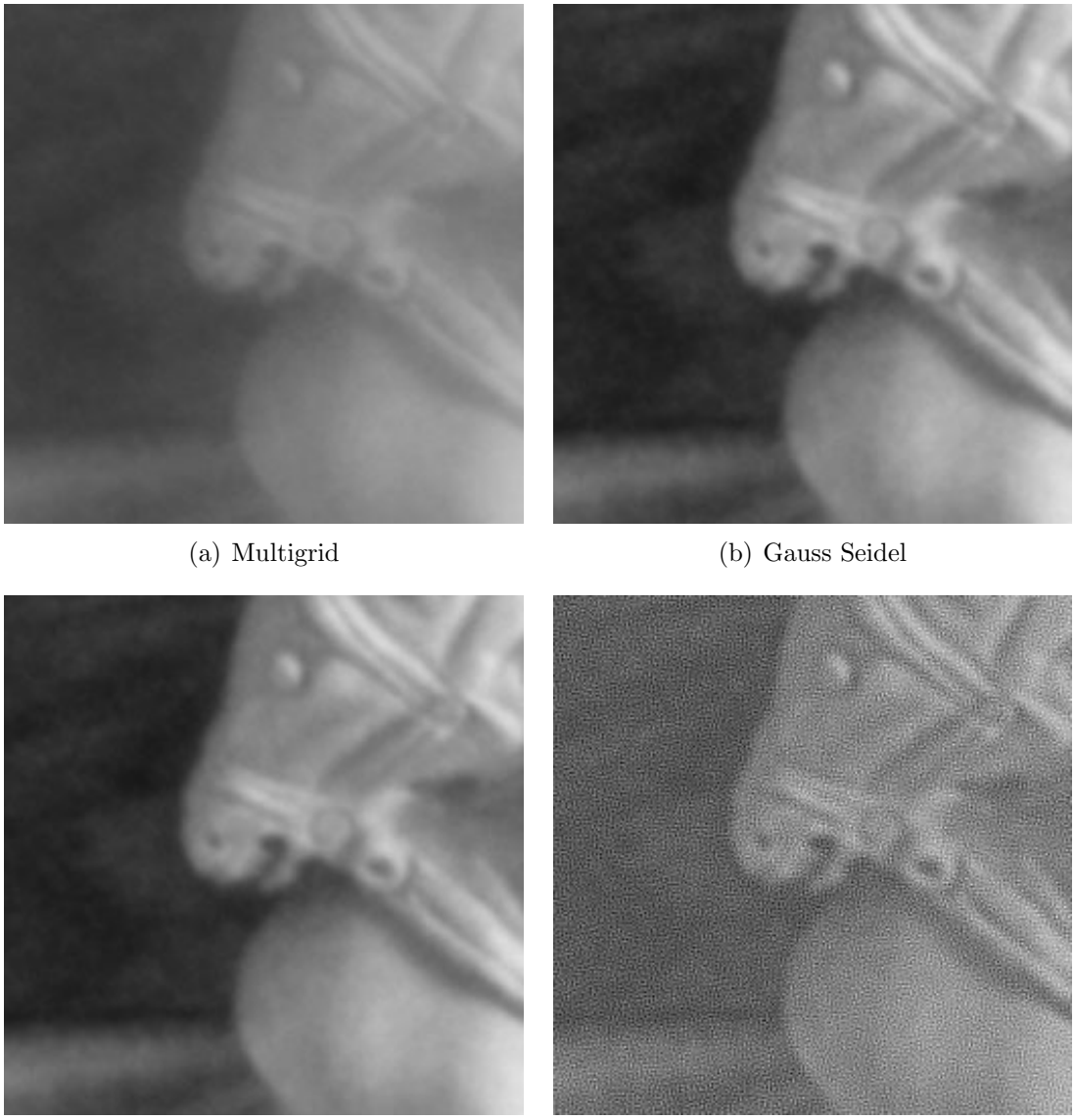
Tabla 8.7: Resultados de la imagen "Ajedrez con ruido", utilizando $\lambda = 3.1$.

Algoritmo	**Tiempo(seg)**	**Costo**
Multigrid	2.62500	31576959591.331
Gauss Seidel	108.31250	35799618268.764
Jacobi	126.25000	35802780747.476
Gradiente	1.71875	1327942014701.960

Tabla 8.8: Los mejores resultados de la función de costo obtenidos en cada algoritmo de la imagen "Ajedrez con ruido".

λ	Tiempo(seg)	Costo
3.1	2.625	31576959591.3314
2.1	2.71875	33049622315.6799
1.1	2.71875	34775832795.5694
0.5	2.6875	35885128092.4729

Tabla 8.9: Resultados obtenidos variando el valor de λ en la imagen "Ajedrez con ruido".

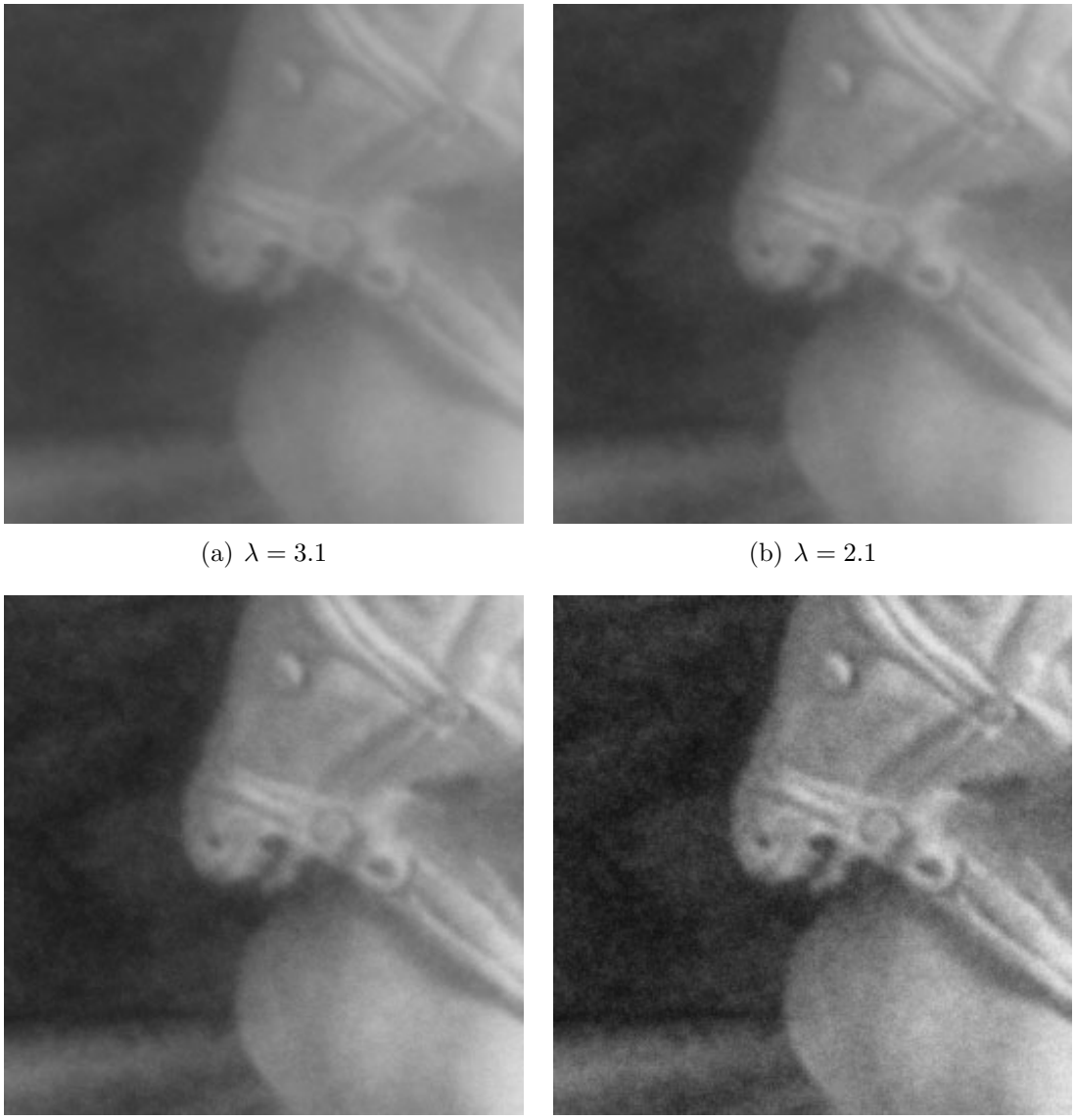


(a) Multigrid (b) Gauss Seidel

(c) Jacobi (d) Gradiente

Figura 8.8: Imagen “Ajedrez con ruido” después de aplicarle cada uno de los algoritmos.

Las imágenes resultado, son las mostradas en la Figura 8.9. En éstas se puede apreciar que entre mayor sea el valor de λ mayor será el efecto de “suavizado” que tendrá la imagen resultante. Esto quiere decir, que la imagen tendrá un aspecto cada vez más liso, logrando valores más homogéneos entre píxeles cercanos. También se puede observar que al ir decrementando el valor de λ, el algoritmo Multigrid tiene un valor más alto al evaluarse en la función de costo. Esto nos dice que entre más pequeño sea el valor de λ el algoritmo converge a un punto más alejado de la solución



(a) $\lambda = 3.1$ (b) $\lambda = 2.1$

(c) $\lambda = 1.1$ (d) $\lambda = 0.5$

Figura 8.9: Imagen "Ajedrez con ruido" después de aplicar el algoritmo Multigrid con variaciones en el valor λ.

$\hat{f}(x, y)$ óptima. Por lo tanto la imagen resultante se parecerá más a la imagen original con ruido.

8.6. Variando Iteraciones

En los algoritmos Jacobi, Gauss Seidel y Gradiente es importante mencionar el número de iteraciones que realizan en cada prueba, puesto que, entre más iteraciones

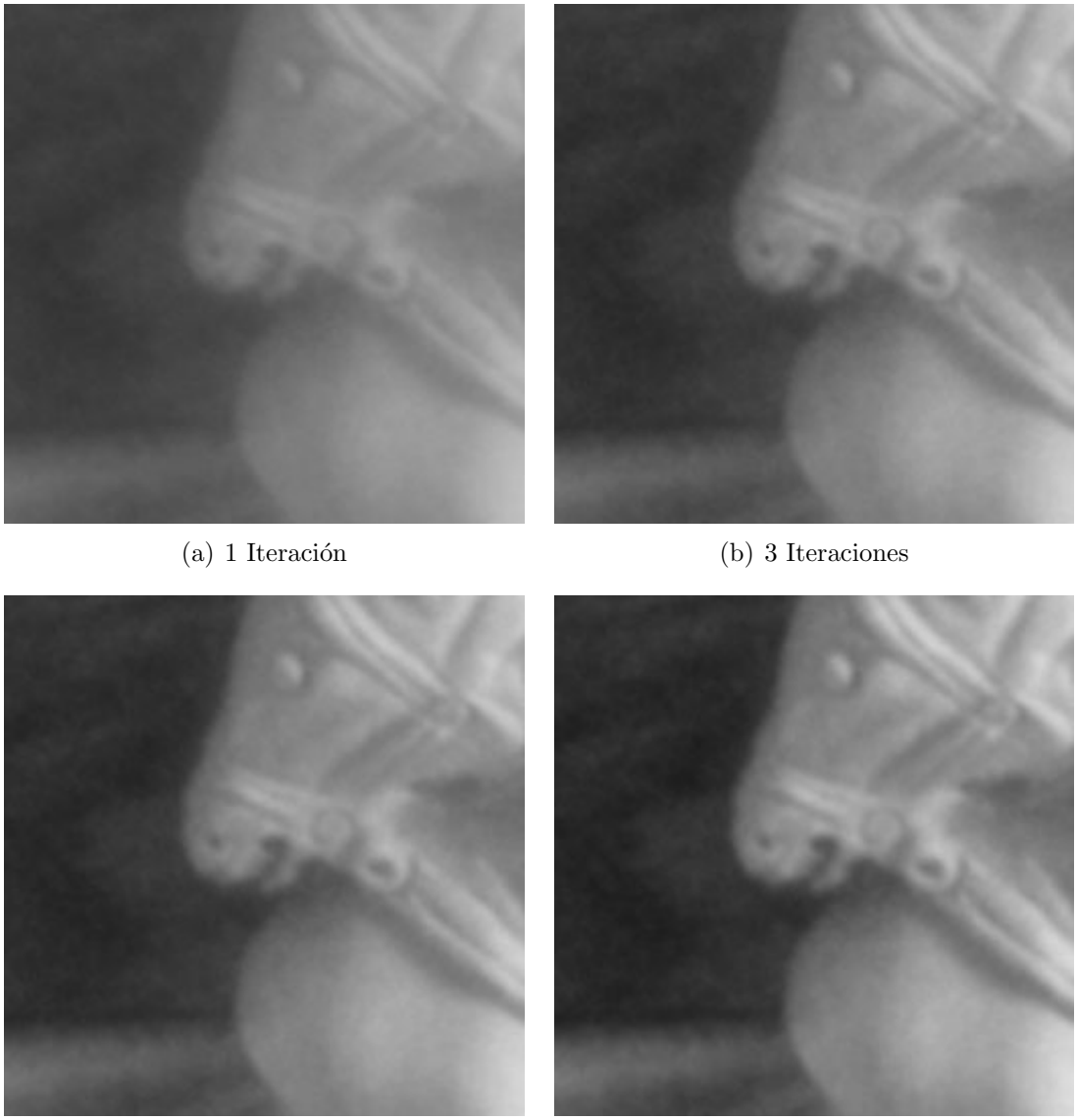
se realicen pueden converger a un valor más cercano al óptimo. Sin embargo esto no es aplicable al algoritmo Multigrid, ya que como se explicó en el capítulo 7, éste realiza un número de iteraciones dependiendo de que malla esté resolviendo en un instante dado. Pero lo que si se sabe es que en la capa más fina solo realiza una iteración del algoritmo Gauss Seidel, aplicando una iteración más en cada malla subsequente. Por ello en esta sección se analiza lo que sucede cuando se modifica el número de iteraciones que se deben realizar en la malla más fina, alterando en sí el número aplicado a cada una de las mallas.

Las pruebas se realizan sobre la imagen "Ajedrez con ruido", aplicando un valor fijo de λ, específicamente $\lambda = 3.1$, y se varía el número de iteraciones en la capa más fina del algoritmo Multigrid, mostrando los resultados en la Tabla 8.10.

No. Iteraciones	Tiempo(seg)	Costo
1	2.625	31576959591.3314
3	5.46875	33275942892.5634
5	8.28125	34216929161.5627
10	15.03125	35195551235.3298

Tabla 8.10: Resultados de aplicar el algoritmo Multigrid a la imagen "Ajedrez con ruido" con distinto número de iteraciones en la malla más fina.

En la tabla 8.10 se muestra el número de iteraciones que se realizan en la capa más fina del algoritmo Multigrid, el tiempo que tardó en realizarlas y, su respectivo valor de la función de costo. Es evidente que el tiempo de ejecución se elevase puesto que se tienen que realizar más iteraciones en cada capa. Sin embargo algo que sorprende es que la función de costo tenga un valor más elevado entre más iteraciones se realicen. Esto significa que entre más iteraciones, el algoritmo hace que se tomen más en consideración los valores de los píxeles de la imagen con ruido, logrando que se parezca más a ésta, aunque no sea del todo deseado. Las imágenes resultado pueden verse en la Figura 8.10.



(a) 1 Iteración

(b) 3 Iteraciones

(c) 5 Iteraciones

(d) 10 Iteraciones

Figura 8.10: Imagen “Ajedrez con ruido” al variar el número de iteraciones realizadas en cada capa.

Capítulo 9

Conclusiones

Al observar los resultados de las pruebas queda muy claro que el algoritmo Multigrid superó contundentemente a los demás algoritmos, tanto en la evaluación de la función de costo, como en el tiempo empleado para resolver el problema. Se puede ver que la velocidad de los algoritmos Gauss Seidel, Jacobi y Gradiente es muy lenta y los mejores resultados de cada uno eran aún muy alejados de la solución que encontraba el algoritmo Multigrid.

Por el momento las deficiencias que tiene el algoritmo Multigrid es que sólo trabaja con matrices cuadradas que son múltiplos de dos, no así los demás algoritmos que trabajan con imágenes de cualquier dimensión, aunque en la realidad esto no presenta un problema insuperable, no así el tiempo y los recursos utilizados por los demás algoritmos que aún dejan mucho que desear, sobre todo al trabajar con imágenes grandes. Al observar los resultados en las tablas, se da uno cuenta que en la mayoría de los casos el algoritmo Jacobi, mostró un peor desempeño que los demás. Aunque el método de Gradiente no siempre convergía.

En algunos resultados de las pruebas, como es en el caso de la imagen "Ajedrez con ruido" (ver Figura 8.8), se podría decir que el algoritmo multigrid suaviza demasiado la imagen a tal grado que pareciese que los demás algoritmos hicieron un mejor trabajo aún teniendo una evaluación en la función de costo mayor que la de Multigrid, sin embargo éste es el resultado óptimo al que deberían llegar todos

los algoritmos, aunque su aspecto no sea del todo agradable. Para solucionar este "problema" de cómo luce la imagen se pueden seguir dos criterios:

1. El primero es disminuir el valor de lambda para que el suavizado no sea demasiado evidente (ver sección 8.5).

2. El segundo, el cual es un poco más complicado, funciona de la siguiente manera: cuando se tiene una imagen y se le aplica el algoritmo Multigrid, a cada capa se le puede aplicar un número de iteraciones distinto. Sin embargo, entre más iteraciones se le apliquen a cada capa el algoritmo converge más lentamente a una solución que es más costosa que la óptima. Esto sucede porque en cada una de las capas se utiliza el algoritmo Gauss Seidel, que obliga a que la imagen resultante se parezca un poco más a la imagen con ruido inicial (ver sección 8.6).

Por todos estos resultados se puede ver que el algoritmo Multigrid es eficiente.

Apéndice A

Tipos de Ruido

Para poder comparar los resultados de los algoritmos presentados en este trabajo de tesis, fue necesario que se trabajara con imágenes degradadas. Para conseguir esto, a las imágenes originales se les agregó un tipo de ruido.

Ruido en imágenes *es una fluctuación aleatoria, generalmente no deseada, de los valores de los píxeles en una imagen [15].*

Los tipos de ruido que se utilizaron son los descritos en las siguientes secciones.

A.1. Sal y Pimienta

El ruido Sal y Pimienta, es típico en imágenes, y se logra al modificar aleatoriamente píxeles, a valores de blanco y negro [15].

A.2. Ruido Uniforme

El ruido Uniforme, sigue una función de densidad de probabilidad (ver Figura A.1) dada por [9]:

$$p(z) = \begin{cases} \frac{1}{b-a} & \text{si } a <= z <= b \\ 0 & \text{de otra forma} \end{cases}$$

con media:

$$\mu = \frac{a+b}{2} \tag{A.1}$$

y varianza:

$$\sigma^2 = \frac{(b-a)^2}{12} \tag{A.2}$$

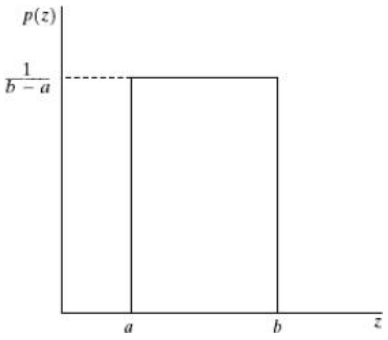


Figura A.1: Función de densidad de probabilidad Uniforme.

A.3. Ruido Gaussiano

La función de densidad de probabilidad (ver Figura A.2), de una variable aleatoria Gaussiana, z, está dada por [9]:

$$p(z) = \frac{1}{\sqrt{2\pi\mu}} e^{-(z-\mu)^2/2\sigma^2} \tag{A.3}$$

donde z representa niveles de gris, μ es la media del valor promedio de z y σ es su desviación estándar.

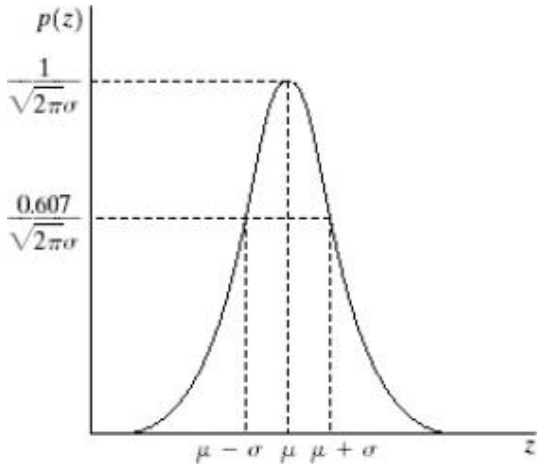


Figura A.2: Función de densidad de probabilidad Gaussiana.

Apéndice B

Algoritmos de los métodos iterativos

En este apéndice se muestran los algoritmos de los métodos iterativos de programación no lineal sin restricciones descritos en la sección 6.2.

B.1. Método de Jacobi

Algoritmo [15]:

Eligir una hipótesis inicial $x^{(0)}$
Para $k = 1, 2, ...$ *repetir hasta que haya convergencia*
    Para $i = 1, 2, ..., n$ *repetir*
        $\sigma = 0$
        Para $j = 1, 2, ..., n$ *repetir*
            si j!=i entonces $\sigma = \sigma + A_{ij} x_j^{(k-1)}$
        $x_i^{(k)} = \frac{(b_i - \sigma)}{A_{ii}}$
    Verificar si la convergencia ha sido alcanzada

B.2. Método Gauss Seidel

Algoritmo [15]:

Eligir una hipótesis inicial $x^{(0)}$
Para $k = 1, 2, ...$ *repetir hasta que haya convergencia*
    Para $i = 1, 2, ..., n$ *repetir*

$\sigma = 0$
Para $j = 1, 2, ..., i-1$ *repetir*
$\sigma = \sigma + A_{ij}x_j^{(k)}$
Para $j = i+1, ..., n$ *repetir*
$\sigma = \sigma + A_{ij}x_j^{(k-1)}$
$x_i^{(k)} = \frac{(b_i - \sigma)}{A_{ii}}$
Verificar si la convergencia ha sido alcanzada

B.3. Método de Gradiente

Algoritmo [15]:

Datos de entrada: x_0, ε *(donde es* ε *un valor de tolerancia)*
Para $k = 0, 1, 2, ...$ *repetir*
Resolver: $d_k = -\nabla f(x_k)$
Calcular α_k
$x_{k+1} = x_k + \alpha_k d_k$
Finalizar hasta que $||\nabla f(x_{k+1})|| < \varepsilon$

Apéndice C

Codificación del algoritmo Multigrid

El algoritmo Multigrid, así como el de Gauss Seidel, Jacobi y el de Gradiente fueron implementados en el lenguaje de programación C#, utilizando el ambiente de programación Microsoft Visual Studio .NET 2003, creando de esta forma el software con nombre "ImageRestore".

Función Multigrid

```
/*
* Multigrid
* En esta función solamente se inicia al algoritmo
* en su forma recursiva
* */

public void multigrid()
{
 multigrid_recursivo(x,y,ref fr,ref gr,No_Iteraciones);
}

 /*
  * multigrid_recursivo
```

```
  * Esta función es la principal
  * Recibe como parámetros:
  * tamaño de la imagen original => (tamx,tamy)
  * imagen FR (imagen que se desea obtener) => tempFR
  * imagen GR (imagen original) => tempGR
  * el número de iteraciones para esa imagen => itr
  * */
private void multigrid_recursivo(int tamx, int tamy,
        ref double [,]tempFR,
        ref double [,]tempGR,
        int iter)
{
 int nx, ny;
  /*
  * Caso Base:
  * cuando tamx = 2 && tamy == 2
  * se aplica el método del gradiente un número iteraciones itr
  * */
 if(tamx == 2 && tamy == 2)
 {
  aplicar_gauss(tamx,tamy,ref tempFR,ref tempGR,iter);
  return;
 }
 /*
  * Si no es el caso base, hay que reducir la imagen
  * en potencia de dos
  * */
```

```
   nx = tamx/2;
   ny = tamy/2;
   double[,] pqueFR = new double[nx,ny];
   double[,] pqueGR = new double[nx,ny];
   for(int i = 0; i < nx; i++)
    for(int j = 0; j < ny; j++)
    {
     pqueFR[i,j] = tempFR[i*2,j*2];
     pqueGR[i,j] = tempGR[i*2,j*2];
    }
   multigrid_recursivo(nx,ny,ref pqueFR, ref pqueGR,iter+1);
   // una vez que se regresó de la recursión, hay que realizar
    // una interpolación para aquellos valores que se desconocen
   interpolar(nx,ny,ref pqueFR, ref tempFR);
    // una vez hecho esto hay que iterar en el resultado, para tomar
    // en cuenta los valores de la imagen original
   aplicar_gauss(tamx,tamy,ref tempFR,ref tempGR,iter);
  }
```

Función Gauss Seidel utilizada.

```
private void aplicar_gauss(int tamx, int tamy,ref double[,] tempFR,
                           ref double[,] tempGR, int iteraciones)
  {
   int id, jd, conta;
```

```
   // conta nos dice cuántos vecinos tiene i,j

   double fs;
   for(int z = 0; z < iteraciones; z++)
    for(int i = 0; i < tamx; i++)
     for(int j = 0; j < tamy; j++)
     {

    // hay que recorrer los vecinos

       fs = 0.0;
       conta = 0;
       for(int k = 0; k < 8; k++)
       {
        id = despx[k] + i;
        jd = despy[k] + j;
        if(id < tamx && jd < tamy && id >= 0 && jd >= 0)
        {
         fs += tempFR[id,jd];
         conta++;
        }
       }
   tempFR[i,j] = (double)(tempGR[i,j] +
                 (lambda*fs))/(double)(1.0+(conta*lambda));
  }
 }
```

Función de interpolación

```
   /*
    * Función bilineal
```

```
  * Toma los valores de los vecinos del píxel desconocido
  * en un rango de 2 x 2 píxeles de la imagen conocida
   * se puede observar que en esta función, solo se obtienen los valores
   * de esos píxeles, pero en donde se hace el promedio es en la
   * función de interpolar
  * */
private void bilineal(ref double []vec, ref double [,]peque,
                      int pi, int pj, int nx, int ny)
{
 int []despx4 = new int[]{0, 0, 1, 1};
 int []despy4 = new int[]{0, 1, 1, 0};
 int ti, tj;
 for(int i = 0; i < 4; i++)
 {
  ti = pi+despx4[i];
  tj = pj+despy4[i];
  if(pixel_valido(nx, ny, ti, tj))
   vec[i] = peque[ti,tj];
  else
   vec[i] = peque[pi,pj];
 }

}


 /*
  * Función que interpola, donde los valores que se desconocen
  * se encuentran mediante el método Bilineal
```

```
  * */

private void interpolar(int nx, int ny, ref double [,]peque,
                        ref double[,] grande)
{
 int ni, nj;
 double []vecinos = new double[]{0,0,0,0};

 // [0,1,2,3] => [(i,j),(i,j+1),(i+1,j+1),(i+1,j)]

 for(int i = 0; i < nx; i++)
  for(int j = 0; j < ny; j++)
  {
   ni = i*2;
   nj = j*2;
   grande[ni,nj] = peque[i,j];
   bilineal(ref vecinos, ref peque, i, j, nx, ny);
   grande[ni+1,nj] = (vecinos[0]+vecinos[3])/2.0;
   grande[ni,nj+1] = (vecinos[0]+vecinos[2])/2.0;
   grande[ni+1,nj+1] = (vecinos[0]+vecinos[1]+vecinos[2]+vecinos[3])/4.0;
  }
}
```

Apéndice D

Manual de Usuario

D.1. Introducción

El programa ImageRestore fue realizado conjuntamente con la elaboración de esta tesis. Éste representa la implementación de los conceptos teóricos, y es de gran ayuda para poder realizar las comparaciones entre los diversos algoritmos, además es utilizado para eliminar ruido de imágenes cuadradas en formato bmp.

El programa fue desarrollado en el lenguaje de programación C#, utilizando el ambiente de programación Microsoft Visual Estudio .Net 2003. Los algoritmos implementados fueron Multigrid, Jacobi, Gauss Seidel, Gradiente y Mediana.

D.2. Aspectos generales de ImageRestore

D.2.1. Requerimientos mínimos

Para que el programa funcione correctamente es necesario que se cuente con una computadora con las siguientes características:

- Computadora PC o compatible.
- Procesador Pentium III, compatible o superior.
- 128 MB en RAM.

- 250 MB libres en disco duro.
- Lector de CD ROM.
- Sistema Operativo Windows XP.

D.2.2. Instalación

1. Instalar DOTNETFX. Este es el framework necesario para poder ejecutar aplicaciones hechas en Microsoft Visual Studio, es posible que su computadora ya tenga instalada una versión de este componente, pero en caso de que no sea así, posiciónese en la raíz del CD que contiene el software, y de doble clic en el programa con nombre DOTNETFX.exe que tiene asociado el siguiente icono (ver Figura D.1). Posteriormente siga los pasos de la instalación.

Figura D.1: Framework necesario.

2. Una vez que se está seguro que ya está instalado el framework posiciónese en la raiz del CD, y de doble clic en el programa con nombre InstalacionImage-Restore.exe que tiene asociado el siguiente icono (ver Figura D.2).

Figura D.2: Instalador de ImageRestore.

3. Se le mostrará una pantalla donde le da la bienvenida a la instalación. Presione el botón “Next”.

4. Ahora se le mostrará una pantalla en donde se le solicita que seleccione el Folder donde se efectuará la instalación. Se recomienda dejar los elementos predeterminados. Presione "Next" para continuar la instalación.

5. Se le preguntará si confirma instalar el programa. Presione "Next" para comenzar la instalación. Verá una ventana muy parecida a la Figura D.3:

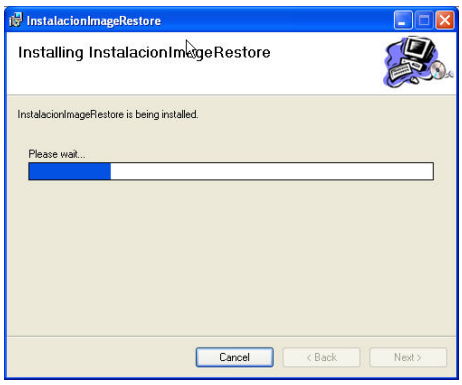


Figura D.3: Instalación en progreso.

6. Cuando haya finalizado la instalación sólo presione "Close", y listo!!!.

D.3. Módulos del sistema.

Para ejecutar el programa, de clic en Inicio⇒Todos los programas⇒ImageRestore. De clic en el programa con el nombre ImageRestore.exe.

Las opciones principales del programa se muestran en la Figura D.4. Todas estas opciones se encuentran en su menú correspondiente. También se cuenta con opciones que sólo sirven cuando se utiliza algún algoritmo en específico como Multigrid, Gauss Seidel, Jacobi, ó Gradiente, ya que especifican valores necesarios para su buen funcionamiento (ver Figura D.5). Aunque estas opciones traen valores por default, éstos se pueden cambiar al hacer clic en ellos y escribir los datos adecuados.

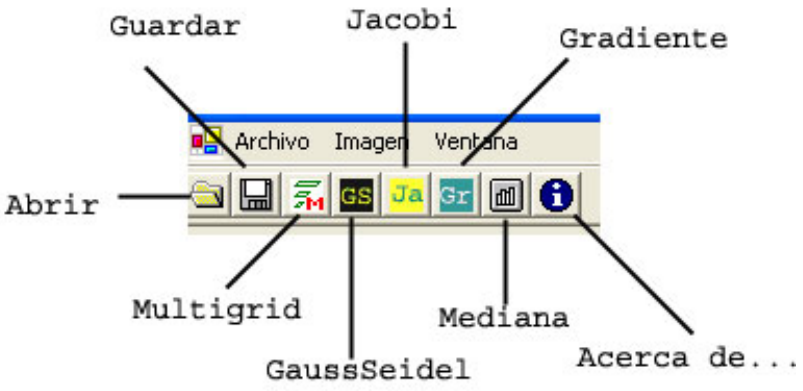


Figura D.4: Opciones principales.

Figura D.5: Opciones Adicionales.

Cada una de las opciones se describe a continuación.

D.3.1. Abrir

Una vez que haya abierto el programa lo primero que debe hacer es abrir una imagen, utilizando la opción del menú Archivo⇒Abrir... , o bien presionando el icono indicado para esto. En la carpeta de instalación C:\Archivos de programa\InstalacionImageRestore hay una carpeta con el nombre Imágenes_de_Prueba que contiene varias imágenes que ya son cuadradas y que están listas para utilizar en el programa.

D.3.2. Guardar

Si tiene una imagen abierta, puede almacenar una copia de ella tal y como luce en ese momento, ya que las imágenes originales que se cargan no se modifican, esto con la finalidad de que se puedan hacer las comparaciones que considere pertinentes.

D.3.3. Multigrid

Para aplicar este algoritmo de clic en el icono correspondiente, claro que debe especificar el valor de Lambda deseado, además de que el Número de iteraciones que se especifiquen son el número de iteraciones que se realizarán en la capa más fina de la imagen.

D.3.4. Gauss Seidel, Jacobi y Gradiente

Para aplicar cualquiera de estos algoritmos defina el Número de iteraciones que desee (entre más sean tendrá un mejor costo, pero el tiempo para que termine puede ser demasiado), así como el valor de Lambda, y a continuación de clic en su icono correspondiente.

D.3.5. Mediana

Este algoritmo a diferencia de los anteriores no utiliza Número de iteraciones, ni valor de Lambda, por ello solo de clic en el icono correspondiente y es todo.

D.3.6. Acerca de...

Información acerca del autor.

También existen otras dos opciones, pero estas no son de entrada sino de salida, se trata de la duración, y Costo, estas opciones muestran el resultado de el algoritmo que se haya aplicado (ver Figura D.6).

Figura D.6: Duración y costo.

El tiempo que se muestra en el campo duración es en segundos, y únicamente mide el tiempo de ejecución del algoritmo, y no el tiempo en que carga la imagen

y la despliega. Costo es la evaluación de la función 7.8 aplicada a la imagen al final de la última iteración.

Si se desea ver el resultado de varias imágenes abiertas, puede alinear las ventanas de manera más conveniente usando el menú ventana⇒mosaico.

Apéndice E

Glosario

D

Discretización. Concerniente al proceso de transferir modelos y ecuaciones continuos en su contraparte discreta. Este proceso es usualmente llevado a cabo como un primer paso para poder realizar evaluaciones numéricas e implementaciones en computadoras [15].

E

Ecuaciones diferenciales parciales (PDE). Es una relación que envuelve una función desconocida de muchas variables independientes y sus derivadas parciales con respecto a dichas variables. Ecuaciones diferenciales parciales son usadas para formular y resolver problemas de funciones con muchas variables [15].

Estocástico. Palabra proveniente del griego. Significa "perteneciente o relativo al azar" según la RAE [10].

Se denomina estocástico a aquel sistema que funciona, sobre todo, por el azar. Las leyes de causa-efecto no explican cómo actúa el sistema (y de modo reducido el fenómeno) de manera determinista, sino en función de probabilidades [15].

Evidencia. Es el conjunto de los hallazgos disponibles en un determinado momento o situación: $e = \{H1 = h1, \ldots, Hr = hr\}$ [3].

F

Función: Una función relaciona cada una de sus entradas a exactamente una salida. Una notación estándar para la salida de una función f con entrada x, es $f(x)$. El conjunto de todas las entradas que acepta una función es llamado el dominio de una función. El conjunto de todas las salidas es llamado rango [15].

Función real: Es una función matemática cuyo dominio es la línea real.

H

Hallazgo: Es la determinación del valor de una variable, H = h, a partir de un dato (una observación, una medida, etc.) [3].

L

Línea real: En matemáticas es simplemente el conjunto R de número reales. Sin embargo este término es usualmente utilizado cuando R es tratado como un espacio de alguna clase, como un espacio topológico, o un espacio vectorial [15].

P

Probabilidad a priori. Es la probabilidad de una variable o subconjunto de variables cuando no hay ningún hallazgo [3].

Probabilidad a posteriori. Es la probabilidad de una variable o subconjunto de variables dada la evidencia e. Una representación puede ser P^* [3].

$$P^*(\bar{x}) \equiv P(\bar{x}|e).$$

Bibliografía

[1] Niclas Bergman. *Recursive Bayesian Estimation, Navigation and Tracking applications.* Linkoping, Sweden, 1999.

[2] Edwin K. P. Chong and Stanislaw H.Zak. *An Introduction to Optimization.* Prentice Hall International, 1996.

[3] F. J Díez. *Introducción al Razonamiento aproximado.* Prentice Hall, revisión 2005.

[4] Marcos Martín Fernández. *Campos aleatorios de Markov.* Publicación PDF, 2004.

[5] González and Woods. *Digital Image Processing.* Prentice Hall, 2002.

[6] Castleman Kenneth R. *Digital Image Processing.* Prentice Hall, 1996.

[7] José Marroquín. Deterministic interactive particle models for image processing and computer graphics. *Graphical Models and Image Processing*, 55:408–417, 1993.

[8] MathWorld. Algorithms. `http://mathworld.wolfram.com`, Fecha de último acceso: Julio de 2007.

[9] Trung Nguyen. Enciclopedia of computer science. `http://www.eleves.ens.fr`, Fecha de último acceso: Agosto de 2007.

[10] RAE. Real academia de la lengua española. `http://www.rae.es`, Fecha de último acceso: Agosto de 2007.

[11] Wolfram Research. Digital image processing. `http://www.wolfram.com/products/applications/digitalimage/quicktour/representation.html`, Fecha de último acceso: Agosto de 2007.

[12] José Marroquín Salvador Botello and Mariano Rivera. Multigrid algorithms for processing fringe pattern images. *Journal of Funny Physics*, 35:39–78, 1997.

[13] Hamdy A. Taha. *Investigación de operaciones*. Prentice Hall, 2004.

[14] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3D Computer Vision*. Prentice Hall, 1998.

[15] Wikipedians. Wikipedia, la enciclopedia libre. `http://en.wikipedia.org`, Fecha de último acceso: Agosto de 2007.