



## DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico III

## Desvelando la mentira de los megapíxeles

---

Métodos Numéricos  
Segundo Cuatrimestre de 2014

Integrante	LU	Correo electrónico
Aldasoro Agustina	86/13	agusaldasoro@gmail.com
Bouzón María Belén	128/13	belenbouzon@hotmail.com
Cairo Gustavo Juan	89/13	gjcairo@gmail.com



Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires  
Ciudad Universitaria - (Pabellón I/Planta Baja)  
Intendente Güiraldes 2160 - C1428EGA  
Ciudad Autónoma de Buenos Aires - Rep. Argentina  
Tel/Fax: (54 11) 4576-3359  
<http://www.fcen.uba.ar>

## Resumen

En el presente trabajo se pretende analizar y comparar distintas alternativas que pretenden dar respuesta al problema de demosaicing. Para ello, las mismas serán implementadas y sometidas a experimentación sobre fotografías crudas sintéticas específicamente seleccionadas para poner de manifiesto las ventajas e inconvenientes del empleo de cada método.

### **Palabras clave:**

- Filtro Bayer
- Demosaicing
- Interpolación
- Artifacts

## Índice

<b>1. Introducción Teórica</b>	<b>3</b>
1.1. Inconvenientes de aproximar mediante el Polinomio de Lagrange . . . . .	4
1.2. Métricas de comparación de imágenes . . . . .	9
1.3. Artifacts . . . . .	10
<b>2. Desarrollo</b>	<b>13</b>
2.1. Vecino más cercano . . . . .	13
2.2. Interpolación Bilineal . . . . .	20
2.3. Interpolaciones Direccionales . . . . .	21
2.4. Splines . . . . .	23
2.5. Algoritmo de Malvar, He y Cutler . . . . .	33
<b>3. Resultados y discusión</b>	<b>34</b>
3.1. Exploración de Artifacts . . . . .	34
3.2. Análisis cualitativo de los Métodos . . . . .	35
3.3. Análisis de tiempos . . . . .	48
<b>4. Conclusiones y trabajo futuro</b>	<b>49</b>
<b>5. Apéndices</b>	<b>49</b>
5.1. Apéndice A: Enunciado . . . . .	49
5.2. Apéndice B: . . . . .	49
<b>6. Referencias</b>	<b>50</b>

## 1. Introducción Teórica

Promediando la década de 1990 comenzaron a introducirse al mercado las cámaras digitales de uso hogareño, empezando a sustituir desde entonces a las ampliamente difundidas cámaras de film. Mientras que estas últimas capturaban la imagen de forma mecánica y sin intercesión de un procesamiento digital, las cámaras electrónicas emplearon una nueva tecnología. Esta tecnología distingue entre en el momento de obtención de un conjunto de datos cerca del objetivo – donde se obtiene una imagen que llamaremos “cruda” – y una etapa de procesamiento automático conjunto a la reconstrucción de la imagen final.

En el presente trabajo nos interesará analizar un fragmento de dicha etapa conocido como *demosaicing*. Para entender de qué consta, es menester comprender previamente la forma en que se capturan las imágenes al utilizar cámaras digitales.

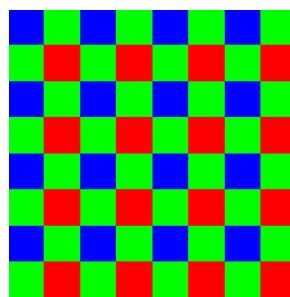
Este tipo de cámaras poseen un sensor CCD compuesto de una matriz de elementos fotosensibles, cada uno de los cuales es capaz de captar la intensidad de la luz que llega a ese punto a través de la lente.

La tarea de capturar todos los colores reflejados por el objetivo que se quiere fotografiar no está recomendada a ninguna posición en particular, sino que cada una de ellas tiene asignada – de acuerdo a algún patrón definido – la captura de la información perteneciente a un solo color. Cada píxel de la imagen va a estar dividido en tres canales ya que la vamos a almacenar, en nuestro caso de estudio, mediante RGB (Red, Green, Blue).

El patrón que determina qué color será capturado por cada fotosensor es denominado *Color Filter Array* y varía de acuerdo al fabricante de cámaras. Existen diversos patrones, en nuestro trabajo nos centraremos en el **Filtro Bayer**, ya que es un filtro comúnmente usado debido a que la cantidad de información que brinda sobre el canal verde - al cual el ojo humano es más sensible por encontrarse en el centro de su espectro visual y es el canal que brinda mayor información sobre la luminiscencia - duplica a cada uno de los otros dos.

\*insertar imágenes por aquí y por allá\* (?)

Figura 1: Bayer CFA



Ahora bien, esto genera una imagen que se aproxima a la realidad sólo en parte ya que para cada píxel de la imagen tenemos la información de sólo uno de sus tres canales. A causa de esto es que la misma debe ser reconstruida algorítmicamente, interpolando los valores de los colores que no fueron almacenados explícitamente para cada pixel.

En este punto adquieren relevancia las diferentes propuestas de reconstrucción de la imagen original. Esto conlleva la ejecución de diversos procedimientos, entre los que se encuentran el demosaicing, el análisis del balance de blancos, la curva tonal, la saturación y el contraste y la compresión final de la imagen. Como nuestra propuesta se aboca al primer paso mencionado, exploraremos en el presente trabajo sólo un subconjunto de una serie de variaciones no finita que admiten métodos de debayering como ser la interpolación por asignación de valores próximos, la interpolación bilineal y interpolación direccional.

A efectos de nuestro caso de estudio, la imagen bajo el Color Filter Array Bayer la vamos a generar sintéticamente, es decir mediante un algoritmo que tome una imagen original y devuelva una imagen de este formato. Esto nos permitirá hacer comparaciones entre los métodos y la imagen original, teniendo así un punto de referencia en cuanto a la correctitud de la imagen.

### 1.1. Inconvenientes de aproximar mediante el Polinomio de Lagrange

Es nuestra intención exponer ahora el motivo que consideramos suficiente para decidir interpolar los valores ausentes a través de Splines en lugar de hacer uso del polinomio de Lagrange.

Sea dada la siguiente muestra:

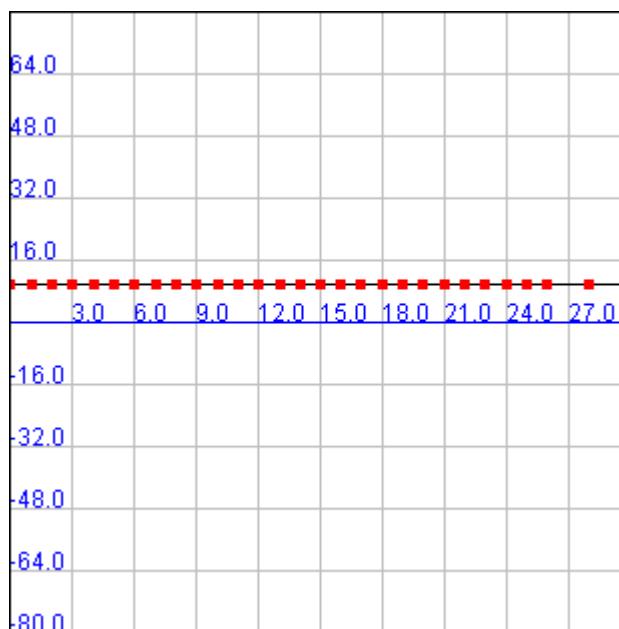
x	1	2	3	4	5	6	7	8	9	10	11	12	13	14
y	10	10	10	10	10	10	10	10	10	10	10	10	10	10

x	15	16	17	18	19	20	21	22	23	24	25	26	27	28
y	10	10	10	10	10	10	10	10	10	10	10	10	10	10

El polinomio de Lagrange que interpola cada uno de esos valores tiene una expresión equivalente al Spline que lo hace, siendo su gráfico el que se muestra a continuación:

Figura 2:



Supongamos ahora que el resultado de una medición - digamos  $x_{12}$  - haya sido calculada con un ruido que represente el 1 % del valor real, modificándose la muestra de la siguiente forma:

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14
y	10	10	10	10	10	10	10	10	10	10	10	10,1	10	10

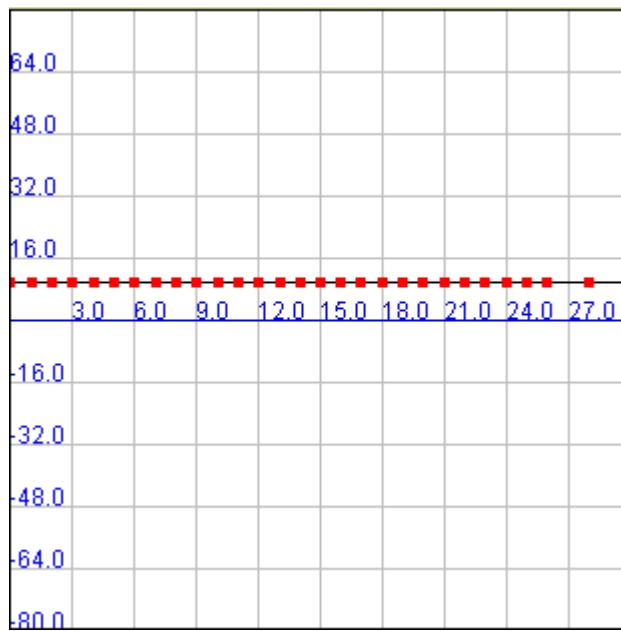
x	15	16	17	18	19	20	21	22	23	24	25	26	27	28
y	10	10	10	10	10	10	10	10	10	10	10	10	10	10

En este caso se obtendría el gráfico representado en la Figura 4 para el polinomio de Lagrange (escalado en la Figura 5) y el que se ilustra en la Figura 3 para el obtenido mediante Splines.

Como se puede observar, para la construcción del Spline el cambio en un valor de la imagen no impactó de una forma apreciable sobre el resto del polinomio. Sin embargo, el nuevo resultado generó un polinomio de Lagrange cuya diferencia con la Figura 3 es grosera en determinados intervalos.

Si se profundizara el error de la medición de acuerdo con la tabla presentada a continuación, notaríamos que el polinomio de Lagrange comenzaría a presentar una mayor cantidad de intervalos con

Figura 3:



valores drásticamente alejados de los dados como referencia. Por otro lado, Splines únicamente presentará variaciones en un entorno de los intervalos cuyos valores hayan sido modificados. Esto se puede constatar en las Figuras 6, 7 y 8.

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14
y	10	10	10	20	10	10	10	10	10	10	10	20	10	10

x	15	16	17	18	19	20	21	22	23	24	25	26	27	28
y	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Figura 4:

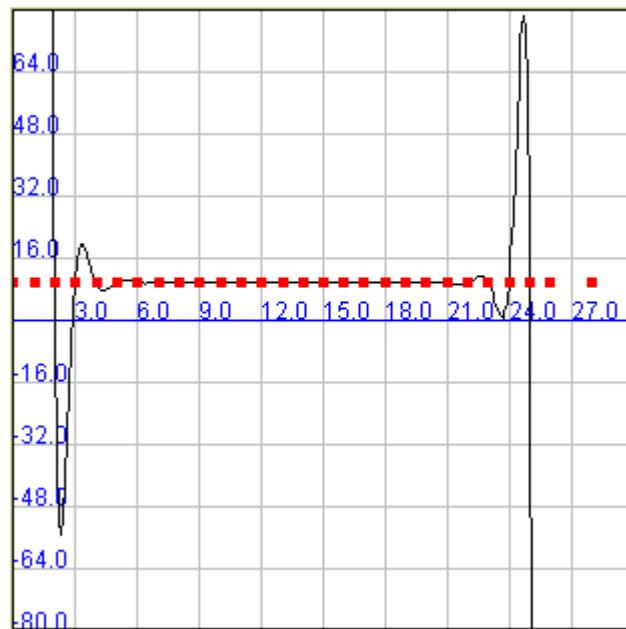


Figura 5:

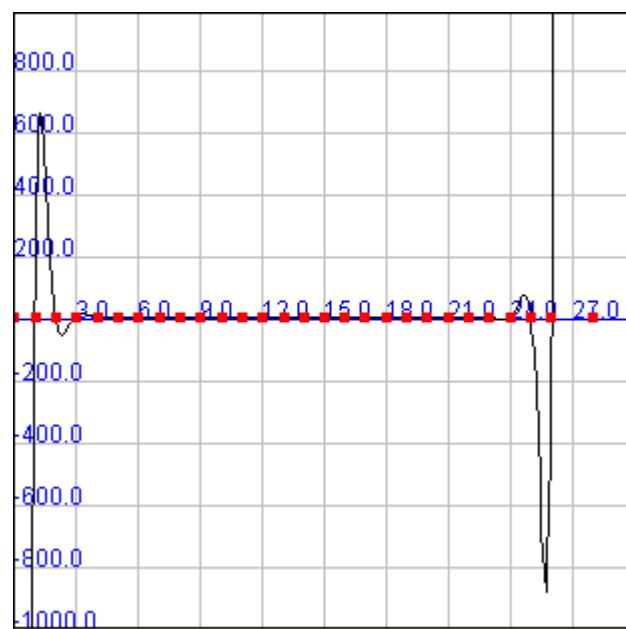


Figura 6:

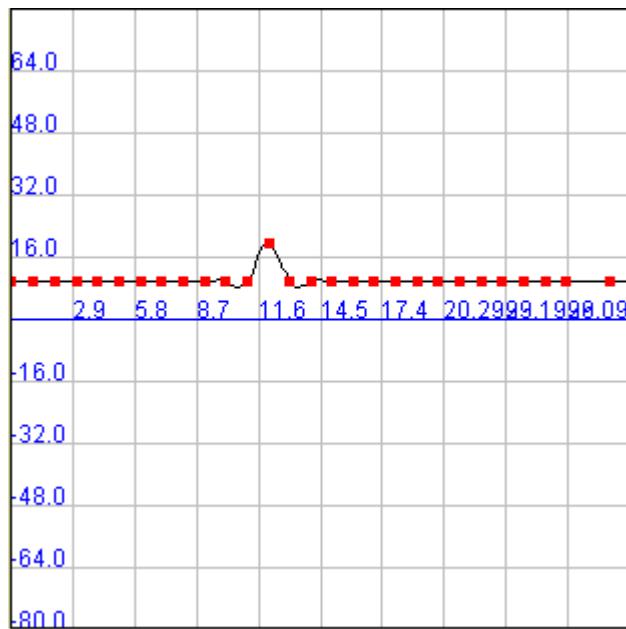


Figura 7:

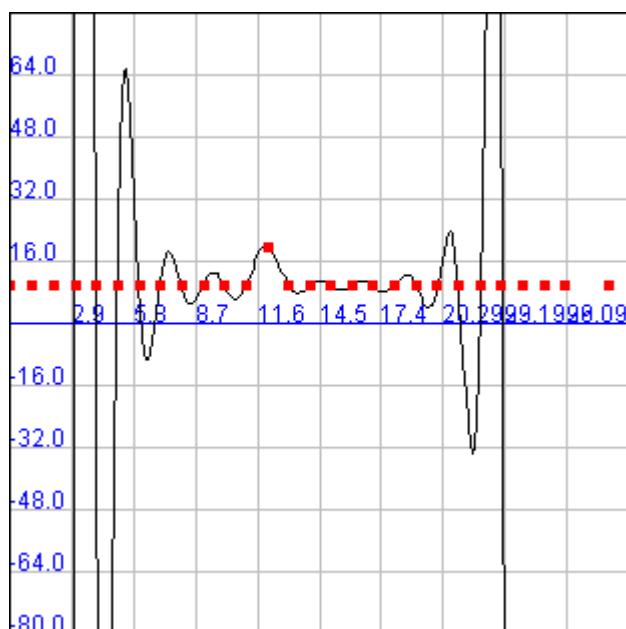
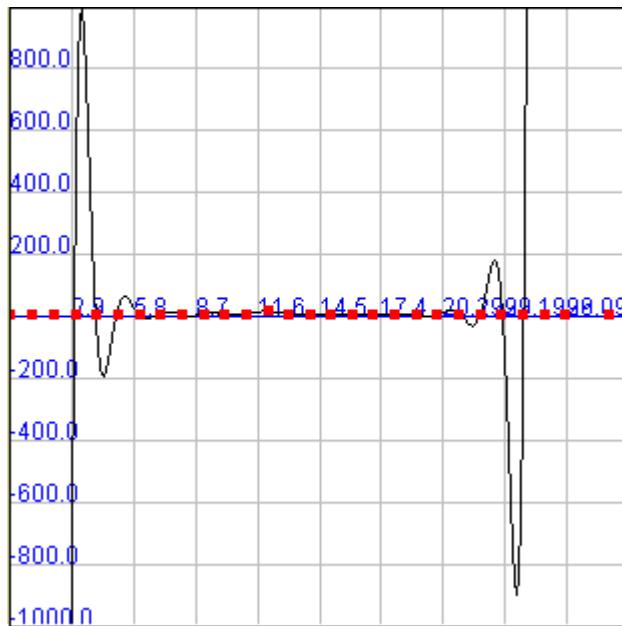


Figura 8: *Zoom out* de la Figura 7

En conclusión, el polinomio de Lagrange se comporta de una manera menos predecible estable entre dos variables independientes y no garantiza que en esos intervalos el valor de la función sea cercano a los ya conocidos.

La carencia de predictibilidad hace del polinomio de Lagrange una mala alternativa para la interpolación, en comparación con las ventajas proporcionadas por el empleo de Splines.

## 1.2. Métricas de comparación de imágenes

Con el fin de establecer una relación entre las imágenes obtenidas con los distintos métodos y la imagen original es que nace la necesidad de contar con *Métricas de comparación de imágenes*.

Existen dos métodos dentro de los mecanismos usados:

*Subjetivos*: Son los conceptos que una persona puede detectar visualmente.

*Objetivos*: Son operaciones matemáticas en el dominio bidimensional o tridimensional de imágenes orientados a determinar la variación presente en una imagen.

Las métricas subjetivas u objetivas se pueden usar con comparación respecto a una referencia o sin considerarla.

En nuestro caso de estudio, nos vamos a centrar en la comparación de la imagen original y la obtenida a través de los distintos métodos. Por esto mismo, vamos a usar métricas que cuenten con una imagen de referencia.

Dos de las métricas más sencillas son el *Error cuadrático Medio* (mse) y *Peak signal-to-noise ratio* (psnr). Ambas cuentan con la capacidad de análisis píxel por píxel.

$$\text{mse} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(x, y) - K(x, y)]^2$$

Donde  $I$  y  $K$  son imágenes a comparar de tamaño  $m \times n$ .

$$\text{psnr} = 10 \log \frac{(2^n - 1)^2}{mse} = 10 \log \frac{255^2}{mse}$$

Otro tipo de medidas a tener en cuenta son las que utilizan indicadores de la imagen similares a los presentes en la visión humana, como es el caso de *Structural similarity* (ssim) que es una función que depende de la luminiscencia, el contraste y la similitud estructural.

$$\text{ssim}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Donde,

$\mu_x$  es la esperanza de  $x$

$\mu_y$  es la esperanza de  $y$

$\sigma_x^2$  es la varianza de  $x$

$\sigma_y^2$  es la varianza de  $y$

$\sigma_{xy}$  es la covarianza entre  $x$  e  $y$

$c_1 = (k_1 L)^2$ ;  $c_2 = (k_2 L)^2$  son dos variables que estabilizan la división con un denominador chico

$L$  es el rango dinámico de los valores de los píxeles (comúnmente es  $2^{\# \text{bitsPorPixel}} - 1$ )

$k_1 = 0,01$  y  $k_2 = 0,03$  por default

Las métricas que vamos a utilizar nosotros como método de comparación de imágenes son *psnr* y *ssim*. Los cuales calcularemos con sus instrucciones respectivas en Matlab®.

### 1.3. Artifacts

#### Explicar cada artifact y agregar grid

Como se explicó anteriormente, dos de los tres canales de cada píxel son interpolados algorítmicamente. Si bien los métodos que pueden proponerse a tal fin no son limitados, un subconjunto de ellos producirá imágenes que diferirán considerablemente de las que quisieron ser capturadas originalmente.

Para evaluar el peso de estas diferencias existen mecanismos cuantitativos (como los mencionados en el apartado *Métricas de comparación de imágenes*) y otros de tipo cualitativo. Un grupo de estos últimos está constituido por la exploración de *artifacts*.

Los *artifacts* son efectos visuales indeseados frecuentes que se producen sobre la imagen procesada a causa de una interpolación poco conveniente.

Ejemplos de artifacts son el efecto «*Blurring*»(Figura 9), el «*False color*»(Figura 10), el «*Jaggies*»(Figura 12), el «*efecto Moiré*»(Figura 13) y el «*Zipper*»(Figura 11).

Figura 9: A la izquierda, la imagen original. A la derecha, la que presenta «blurring».

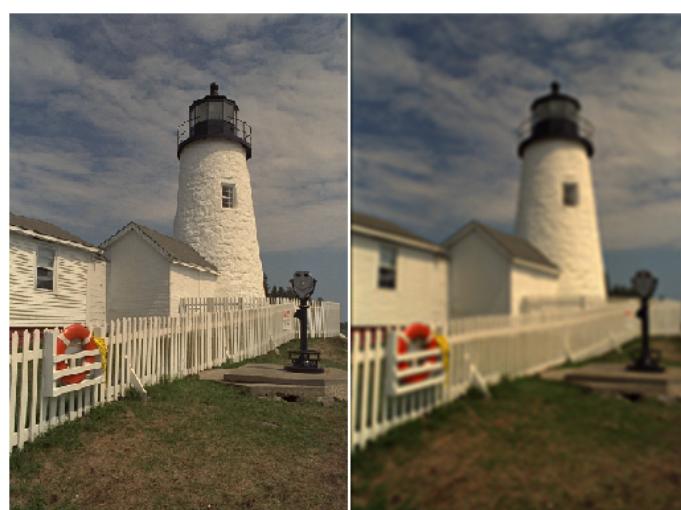


Figura 10: A la izquierda, la imagen original. A la derecha, la que presenta «false color».



Figura 11: A la izquierda, la imagen original. A la derecha, la que presenta «zipper».



Figura 12: A la izquierda, la imagen original. A la derecha, la que presenta «jaggies».



Figura 13: A la izquierda, la imagen original. A la derecha, la que presenta el efecto «moiré».



ESTARÍA BUENO AGREGAR TIPO ANECDÓTICO LOS ARTIFACTS DEL OJO HUMANO (?) PERO NO SÉ EN PRINCIPIO CÓMO MECHARLO ACÁ

## 2. Desarrollo

Dada una *imagen cruda*, los valores que contiene en sus píxeles para cada canal los asumimos reales y ciertos. Por lo tanto, resta averiguar para cada uno de ellos los valores de los dos canales para los cuales dicha información se encuentra ausente.

### 2.1. Vecino más cercano

La implementación más sencilla para la estimación de los canales desconocidos de cada pixel consiste en otorgar a cada canal “vacío” el valor real más próximo que corresponda a dicho color. Este método es algorítmicamente asequible, pero no tiene en cuenta dos factores:

▷ Por un lado, al otorgar el valor de un sólo pixel cercano, surge la necesidad de definir arbitrariamente cuál se debe escoger como valor referencial en casos en que un conjunto de pixeles son equidistantes al analizado y no existe un criterio claro acerca de cuál de ellos debe ser considerado el “más próximo”. Que esta decisión impacte en el resultado de forma inocua o parcial o totalmente nociva dependerá en cierta medida de la fotografía que se esté analizando.

El ejemplo más trivial se encuentra en una imagen monocromática: al ser los valores reales idénticos para todos los pixeles, tomar las magnitudes de cualquier otra posición para cada canal correspondiente no afectará el resultado final. Sin embargo, si consideráramos una imagen formada por columnas alternadas de un pixel de ancho rojas (#0000FF) y blancas (#FFFFFF), entonces podríamos analizar cómo sería procesada si se resolviera:

■ **Caso 1:** Adjudicar:

- A los valores nativos verdes: Los valores nativos - rojo y azul - que se encuentren a la derecha y por debajo de su posición.
- A los valores nativos azules: El valor verde de la posición inferior y el del rojo cuya ubicación se encuentra en su diagonal inferior derecha.
- A los valores nativos rojos: El valor verde de la posición inferior y el del azul cuya ubicación se encuentra en su diagonal inferior derecha.

■ **Caso 2:** Adjudicar:

- A los valores nativos verdes: Los valores nativos - rojo y azul - que se encuentren a la derecha y por debajo de su posición.
- A los valores nativos azules: El valor verde de la posición derecha y el del rojo cuya ubicación se encuentra en su diagonal inferior derecha.
- A los valores nativos rojos: El valor verde de la posición derecha y el del azul cuya ubicación se encuentra en su diagonal inferior derecha.

Los procedimientos y resultados de las experimentaciones realizadas con estos dos criterios se encuentran respectivamente en las Figuras 14, 15 y 16 para el **Caso 1** y en las ilustraciones 14, 15 y 17 para el **Caso 2**.

Figura 14:

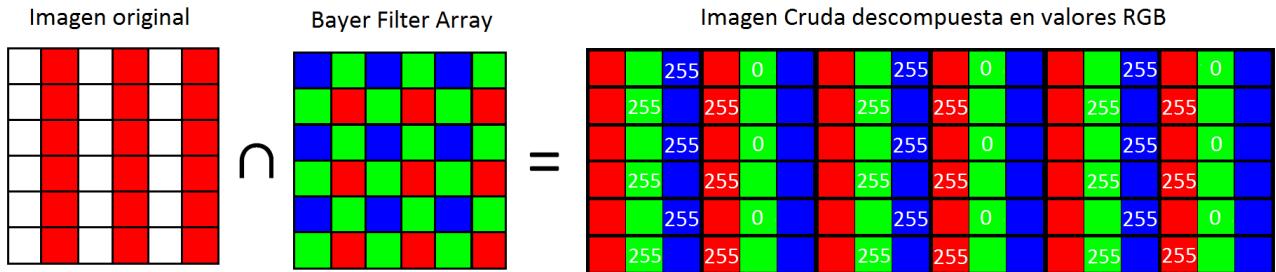


Figura 15:

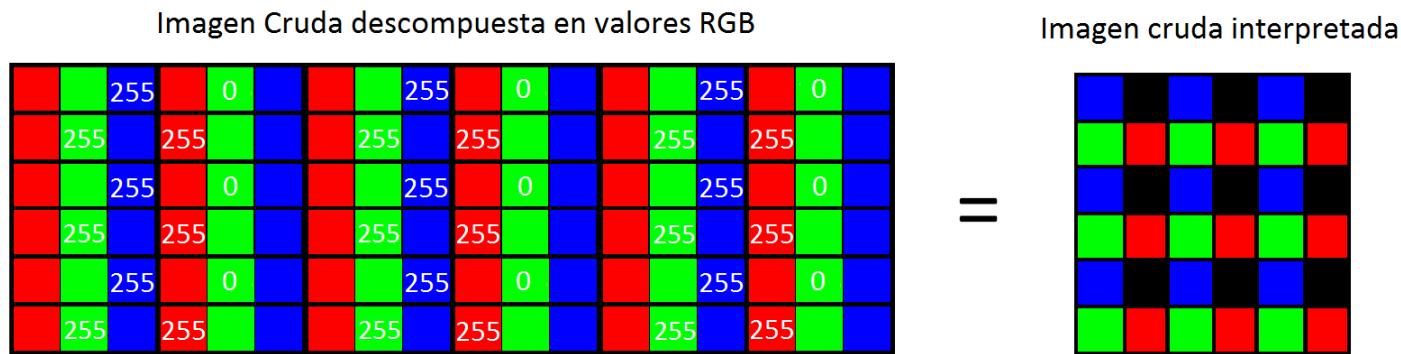
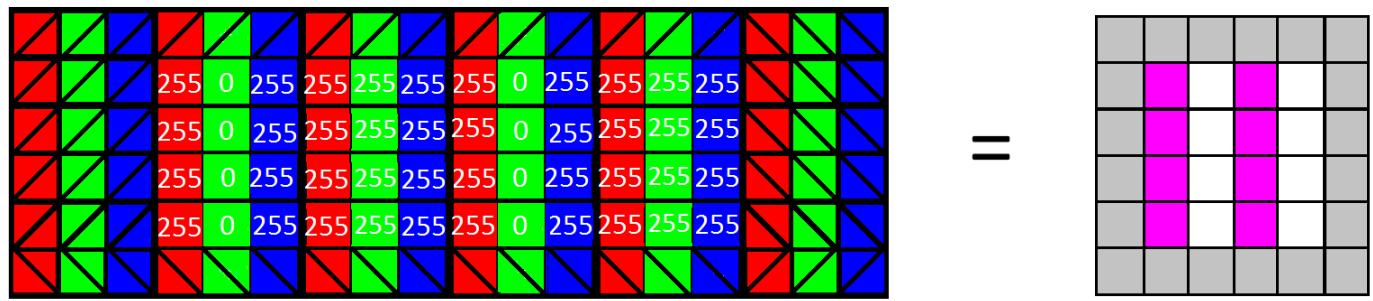


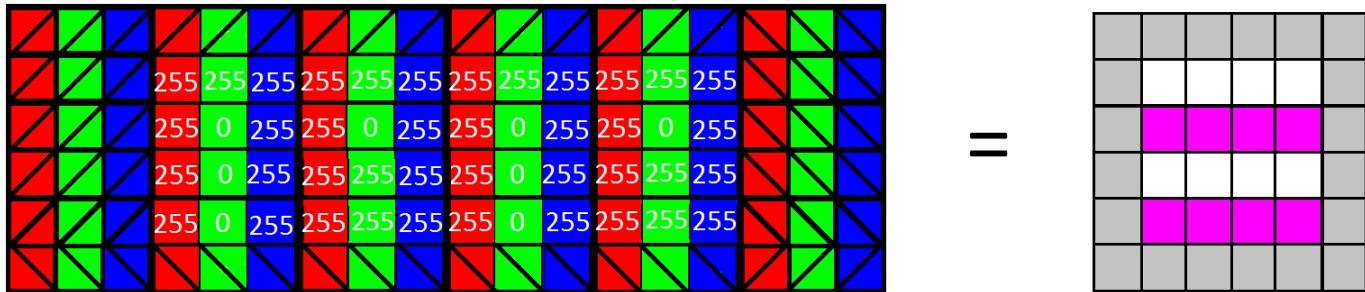
Figura 16: Valores para todos los canales luego de la interpolación realizada con un algoritmo de “vecino más cercano”, Caso 1



Como se ve, en ninguno de los dos casos se logró reconstruir la imagen original, pero sin duda alguna el error en el primer caso fue menor al segundo.

El algoritmo planteado evidencia que los resultados proporcionados serán acordes a la imagen que se procesa y a las decisiones arbitrarias que se tomen al implementarlo.

Figura 17: Valores para todos los canales luego de la interpolación realizada con un algoritmo de “vecino más cercano”, Caso 2



Si bien se trata de un ejemplo de alcance limitado, resulta suficiente para ilustrar los potenciales problemas de la aplicación de este algoritmo.

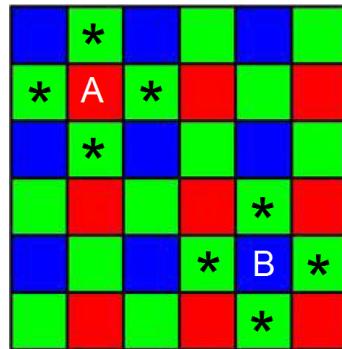
▷ Por otra parte - y en cierta forma vinculado al ejemplo anterior - este método no especifica ninguna distinción respecto de la forma de proceder en casos de borde o de superficies parejas.

### Decisiones tomadas en el Algoritmo de Vecino Más Cercano

A la hora de formular nuestro algoritmo de Vecino más Cercano nos vimos obligados a determinar aspectos que conciernen al diseño del mismo.

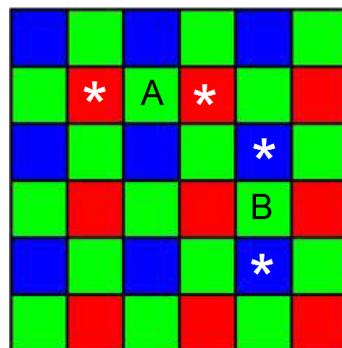
La principal decisión que debimos tomar fue con qué criterio elegir al píxel más cercano cuando no es único (esto ocurre en todos los casos).

Figura 18:



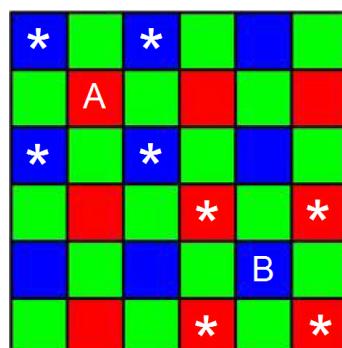
Para determinar el valor del canal verde sobre un píxel definido rojo(A) o azul(B) tenemos cuatro píxeles a la misma distancia (1) que nos brindan información sobre el canal verde.

Figura 19:



Si nos situamos en un píxel naturalmente verde, tanto para determinar su canal rojo(A) como para determinar su canal azul(B) sus píxeles más cercanos que nos brindan información son dos (a 1 de distancia cada uno).

Figura 20:



Y por último para determinar el valor del canal azul sobre un píxel de origen rojo(*A*) (y viceversa(*B*)) nos encontramos con cuatro píxeles a la misma mínima distancia (2) en sus diagonales.

Por este motivo, realizamos dos versiones de este algoritmo variando en ellas cuál píxel elegir para llenar el canal verde cuando se está situado sobre un píxel azul o rojo.

En el **algoritmo 1** si se está en un píxel rojo se elige al de arriba y si se está en un píxel azul se elige el de la derecha. En cambio, en el **algoritmo 2** si se está en un píxel rojo se elige al de la izquierda y si se está en un píxel azul se elige al de abajo.

Corrimos ambos algoritmos con diversas fotos, a continuación se muestran algunos ejemplos que contienen detallado el valor de psnr y ssim para cada imagen respecto de la original.

Figura 21:



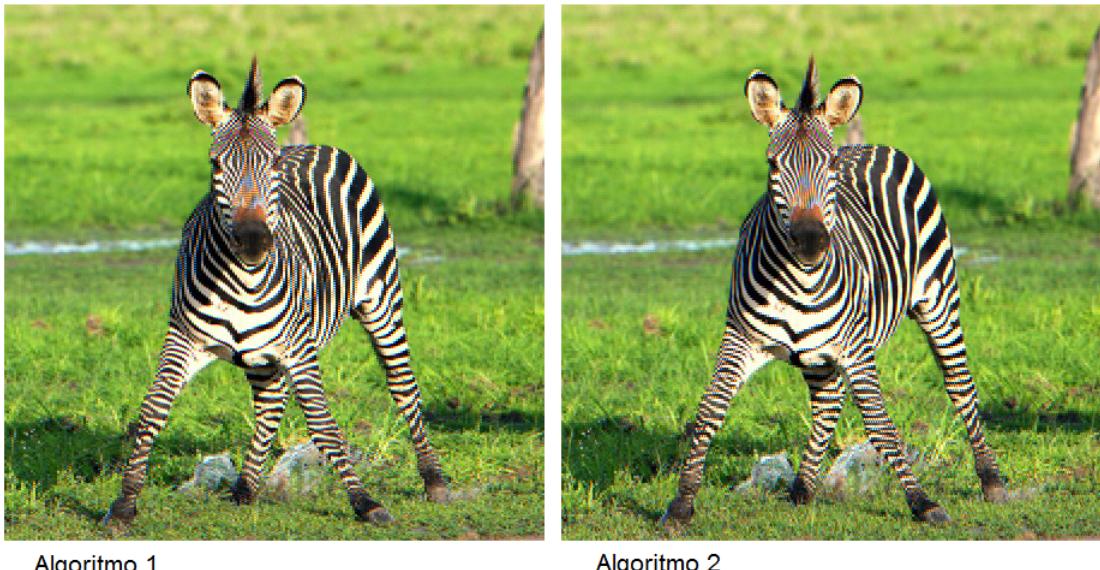
$$\text{psnr1} = 14.3119$$

$$\text{psnr2} = 14.3533$$

$$\text{ssim1} = 0.3819$$

$$\text{ssim2} = 0.3856$$

Figura 22:



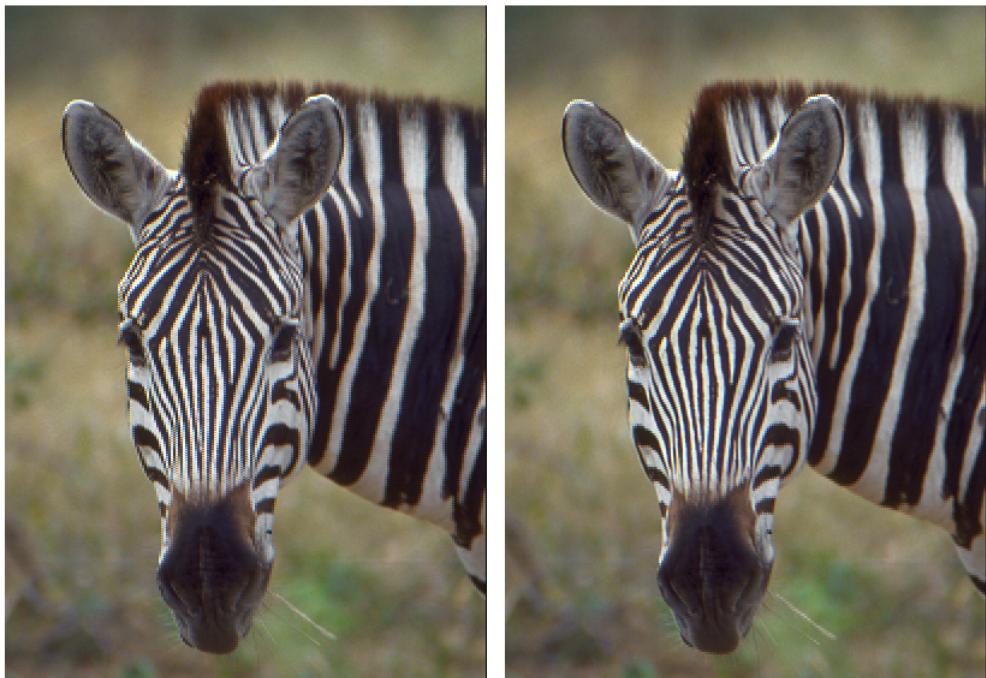
$$\text{psnr1} = 19.8295$$

$$\text{psnr2} = 20.0706$$

$$\text{ssim1} = 0.8806$$

$$\text{ssim2} = 0.8841$$

Figura 23:



Algoritmo 1

Algoritmo 2

psnr1 = 21.1167

psnr2 = 21.3368

ssim1 = 0.8374

ssim2 = 0.8417

Figura 24:



Algoritmo 1

Algoritmo 2

psnr1 = 23.2721

psnr2 = 23.2371

ssim1 = 0.8560

ssim2 = 0.8553

Figura 25:



Algoritmo 1

Algoritmo 2

$\text{psnr1} = 23.8518$

$\text{psnr2} = 23.8270$

$\text{ssim1} = 0.7514$

$\text{ssim2} = 0.7515$

Figura 26:



Algoritmo 1

Algoritmo 2

$\text{psnr1} = 21.7691$

$\text{psnr2} = 21.7494$

$\text{ssim1} = 0.9508$

$\text{ssim2} = 0.9506$

Se puede apreciar que el Algoritmo 1 es capaz de delimitar mejor bordes que poseen rayas en forma horizontal, en cambio el Algoritmo 2 le da una mejor definición a los bordes verticales. Dada la forma de la implementación, las anomalías observadas en las imágenes cobran sentido.

Debido a que no se puede establecer con un criterio cuál es más correcta que la otra, ya que la veracidad de la imagen obtenida varía depende la imagen inicial y su estructura. Nosotros optamos por utilizar el **Algoritmo 1**.

*Se adjuntan las imágenes en su tamaño original.*

## 2.2. Interpolación Bilineal

La interpolación bilineal es una composición de interpolaciones lineales en dos direcciones diferentes. Este método desmerece el carácter arbitrario de asignación de un canal a partir del valor más próximo para el mismo y propone que el resultado final sea calculado teniendo en cuenta y promediando todos los valores cercanos sin priorizar ninguno de ellos.

Como la estructura del filtro Bayer sólo asegura para los píxeles nativamente azules o rojos tener todas sus posiciones vecinas netamente verdes, aquel es el único caso en que el cálculo de un canal se reduce a computar el promedio de los valores superior, inferior, derecho e izquierdo al espacio que se quiere interpolar.

Sin embargo, si se quisiera obtener el valor de azul o rojo que correspondería asignar a un píxel nativamente verde, el entramado de Bayer obligaría a estimarlo a partir de sus dos valores más cercanos ubicados a la derecha e izquierda o arriba y abajo dependiendo de la ubicación del píxel.

Por otro lado, los valores del canal azul para un píxel rojo (y los del canal rojo para un píxel azul) se calcularían tomando en cuenta las cuatro posiciones diagonales a la analizada. Este proceso no se realizaría simplemente tomando el promedio de sus cuatro diagonales sino que requeriría tomar dos píxeles cercanos que pertenezcan a la misma fila o columna, cuyo canal real sea el verde, calcular sus valores de azul/rojo promediando los de las casillas nativas azules/rojas más próximas y posteriormente realizar un nuevo promedio entre los valores resultantes.

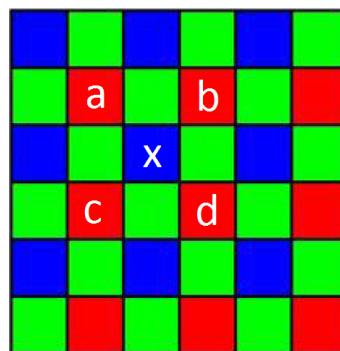
Tomando como referencia la ilustración 27 se puede ver que el cálculo recién descrito equivale a calcular el valor en el canal rojo para el píxel x. Lo cual consistiría en:

- Hacer el promedio entre a y b y luego, ubicar ese valor para el canal rojo para el píxel ubicado arriba de x.
- Hacer el promedio entre c y d y luego, ubicar ese valor para el canal rojo para el píxel ubicado debajo de x.
- Una vez obtenidos estos dos valores, promediarlos y ubicarlos en el canal rojo de x.

Llegando así a que realizar este cálculo es equivalente a realizar el promedio entre los valores de sus diagonales:

$$\frac{\frac{a+b}{2} + \frac{c+d}{2}}{2} = \frac{a+b+c+d}{4}$$

Figura 27:



### 2.3. Interpolaciones Direccionales

Se trata de una gran cantidad de métodos que buscan comenzar a resolver el problema de demosaicing realizando inicialmente una interpolación en una dirección y combinando con algún criterio determinado el resultado estimado con el obtenido de una aproximación realizada en otra dirección.

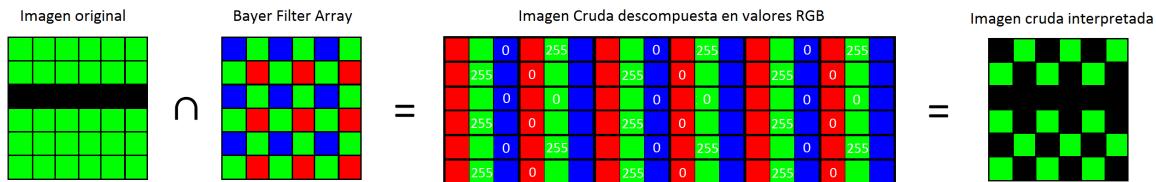
La implementación de este tipo de métodos exige tomar al menos dos decisiones de peso que distinguirán a unos de otros:

- De qué forma se realizarán las interpolaciones (en qué direcciones y mediante qué mecanismo de interpolación)
- Con qué criterio y consideraciones combinar la información proporcionada por las direcciones escogidas.

Para escoger alguna opción apropiada corresponde analizar las características del Color Filter Array con el que se va a trabajar, puesto que no todas las direcciones van a aportar en la totalidad de los casos la misma información.

En cuanto a la combinación de los resultados de las distintas interpolaciones, es conveniente tener presente que el promedio de los mismos no es siempre una buena opción. Por ejemplo, supongamos que se analiza una imagen que presenta en un sector las características observables en la Imagen original de la ilustración 28.

Figura 28:



En dicha figura se analiza la información de la misma que sería captada por los fotosensores respondiendo al Bayer Filter Array.

En el esquema 29 se puede observar que para varios píxeles la interpolación horizontal retorna valores distintos a la realizada verticalmente (esto tiene sentido, pues se trata de un área homogénea atravesada por una fila de distinto color).

El resultado de empleo del promedio como estrategia de nivelación de dichos valores queda expuesto en la Figura 30, en la cual se puede observar el causante del artifact “zipping”, producto de un algoritmo indiferente ante los cambios de color abruptos.

Figura 29:

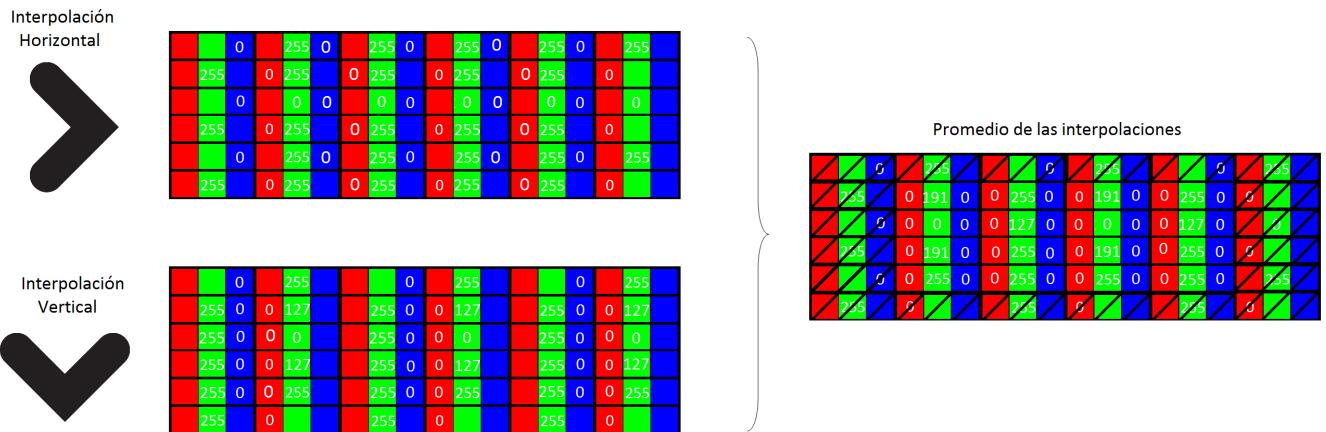


Figura 30:



Ciertamente esto no es lo esperado y existen mecanismos que permiten corregir este error. Uno de ellos consiste en calcular el gradiente del color para utilizarlo al combinar los resultados, de modo tal que si el mismo fuera de gran magnitud (es decir, la variación del color es considerable) entonces el peso que debe tener el valor calculado en el resultado final debería ser menor a aquel que fue calculado en una dirección cuyo gradiente fuera menor (es decir, que se tratara de una superficie más cromáticamente homogénea).

## 2.4. Splines

### Generación de los Splines

No se si estan de acuerdo, pero no me parecería mal dividir un poco la intro teórica en este TP, y hablar acá de por qué no usamos Lagrange. Si no les parece da igual, acá se puede hacer referencia a eso. Y listo (es lo que voy a dejar escrito ahora) pero hay que meditar qué queda mejor.

Hablar de todos sus tipos

Decir que bilineal y poner 0.5 queda igual (o casi igual je), usar psnr.

Y ver cual vamos a dejar

Y que usamos NATURALES

Yendo un paso más allá de la interpolación bilineal, podemos preguntarnos qué otros métodos existen que nos permitan obtener mejores resultados. Una buena primera idea sería analizar qué sucede con los Splines.

Los Splines (o trazadores cúbicos) son un tipo particular de interpoladores fragmentarios: funciones partidas que interpolan a través de polinomios un conjunto de puntos. En el caso de los Splines, se interpola entre dos puntos mediante un polinomio cúbico.

Como fue explicado anteriormente, los Splines son una opción muy buena a la hora de interpolar funciones: en el análisis realizado contra el Polinomio de Lagrange pudimos observar que este último puede llegar a ser excesivamente malo en ciertos casos. Considerando la superioridad de los Splines, entonces, nos propusimos encontrar un algoritmo que los utilice para conseguir buenas interpolaciones que generen imágenes de calidad superior a la hora de realizar el demosaicing.

Conseguir el Spline interpolador de una función  $f$  dado un conjunto de puntos  $\{X_0, \dots, X_n\}$  y sus respectivos valores de  $f(x_j)$  consiste en resolver un sistema  $Ax = b$  que nos permita encontrar los 4 coeficientes de cada uno de los  $n - 1$  polinomios que lo conformen. Siguiendo la lógica presentada en [3], donde se despeja el coeficiente que acompaña al término de grado 2 en cada polinomio, se puede llegar a ver que  $A$  resulta ser una matriz tridiagonal de  $(n + 1) \times (n + 1)$  que posee el siguiente aspecto (trabajando con splines naturales):

$$A = \begin{pmatrix} 1 & & & & \\ h_0 & 2 \cdot (h_0 + h_1) & h_1 & & \\ & h_1 & 2 \cdot (h_1 + h_2) & h_2 & \\ & & \ddots & & \\ & & & h_j & 2 \cdot (h_j + h_{j+1}) & h_{j+1} \\ & & & & \ddots & \\ & & & & & h_{n-2} & 2 \cdot (h_{n-2} + h_{n-1}) & h_{n-1} \\ & & & & & & & 1 \end{pmatrix}$$

Donde  $h_j$  es igual a la distancia entre los puntos  $X_j$  y  $X_{j+1}$ .

Mientras tanto,  $b$  será un vector de la forma:

$$b_i = \begin{cases} 0 & i = 0 \vee i = n \\ \frac{3 \cdot (f(x_{i+1}) - f(x_i))}{h_i} - \frac{3 \cdot (f(x_i) - f(x_{i-1}))}{h_{i-1}} & 1 \leq i \leq n - 1 \end{cases}$$

Para nuestro problema en particular, vamos a considerar las funciones a interpolar como el valor de un canal de RGB en alguna porción de la imagen. Dada la fisonomía del Bayer Filter Array, puede observarse con facilidad que, tomando una fila o una columna cualesquiera, la distancia que hay entre dos píxeles del mismo color es de 2 unidades (es decir, para un pixel azul en la posición  $j$  de una fila, el siguiente

pixel azul se va a encontrar en la posición  $j + 2$ ). Esto incluso aplica a si tomamos diagonales (no las que contienen exclusivamente verde, aunque dicho caso no nos es de interés ya que no se interpolaría el verde sobre la misma).

Por esta razón, podemos reemplazar  $h_j$  por 2, para todo  $j$ , y obtener:

$$A = \begin{pmatrix} 1 & & & \\ 2 & 8 & 2 & \\ & \ddots & & \\ & & 2 & 8 & 2 \\ & & & \ddots & \\ & & & & 2 & 8 & 2 \\ & & & & & 1 \end{pmatrix}$$

$$b_i = \begin{cases} 0 & i = 0 \vee i = n \\ \frac{3 \cdot (f(x_{i+1}) - 2 \cdot f(x_i) + f(x_{i-1}))}{2} & 1 \leq i \leq n-1 \end{cases}$$

Este sistema tiene la particularidad de que  $A$  es estrictamente diagonal dominante, y por lo tanto va a tener solución única, con lo cual sabemos que el método interpolador funcionará correctamente.

Una vez resuelto el mismo, sólo queda despejar los coeficientes principales y el de grado 1 de cada polinomio. El algoritmo en su totalidad se describe a continuación:

---

#### Algoritmo generar\_spline

---

**input** : Vector F

**output**: Vector S

n = size(F)  
 Matriz A(n)  
 Vector S(4 · (n - 1))

A(0, 0) = 1  
 A( $n - 1, n - 1$ ) = 1  
 b(0) = 0  
 b( $n - 1$ ) = 0

**for** cada fila  $i$  de la matriz A **do**  
 | A( $i, i - 1$ ) = 2  
 | A( $i, i$ ) = 8  
 | A( $i, i + 1$ ) = 2  
 | b( $i$ ) =  $\frac{3 \cdot (F(x_{i+1}) - 2 \cdot F(x_i) + F(x_{i-1}))}{2}$

Resolver el sistema  $Ax = b$  y guardarlo en el vector  $sol$

**for** cada elemento  $i$  del vector sol **do**  
 | S( $i$ ) = F( $i$ )  
 | S( $(i \cdot 4) + 1$ ) =  $\frac{f(x_{i+1}) - f(x_i)}{2} - \frac{2 \cdot sol(i+1) + 4 \cdot sol(i)}{3}$   
 | S( $(i \cdot 4) + 2$ ) =  $sol(i)$   
 | S( $(i \cdot 4) + 3$ ) =  $\frac{sol(i+1) - sol(i)}{6}$

---

Retornar S

---

Notar que  $F$  representa un vector que contiene los valores de  $f(X_j)$  que queremos interpolar. Por otro lado, el conjunto de puntos  $\{X_0, \dots, X_n\}$  está implícito, ya que, como la distancia entre ellos es siempre 2, no es necesario saber su valor para realizar la interpolación: los consideramos enumerados de 0 a  $n$ .

Por su parte,  $S$  es un vector que contiene los cuatro coeficientes de cada polinomio, de corrido, ordenados de término independiente a coeficiente principal.

Creemos importante comentar acerca de la estructura elegida para las matrices utilizadas: como las matrices que resuelven el problema de la interpolación son siempre matrices tridiagonales, nos pareció una buena idea aprovechar el hecho de que son esparsas para poder trabajar con ellas más eficientemente en términos de memoria y complejidad algorítmica. Para lograr dicho objetivo, procedimos a reutilizar, haciendo algunas modificaciones menores, los algoritmos utilizados en el primer Trabajo Práctico.

La estructura elegida, entonces, es un simple vector de vectores, que sólo almacena 4 columnas: las 3 diagonales y el vector resultado.

Ya generado el Spline, sólo resta evaluar los puntos que nos interesen para conseguir una estimación de  $f$  en los mismos. Para ello, primero se localiza el polinomio cúbico  $S_j$  correspondiente en el vector  $S$  generado (de acuerdo al intervalo que pertenezca el  $X_i$  que se quiere estimar) y se resuelve:

$$f(X_i) \approx a + b \cdot (X_i - X_j) + c \cdot (X_i - X_j)^2 + d \cdot (X_i - X_j)^3$$

Donde  $a, b, c$  y  $d$  son los coeficientes conseguidos a través del algoritmo `generar_spline`, y  $X_j$  es el valor del punto que se corresponde con el polinomio  $S_j$  a utilizar.

El algoritmo que resuelve este problema fue implementado de la siguiente manera:

---

#### Algoritmo evaluar

---

**input** : Vector  $S$ , Int  $x$   
**output**: Double  $y$

Int  $j = \lfloor \frac{x}{2} \rfloor$   
 Int  $i = 4 \cdot j$

Double  $a = S(i)$   
 Double  $b = S(i + 1)$   
 Double  $c = S(i + 2)$   
 Double  $d = S(i + 3)$

$j = 2 \cdot j$

Retornar  $a + b \cdot (x - j) + c \cdot (x - j)^2 + d \cdot (x - j)^3$

---

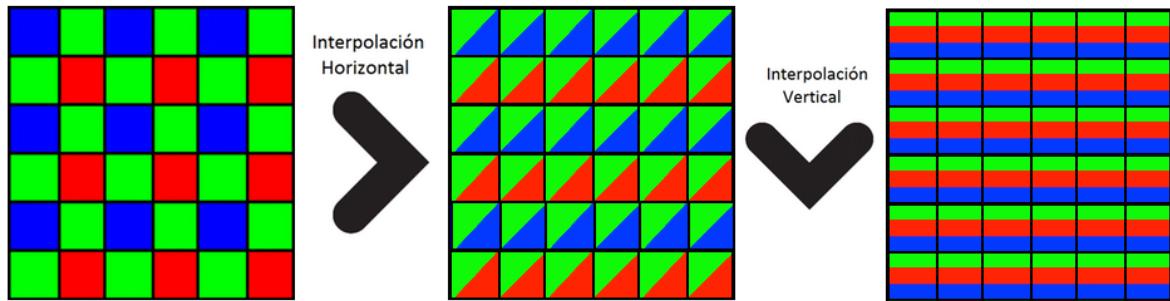
Ver de explicar un poco mejor el tema de las cuentas con los indices

Vale la pena aclarar que el algoritmo recibe como input un entero ya que cada uno de nuestros  $X_i$  es un pixel, y no tendría sentido trabajar con fracciones de los mismos.

#### Aplicación de Splines al problema de demosaicing

Una vez resuelto el problema de la implementación de los algoritmos que generan el spline e interpolan valores en el mismo, el siguiente paso fue pensar cómo aplicarlo al problema que nos concierne: el *demosaicing* de imágenes. Diversos métodos fueron considerados y estudiados, y serán presentados a continuación.

Figura 31: cómo se realiza la interpolación



El primer acercamiento que tuvimos con Splines fue simple: para cada pixel, interpolar su fila y su columna (por completo) para los canales faltantes, y asignarle el promedio entre ambos valores obtenidos. Este método aparenta ser muy similar a la interpolación bilineal, donde se asigna el promedio de los valores de sus vecinos, razón por la cual decidimos comparar cualitativamente los outputs de ambos algoritmos:

- A simple vista, las imágenes son prácticamente idénticas para ambos métodos. Las diferencias son demasiado mínimas como para ser realmente apreciadas. La versión con splines mostró una mayor presencia de artifacts (principalmente *false colors*) pero no muy notable.
- Objetivamente, las mediciones de PSNR arrojaron valores sorpresivamente distintos (alrededor de 1 punto de diferencia), favoreciendo a la interpolación bilineal.

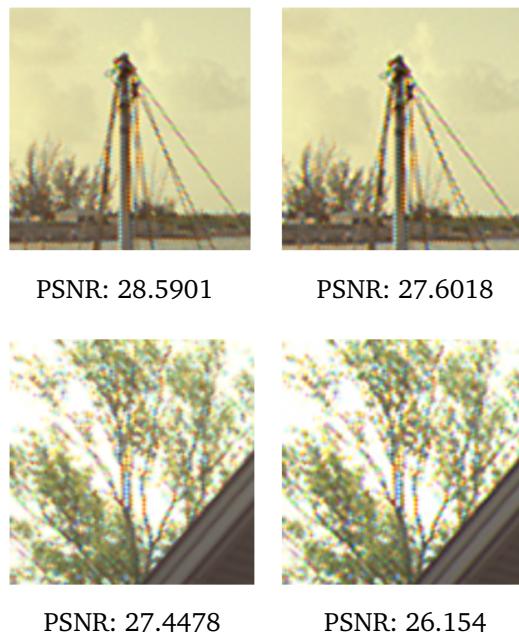


Figura 32: Interpolación bilineal (izquierda), y mediante esta implementación de Splines (derecha)

Respecto al tiempo de ejecución de ambos algoritmos, **MEDIR TIEMPOS POR SSH**

En vista de estos resultados, decidimos descartar (o mejor dicho, intentar mejorar) el algoritmo mediante Splines, ya que ni en tiempo ni en calidad lo encontramos superior que los métodos anteriores.

El primer aspecto con el que decidimos experimentar fue la manera de combinar la información obtenida a través de las interpolaciones. En el caso anterior, simplemente realizábamos el promedio de los valores obtenidos al interpolar una fila y una columna. Sin embargo, este acercamiento no es siempre bueno, ya que puede traer resultados no deseados en los bordes. Es por esta razón que decidimos proponer una idea más sofisticada, detallada brevemente en la sección de interpolación bilineal: el uso del gradiente.

A modo de ejemplo, la derivada horizontal en el canal verde del pixel puede estimarse de la siguiente manera:

$$|\partial_x G(x, y)| \approx |G(x - 1, y) - G(x + 1, y)|$$

Y, análogamente, la vertical:

$$|\partial_y G(x, y)| \approx |G(x, y - 1) - G(x, y + 1)|$$

Utilizando este concepto, modificamos nuestro algoritmo para que, en lugar de combinar la información obtenida mediante el promedio, haga lo siguiente:

- Interpolar por filas
- Al realizar la interpolación por columnas, para los pixeles rojos y azules tenemos ahora dos valores posibles de verde. Comparamos los valores de las derivadas horizontal y vertical para el color verde:
  - Si la derivada horizontal es mayor que la vertical, entonces le asignamos el valor de verde dado por la interpolación de la columna
  - Si la derivada vertical es mayor que la horizontal, entonces le asignamos el valor de verde dado por la interpolación de la fila

El algoritmo que resuelve esto fue implementado de la siguiente manera:

---

#### Algoritmo spline\_der

---

```
for cada fila de la imagen do
    if la fila es par then
        Generar el spline de toda la fila para el canal azul e interpolar para los píxeles verdes
        Generar el spline de toda la fila para el canal verde e interpolar para los píxeles azules
    else
        Generar el spline de toda la fila para el canal rojo e interpolar para los píxeles verdes
        Generar el spline de toda la fila para el canal verde e interpolar para los píxeles rojos

for cada columna de la imagen do
    if la columna es par then
        Generar el spline de toda la columna para el canal rojo e interpolar para los píxeles azules
        Generar el spline de toda la columna para el canal azul e interpolar para los píxeles verdes
        if  $\partial_x G > \partial_y G$  then
            Generar el spline de toda la columna para el canal verde e interpolar para los píxeles
            azules, reemplazando el valor que había sido asignado en el primer ciclo
    else
        Generar el spline de toda la columna para el canal azul e interpolar para los píxeles rojos
        Generar el spline de toda la columna para el canal rojo e interpolar para los píxeles verdes
        if  $\partial_x G > \partial_y G$  then
            Generar el spline de toda la columna para el canal verde e interpolar para los píxeles
            rojos, reemplazando el valor que había sido asignado en el primer ciclo
```

---

Aplicando este método, pudimos observar mejoras en varios de los bordes de nuestras imágenes: los mismos ahora son más suaves, eliminando casi por completo la presencia de varios artifacts, como por

ejemplo zipper.

Sin embargo, también hubo algunos aspectos en los cuales la imagen final empeoró: algunas de las líneas más finas mostraron *false colors*, muchas veces con la presencia de algunos píxeles de colores muy fuertes.

Respecto a las mediciones de calidad objetivas, los resultados arrojados por PSNR fueron totalmente inesperados: a pesar de ser una diferencia muy chica, PSNR siempre fue superior para la imagen procesada con el método del promedio. Creemos que esto puede deberse a la anormalidad de los colores para ciertos píxeles mencionada en el párrafo anterior, ya que PSNR mide cuánto “ruido” hay entre la imagen original y la procesada.

De todas maneras, consideramos al método de Splines pesados mediante las derivadas como superior al del promedio, ya que, a la vista, es más notorio que los bordes sean suaves, a que haya algunos pocos píxeles dispersos con colores no muy fieles.

Respecto al tiempo de cómputo, **MEDIR TIEMPOS POR SSH**



Figura 33: Combinación de datos mediante promedio (izquierda), y mediante el presente método (derecha)

En estas imágenes se evidencia la mejora que brinda el método de las derivadas sobre los bordes de las imágenes. En el primer caso, el texto aparece mucho más nítido, y el *zipper effect* que se podía percibir en los bordes de la franja marrón del barco es mucho menos visible.

La segunda imagen, por su parte, también muestra notables mejoras en la ventana: combinando los datos mediante el promedio, podía observarse claramente *zipper effect* y *jaggies* en los bordes internos de la misma; pero al procesar la imagen con el algoritmo que hace uso de las derivadas, las líneas ahora aparecen mucho más suaves y definidas.

De todas formas, sigue siendo visible el artifact de Moiré, y pueden verse algunos de los píxeles anormalmente fuertes de los que hablamos antes.



Figura 34: Combinación de datos mediante promedio (izquierda), y mediante el presente método (derecha)

En estas otras dos imágenes pretendemos dejar en evidencia el problema de los píxeles con colores incorrectos. Como puede verse en ambas, las líneas en el caso del segundo método parecen más definidas y con una alta reducción del *zipper effect* y *jaggies*. Sin embargo, puede observarse la presencia de píxeles, sobre todo verdes y rojos, prácticamente saturados, sobre los bordes más finos.

Intentando encontrar explicaciones para este fenómeno, concluimos que podía deberse a que estábamos dejando de lado información valiosa a la hora de componer la imagen: sólo usábamos una dirección.

Por esta razón, decidimos no arrojar una de las direcciones, sino buscar alguna manera de utilizar ambas para obtener mejoras en la composición.

El siguiente método surgió entonces como un intento de mejorar el anterior algoritmo.

El mismo consiste en, para cada pixel que no sea verde, calcular los valores resultantes de ambas interpolaciones para el canal verde: tanto la de la fila, como la de la columna, y luego multiplicar a cada uno por algún factor relacionado con las derivadas direccionales de ese mismo canal, en el pixel a estimar.

Luego de un análisis sobre qué factor era más conveniente usar, llegamos a la conclusión de que uno inversamente proporcional al valor que tenga cada derivada sería una buena idea. Por lo tanto, definimos:

$$\sigma_G = \partial_x G + \partial_y G$$

$$\alpha_G = \begin{cases} 0,5 & \sigma_G = 0 \\ 1 - \frac{\partial_x G}{\sigma_G} & \sigma_G \neq 0 \end{cases}$$

$$\beta_G = \begin{cases} 0,5 & \sigma_G = 0 \\ 1 - \frac{\partial_y G}{\sigma_G} & \sigma_G \neq 0 \end{cases}$$

En términos implementativos, el algoritmo que resuelve este problema es el siguiente:

---

Algoritmo spline\_der (proporcional)

---

```
for cada fila de la imagen do
    if la fila es par then
        ▪ Generar el spline de toda la fila para el canal azul e interpolar para los píxeles verdes
        ▪ Generar el spline de toda la fila para el canal verde e interpolar para los píxeles azules
        ▪ Multiplicar al resultado de la interpolación anterior por  $\alpha$ 
    else
        ▪ Generar el spline de toda la fila para el canal rojo e interpolar para los píxeles verdes
        ▪ Generar el spline de toda la fila para el canal verde e interpolar para los píxeles rojos
        ▪ Multiplicar al resultado de la interpolación anterior por  $\alpha$ 

for cada columna de la imagen do
    if la columna es par then
        ▪ Generar el spline de toda la columna para el canal rojo e interpolar para los píxeles azules
        ▪ Generar el spline de toda la columna para el canal azul e interpolar para los píxeles verdes
        ▪ Generar el spline de toda la columna para el canal verde e interpolar para los píxeles azules
        ▪ Multiplicar al resultado de la interpolación anterior por  $\beta$ 
        ▪ Sumarlo al valor previo del pixel en el canal verde
    else
        ▪ Generar el spline de toda la columna para el canal azul e interpolar para los píxeles rojos
        ▪ Generar el spline de toda la columna para el canal rojo e interpolar para los píxeles verdes
        ▪ Generar el spline de toda la columna para el canal verde e interpolar para los píxeles rojos
        ▪ Multiplicar al resultado de la interpolación anterior por  $\beta$ 
        ▪ Sumarlo al valor previo del pixel en el canal verde
```

---

Las imágenes resultantes luego de ser procesadas con el presente algoritmo no produjeron los resultados esperados: las mejoras fueron extremadamente sutiles, al borde de ser indetectables a simple vista, y arrojando resultados de PSNR y SSIM prácticamente idénticos.

### IMAGENES

Aún más, este algoritmo es considerablemente más lento que el anterior. Teniendo en cuenta el contexto de aplicación, esto no es para nada aceptable: idealmente, un algoritmo de demosaicing debería ser lo suficientemente rápido -o al menos producir imágenes de calidad realmente superior- para justificar el tiempo de cómputo extra requerido.

### GRAFICO TIEMPO

En conclusión, decidimos descartar este algoritmo: las mejoras eran demasiado sutiles y el tiempo inaceptablemente mayor como para preferirlo.

Ya habiendo probado distintas formas de mejorar el algoritmo original (interpolación de una fila y columnas completas, combinando los resultados mediante promedio) y sin obtener resultados considerablemente mejores, decidimos cambiar, no sólo la forma de combinar los resultados, sino también la manera de obtenerlos: nos propusimos dejar de interpolar una fila y columnas completas, y optar por tomar “rangos” alrededor de cada pixel sobre los cuales hacer la interpolación.

Nuestra hipótesis al plantear esto fue la siguiente: el color de un píxel está definido por un entorno del mismo, es decir, habrá todo un entorno del pixel en donde los colores serán similares (excepto en los bordes). Tiene más sentido estimar el color de un píxel sólo tomando los valores que nos brindan información “cierta”, y no estimar mediante un polinomio que tiene en cuenta la variación del color a lo largo y ancho de toda la imagen, que puede contener muchos bordes y cambios cromáticos.

Para este método seguiremos utilizando la información que nos brindan las derivadas direccionales, más específicamente, utilizar sólo la interpolación de una fila o una columna (en lugar de la técnica proporcional de los pesos), dado que la experimentación antes realizada demostró que el *trade-off* calidad-tiempo beneficia al primero.

Además, dado que este método realiza interpolaciones en rangos mucho más chicos (y, por ende, genera Splines mucho más pequeños), esto nos permitió repensar la manera en la que se recorre la imagen, para así intentar realizar una interpolación de mejor calidad para todos los canales: notar que en los métodos explicados hasta el momento, los canales rojo y azul de los píxeles originalmente verdes sólo eran interpolados con la información de las filas (para el canal rojo) o las columnas (para el canal azul), es decir, sólo se interpolaba en una dirección para los colores distintos al verde. Esto se debe a que hubiera sido muy costoso volver a recorrer la imagen, interpolando nuevamente filas y columnas enteras, para interpolar estos canales en las direcciones faltantes.

Aún más, dado que ahora le daremos peso a las direcciones de los tres canales por igual, se aplicará el mismo concepto del gradiente para los canales faltantes, notando  $\partial_x B$  y  $\partial_y B$  a las derivadas horizontal y vertical para el canal azul, y  $\partial_x R$  y  $\partial_y R$  a las derivadas horizontal y vertical para el canal rojo.

El algoritmo que implementa esta idea se presenta a continuación:

---

Algoritmo spline\_rango

---

```
for cada fila de la imagen do
    if el pixel actual es azul o rojo then
        Generar un spline de rango  $r$  para el canal verde de acuerdo a  $\partial_x G$  y  $\partial_y G$  e interpolar
    else if el pixel actual es verde then
        if es una fila de verdes y rojos then
            Generar un spline horizontal de rango  $r$  para el canal rojo e interpolar
            Generar un spline vertical de rango  $r$  para el canal azul e interpolar
        else
            Generar un spline horizontal de rango  $r$  para el canal azul e interpolar
            Generar un spline vertical de rango  $r$  para el canal rojo e interpolar

for cada fila de la imagen do
    if el pixel actual es azul then
        Generar un spline de rango  $r$  para el canal rojo de acuerdo a  $\partial_x R$  y  $\partial_y R$  e interpolar
    else if el pixel actual es rojo then
        Generar un spline de rango  $r$  para el canal azul de acuerdo a  $\partial_x B$  y  $\partial_y B$  e interpolar

for cada fila de la imagen do
    if es una fila de verdes y azules, y el pixel actual es verde then
        if  $\partial_y R > \partial_x R$  then
            Generar un spline horizontal de rango  $r$  para el canal rojo e interpolar
        if  $\partial_x B > \partial_y B$  then
            Generar un spline vertical de rango  $r$  para el canal azul e interpolar
    else if es una fila de verdes y rojos, y el pixel actual es verde then
        if  $\partial_y B > \partial_x B$  then
            Generar un spline horizontal de rango  $r$  para el canal azul e interpolar
        if  $\partial_x R > \partial_y R$  then
            Generar un spline vertical de rango  $r$  para el canal rojo e interpolar
```

---

Donde “generar un spline de rango  $r$  para el canal  $C$  de acuerdo a  $\partial_x C$  y  $\partial_y C$  e interpolar” significa “calcular la dirección cuya derivada es mayor y generar el spline correspondiente para usar en la interpolación”, como en los métodos anteriores.

- La primera iteración sobre la imagen se encarga de completar el canal verde para los píxeles distintos del verde, y, para los verdes, completar los canales rojo y azul, sólo en la dirección que tenga información acerca de ese canal (dependerá de la fila en la que se encuentre el pixel verde en cuestión).
- La segunda iteración se encargará de completar el canal rojo para los píxeles originalmente azules, y el canal azul para los píxeles originalmente rojos. Notar que podemos utilizar las derivadas direccionales también para estos colores porque en el ciclo anterior nos encargamos de que haya información acerca del rojo y el azul en todos los píxeles verdes.
- Por último, el tercer ciclo se encarga de mejorar la interpolación de los canales rojo y azul para los píxeles verdes. En principio, a los píxeles verdes se les había asignado información del rojo y el azul en una sola dirección, pero ahora, aprovechando que tenemos información del rojo y el azul en todos los píxeles, podemos utilizar el criterio del gradiente para mejorar la estimación.

## 2.5. Algoritmo de Malvar, He y Cutler

Luego de la lectura y estudio de la publicación *HIGH-QUALITY LINEAR INTERPOLATION FOR DEMOSAICING OF BAYER-PATTERNE COLOR IMAGES* [2], nos decidimos por experimentar con el algoritmo planteado y así observar las mejoras planteadas en este trabajo.

Malvar, He y Cutler presentan una crítica a los métodos que no utilizan los tres canales RGB en su conjunto, como por ej. la *interpolación Bilineal* a la cual le adjudica la generación de significantes *artifacts*. Y presentan un método de Demosaicing en el cual el estudio de cada canal no se hace de manera independiente a los otros dos, que mejora en aspectos de tiempo de cómputo a algoritmos similares.

Los tres canales en su conjunto contienen información sobre la luminiscencia de la imagen, es por esto que al calcular por ej. el valor verde de un píxel orginalmente rojo, se utiliza la información contenida en el rojo para la nueva asignación. Si el valor del píxel rojo difiere de una cantidad significante con sus estimaciones mediante la interpolación bilineal con sus vecinos, esto significa que hay un cambio abrupto en la luminiscencia y por esto se debe corregir el valor calculado en el canal verde de este píxel añadiéndole, en proporción, este cambio de luminiscencia.

El método se basa en una interpolación bilineal con una corrección respecto al gradiente usando un parámetro deseable para controlar cuanta corrección es aplicada.

El algoritmo consiste en utilizar una ventana de 5x5, es decir para estimar el valor de un píxel se utilizan los píxeles cercanos dentro del cuadrante con ancho de tamaño 5 píxeles. Siendo el que está situado en el centro el píxel que va a ser estimado.

El algoritmo se comporta de la siguiente manera: situado en un píxel a estimar, se calcula la derivada en su color nativo y luego al calcular los dos canales faltantes se suma esta derivada multiplicada por una constante. De este modo es como se nivela la luminiscencia.

Luego de programar el algoritmo y realizar casos de prueba, los resultados arrojados fueron mejores que para los algoritmos anteriores.

Poner fotitos y mediciones... no?

### 3. Resultados y discusión

#### 3.1. Exploración de Artifacts

Ver si hay otros artifacts mas para agregar.

Ver como arreglar el de Moire y bla...

Poner fotitos de ampliaciones donde se vean bien los artifacts.

EN MHC:

Visualmente, podemos definir que las fotos adquieren una mayor definición con contornos menos borrosos e imperfectos. Este algoritmo optimiza la detección de bordes cuando la diferencia de colores es muy fuerte, por ej. un borde oscuro sobre un fondo claro. Si se comparara con el algoritmo de interpolación bilineal, en el mismo caso se producían colores fuertes alrededor de los bordes negros (artifact).

Sim embargo, para los bordes con un menor contraste o bordes más claros, sigue viéndose este artifact donde aparecen colores artificiales muy fuertes en la zona de los mismos.

Poner recortes de bordes como estos nombrados arriba

### 3.2. Análisis cualitativo de los Métodos

Luego de haber generado las imágenes con todos los algoritmos que hemos propuesto, nos dispusimos a correr en Matlab® la diferencia (resta) entre ellas y la imagen original de la cual partimos. Luego de eso, invertimos los colores para poder apreciar mejor dónde se notaban las mayores diferencias. A continuación se incluyen las imágenes obtenidas con este proceso, en color blanco y negro con el objetivo de distinguir idóneamente las regiones de la imagen donde exista un error contundente.

Figura 35:

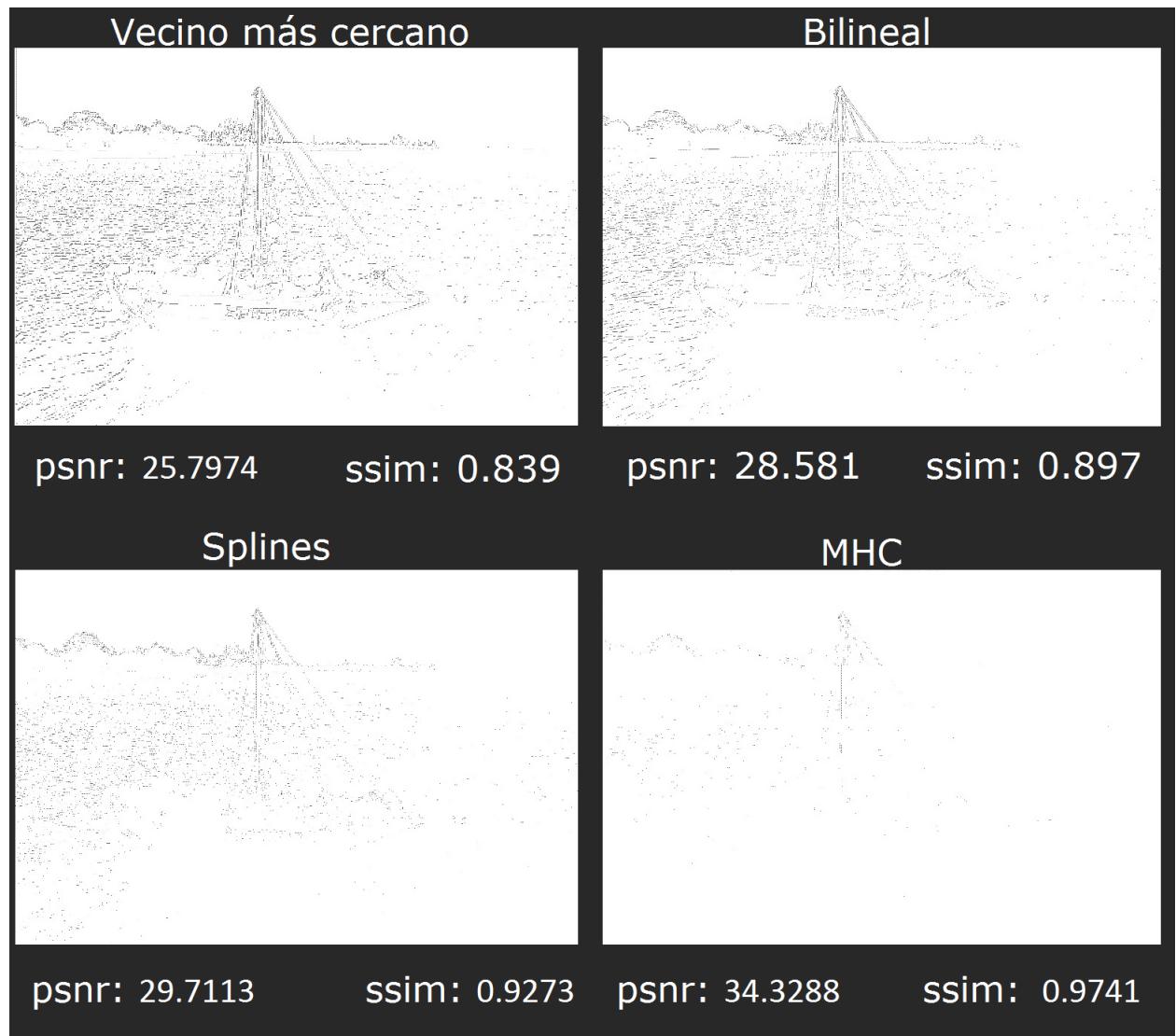


Figura 36: Imagen Original



En la Figura 35 se puede observar que, si bien al ejecutar el algoritmo de MHC disminuye abundantemente la diferencia con la imagen original, al tener una superficie en su mayoría de agua se dificulta a niveles algorítmicos aproximar el agua de una manera más real ya que esta no es para nada una superficie pareja. Estas diferencias tal vez no sean tan visibles al ojo humano, pero en cuanto a diferencia de color píxel a píxel es notorio.

También se hace presente el artifact que hay sobre la punta superior del mastil, dado que se puede observar la anomalía en los cuatro algoritmos. En un primer momento, se puede apreciar Zipper y Moiré (en el algoritmo de Vecino Más Cercano) y a medida que se van utilizando métodos “mejores” este artifact se reduce simplemente a un pequeño Moiré.

Figura 37: Zoom en imágenes obtenidas mediante los distintos métodos

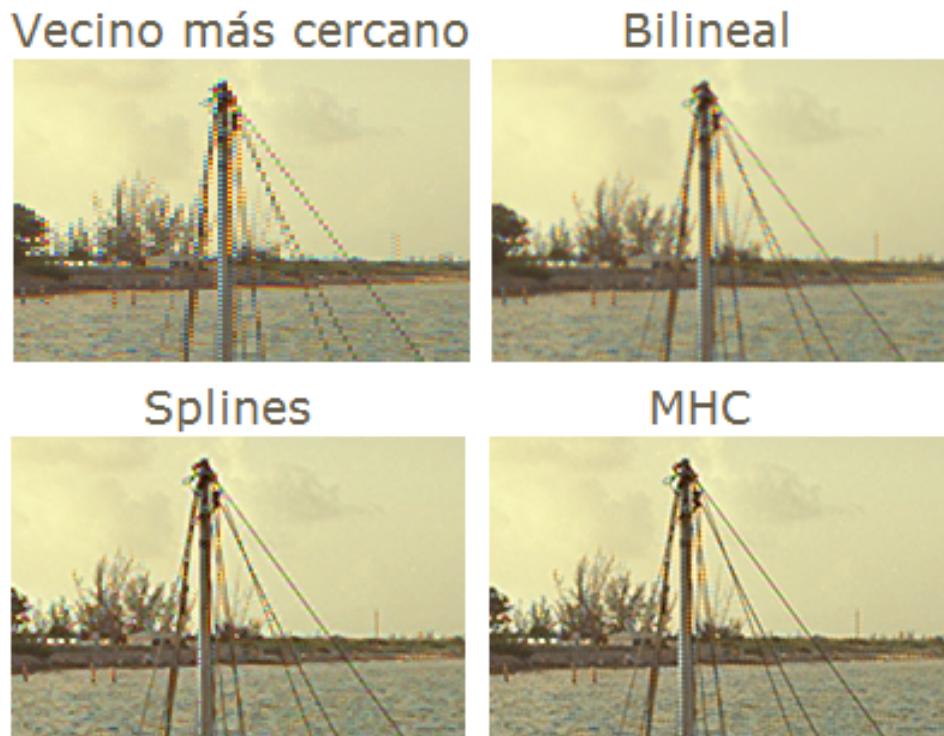
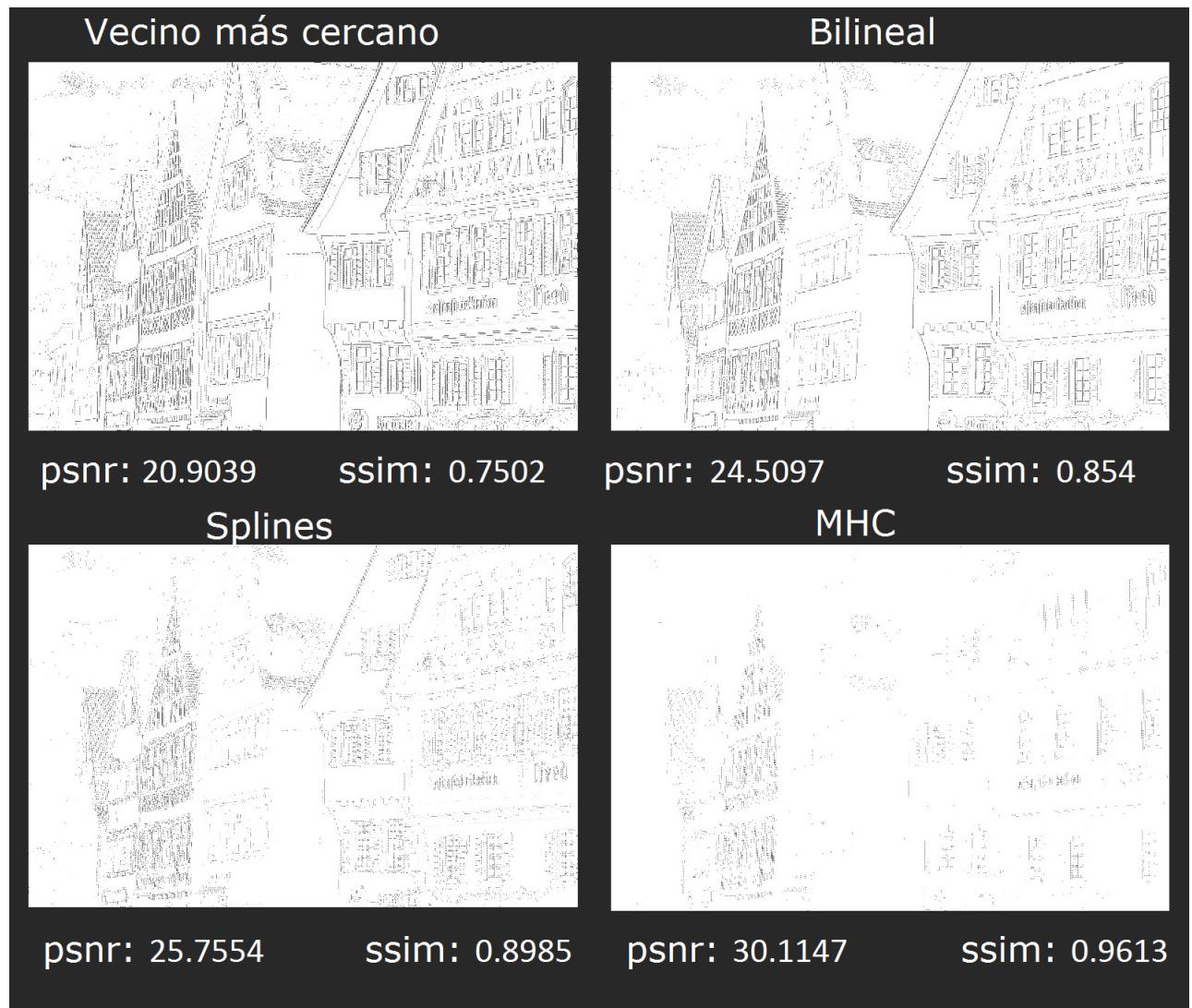


Figura 38: Imagen Original



Figura 39:



En la Figura 39 se puede apreciar el inconveniente de aproximar píxeles que son borde cuando estos son dados en abundancia y en líneas recta en su mayoría. Inclusive en el mejor caso de esta imagen, (la imagen creada mediante el Algoritmo de MHC) se puede apreciar una notable diferencia con la original en lo que respecta a los bordes.

Si ahondamos en la imagen, notamos que es en las ventanas donde perciste el problema para todos

los métodos. Si nos detenemos en la imagen obtenida se puede apreciar un artifact en específico: Moiré.

Figura 40: Zoom en imágenes obtenidas mediante los distintos métodos



El otro inconveniente presente se ubica en el letrero, el cual gradualmente adquiere mayor definición, pero así mismo persiste marcándose la diferencia con el original.

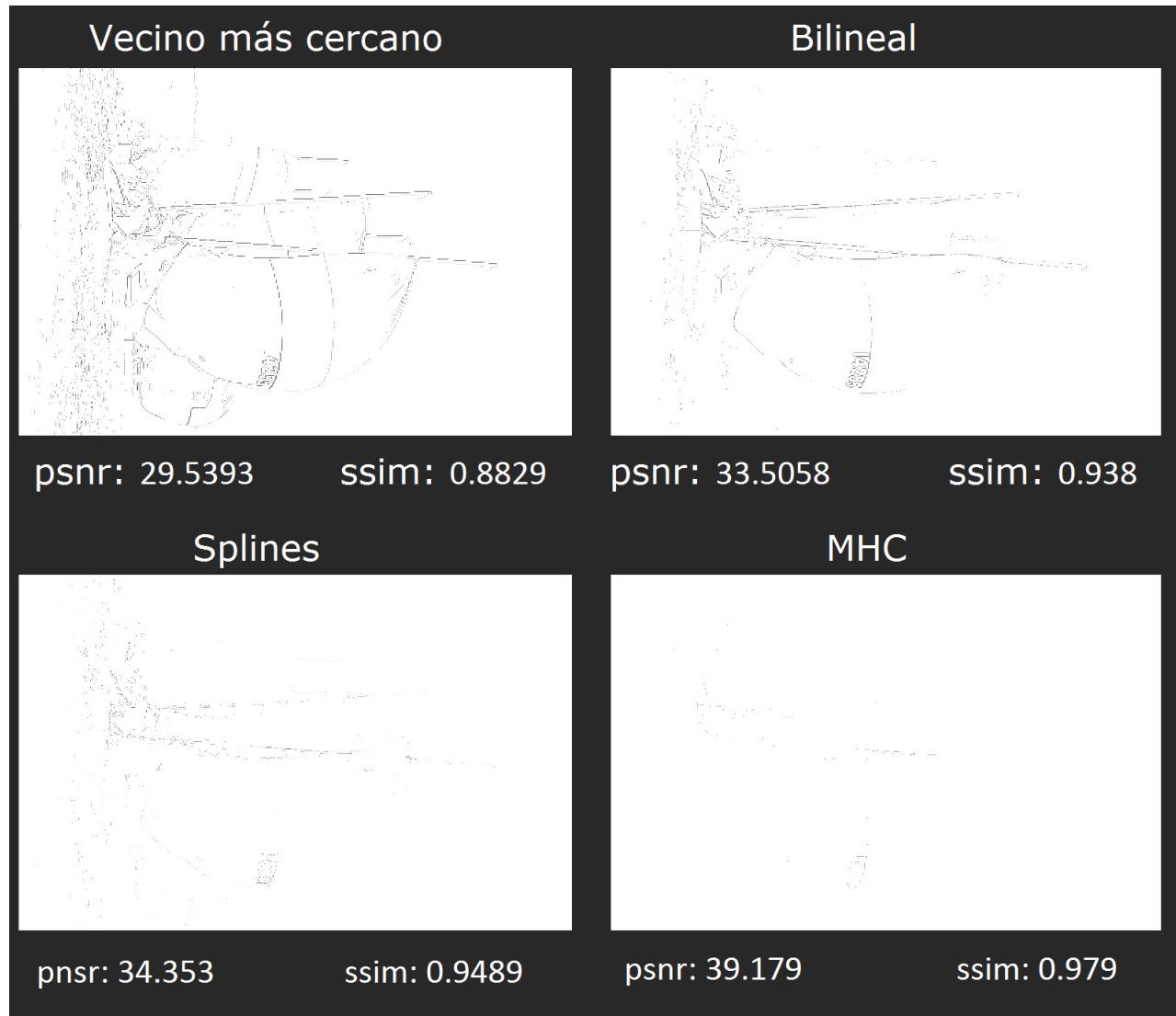
Figura 41: Zoom en imágenes obtenidas mediante los distintos métodos



Figura 42: Imagen Original



Figura 43:



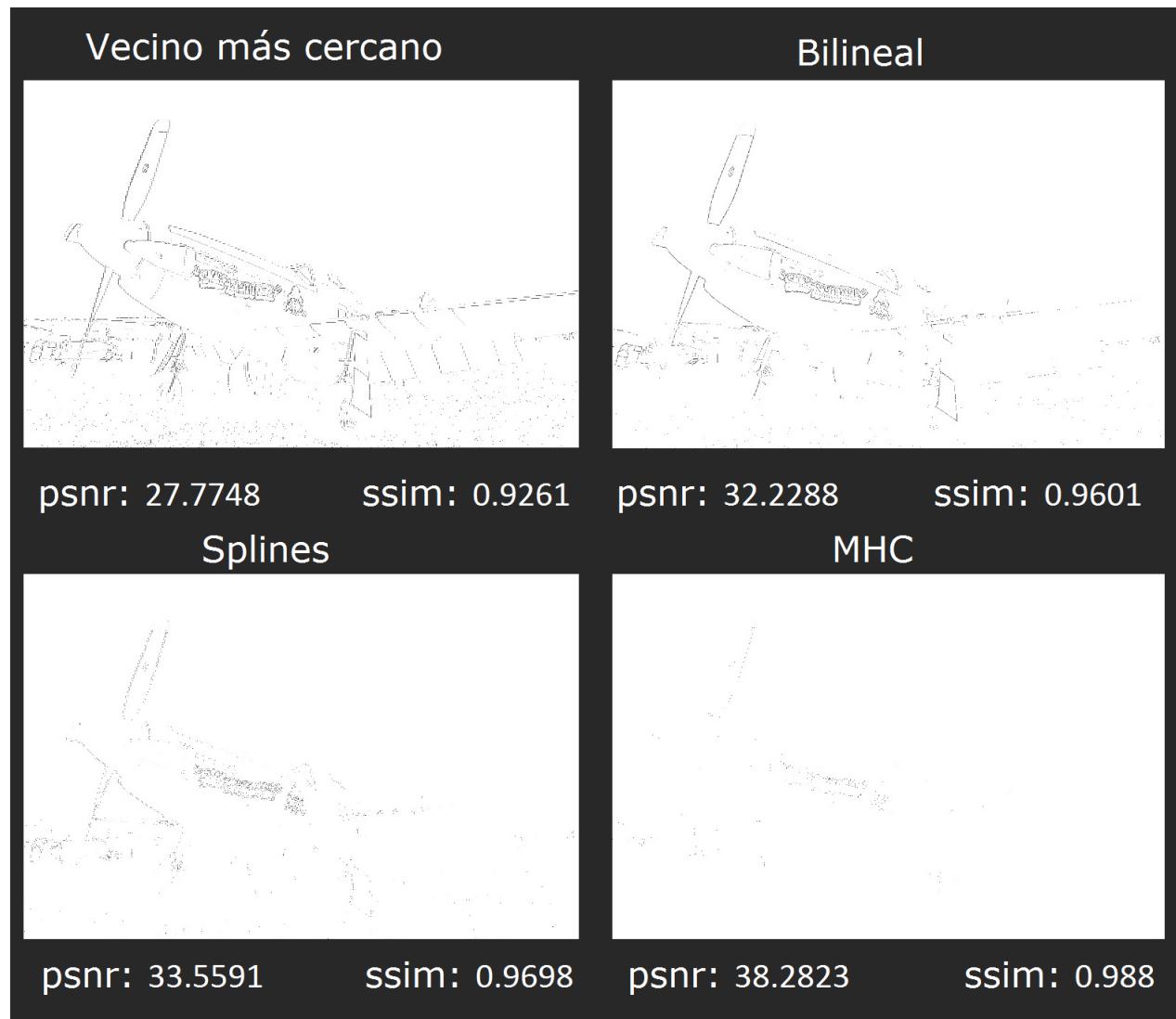
En cuanto a la Figura 43 se puede notar un comportamiento similar al de la imagen en la Figura 35 dado a la existencia de agua, la cual es una superficie difícil de estimar para los más básicos algoritmos.

Lo interesante a destacar de este caso, es el gran avance que se realiza mediante el algoritmo de MHC en contraposición de los otros dos. Se puede apreciar que las diferencias entre la imagen estimada por este algoritmo y la imagen original se convierten en ínfimas.

Figura 44: Imagen Original

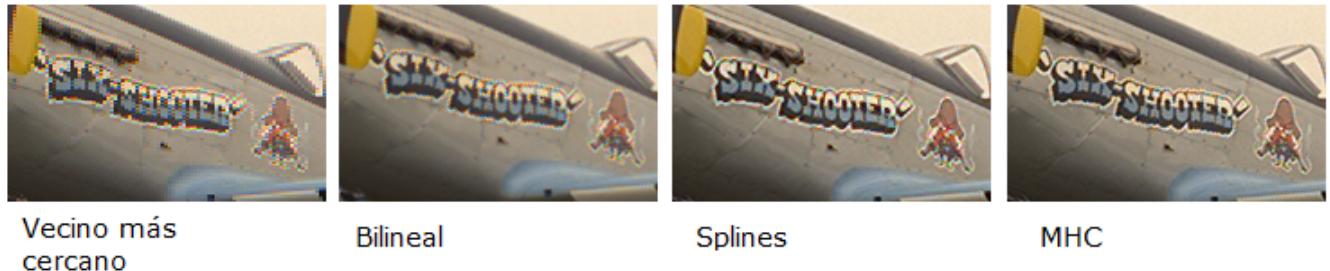


Figura 45:



En la Figura 45 se puede ver un comportamiento similar a lo ya estudiado, lo interesante de destacar en cuanto a esta foto es que al poseer un letrero es una imagen menos abstracta que las anteriores y por lo tanto, se adquiere una definición mayor cuando se tiene una diferencia menor con la original y así facilitar la lectura del cartel.

Figura 46: Zoom en imágenes obtenidas mediante los distintos métodos



Vecino más cercano

Bilineal

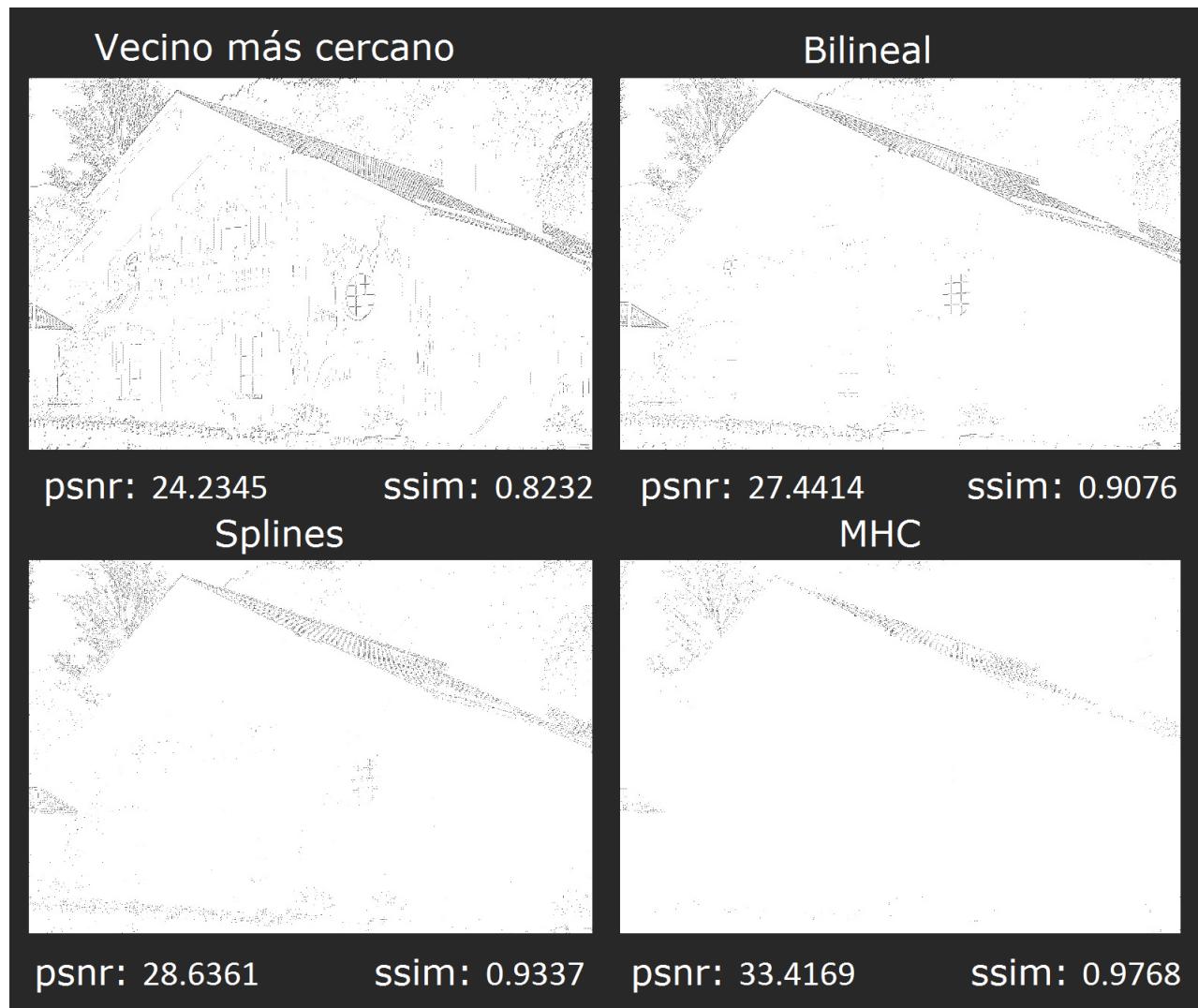
Splines

MHC

Figura 47: Imagen Original



Figura 48:



La Figura 50 es una imagen con muchas líneas y dibujos con contornos. Esto se aprecia en la gran densidad que tiene la imagen que muestra las diferencias entre la imagen original y la imagen obtenida mediante el algoritmo de Vecino Más Cercano. Si observamos las imágenes obtenidas mediante la resta con sus respectivas originales se denota una gran diferencia entre el Algoritmo de Vecino Más Cercano y el Algoritmo de MHC.

Figura 49: Zoom en imágenes obtenidas mediante los distintos métodos



Figura 50: Zoom en imágenes obtenidas mediante los distintos métodos



En la sección de la foto que le corresponde al techo se puede apreciar el artifact de *False Colors*. Ya que aparecen píxeles con tonos y colores que no corresponden a la imagen original. Se puede apreciar como, gradualmente, esta anomalía va desapareciendo otorgándole a la imagen mayor nitidez. Por este motivo, se podría decir también que MHC elimina el artifact *mblur*, al menos en este caso.

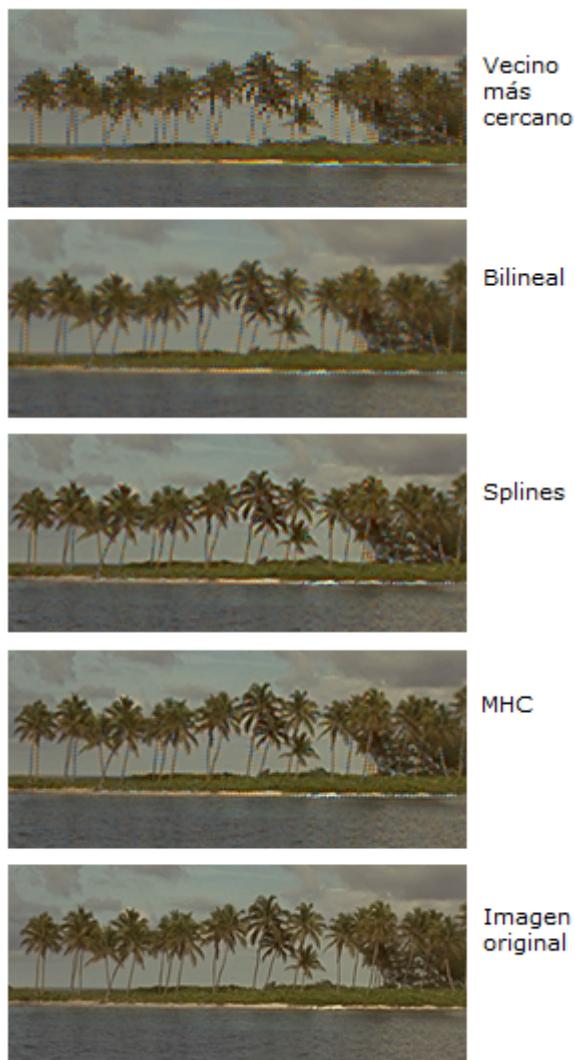
La ventana es un punto de análisis importante ya que las dos primeras estimaciones producen Zipper, mientras que con Splines disminuye hasta la casi inexistencia no llega a estimarse a la perfección. En el caso de MHC se podría decir que la ventana se ve con una similitud importante respecto de la imagen original.

Nos pareció un caso de interés lo ocurrido en la siguiente imagen:

Figura 51: Imagen Original



Figura 52:



En la Figura 52 cabe destacar dos aspectos:

- **Nitidez de los troncos de las palmeras:** En esta imagen se repite algo que observamos en casos anteriores. El algoritmo de Vecino más cercano produce Zipper notablemente y a medida que se avanza con los algoritmos, este artifact decremente su existencia. De este modo, lo que solía ser una línea inexacta y con colores pocos reales termina convirtiéndose en un tronco muy cercano en aspecto al de la imagen original.
- **False colors:** Algo llamativo que acontece con esta foto es la aparición de una espuma turquesa en la orilla. Si bien bajo el algoritmo de Vecino más cercano posee una pronunciación mayor ningún algoritmo es capaz de eliminarla del todo. Los resultados obtenidos difieren bastante visualmente de la imagen original. Esta aparición de un color que no existe es un caso más del artifact False Colors.

Como conclusión, podemos inferir que los lugares donde se muestra una variación son siempre lugares pertenecientes a contornos o bordes de objetos en las imágenes. En cambio, en píxeles sobre superficies llanas o parejas del mismo cuerpo la diferencia entre la imagen original y nuestra imagen obtenida es casi nula.

Y también se puede concluir que para todos los casos de test realizados, coincide que el algoritmo óptimo es el de MHC.

*Se adjuntan las imágenes en su tamaño original.*

En las siguientes figuras se puede observar como aumentan, acorde al método, los valores de psnr y ssim.

Figura 53: Evaluación de psnr para 12 imágenes del mismo tamaño

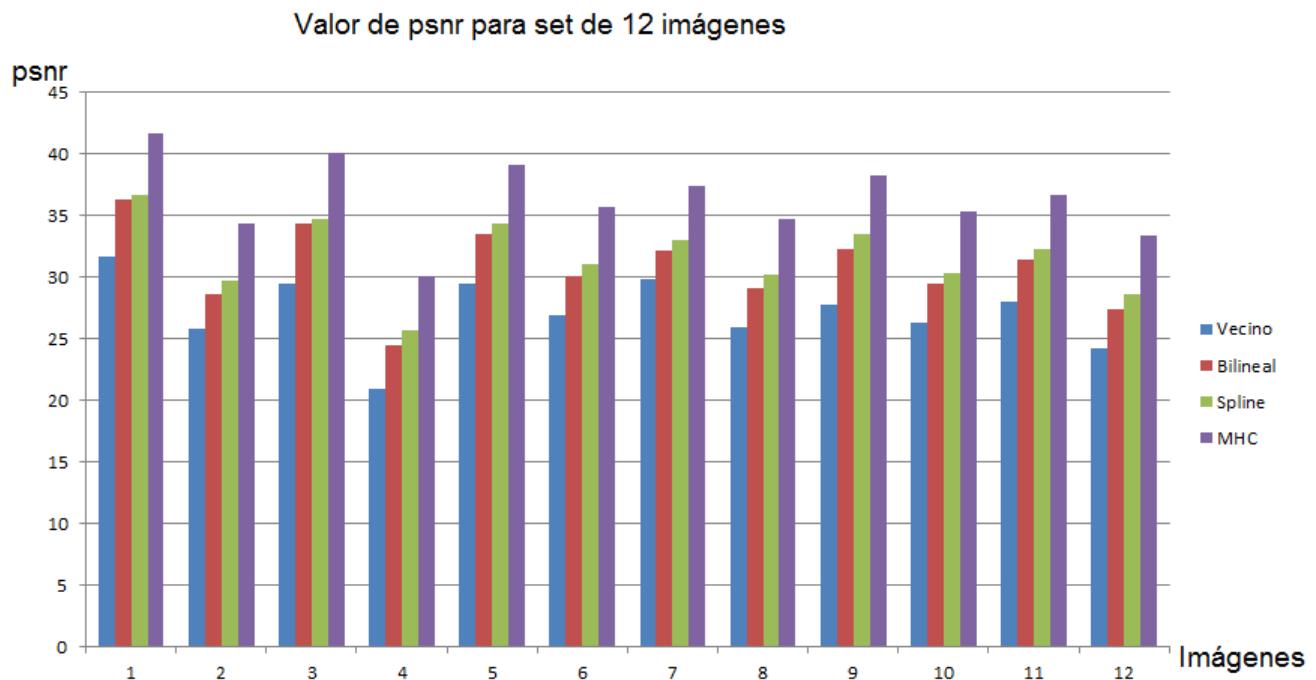
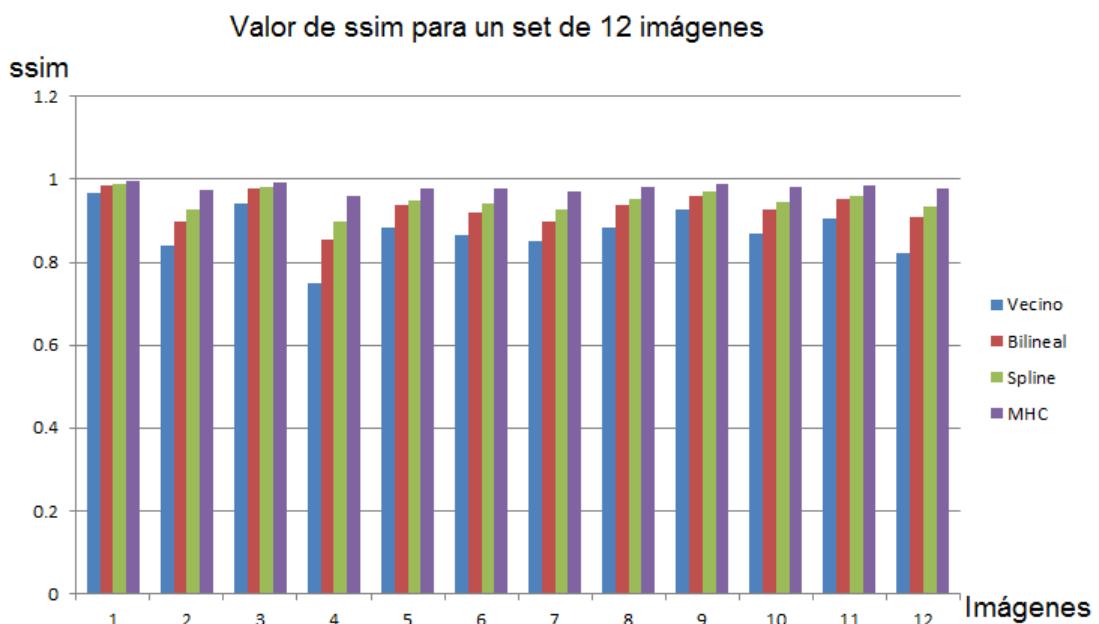


Figura 54: Evaluación de ssim para 12 imágenes del mismo tamaño



### 3.3. Análisis de tiempos

Aca hay que comparar los tiempos de ejecucion entre todos los algoritmos dispuestos.

## 4. Conclusiones y trabajo futuro

Es importante conocer nuestras limitaciones. Si bien bajo el algoritmo de MHC la existencia de *Artifacts* es casi nula, hay algunos que persisten y la forma de solucionar esto excede a nuestro trabajo. Por este motivo, nos centramos en el óptimo método para resolver el *Demosaicing* sin esperar que este sea infalible y meramente real.

Como primera instancia, el Algoritmo de *Vecino Más Cercano* es capaz de esbozar la estructura de la imagen a grandes rasgos. Sin embargo, los bordes no suelen ser muy precisos y la definición tiende a ser de baja calidad. Sumado a que es el método donde encontramos mayor diversidad de Artifacts y mayor ocurrencia bajo una misma imagen. Cabe destacar que los valores de psnr y ssim arrojados en nuestra experimentación para este método fueron siempre los menores.

La interpolación Bilineal tiende a ser una buena solución respecto de los contornos cuadrados que se generan bajo el método de Vecino más cercano, ya que admite un dibujo de contornos más flexible aunque no óptimo.

Los splines bla...

Con los resultados arrojados por nuestra experimentación nos vemos condicionados a determinar que el método óptimo entre los cuatro probados es el del Algoritmo de Malvar, He y Cutler. Este método es el que mejores números obtuvo de psnr y ssim para todos los casos evaluados. Además, es el método donde encontramos menor diversidad de artifacts y, sobre todo, menor cantidad de artifacts en una misma imagen.

Ademas de completar los puntos suspensivos, hay que agregar algo respecto del tiempo empleado por cada uno y si vale la pena esperar o no y bla...

También poner que los valores de ssim y psnr fueron constantes, es decir siempre en el mismo orden.

No hay que perder de vista que la implementación deberá ser ejecutada en tiempo real, lo cual implica que si un algoritmo es muy bueno para la estimación de la imagen pero tarda un tiempo considerablemente alto no es realmente el óptimo.

## 5. Apéndices

### 5.1. Apéndice A: Enunciado

### 5.2. Apéndice B:

## 6. Referencias

[1] Ron Kimmel. Demosaicing: image reconstruction from color ccd samples. *IMAGE PROCESSING, IEEE TRANSACTIONS ON*, 1999.

[2] Henrique S. Malvar, Li wei He, and Ross Cutler. High-quality linear interpolation for demosaicing of bayer-patterned color images. In *Proceedings of the IEEE International Conference on Speech, Acoustics, and Signal Processing*, 2004.

[3] R. Burden y J.D.Faires, *Análisis numérico*, International Thomson Editors, 1998