Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 3, No. 2, pp. 81-93, Diciembre 2016
Selección de los Mejores Artículos de CoNCISa 2016

k-Inpainting: Un Enfoque Híbrido para Inpainting

Esmitt Ramírez[1], Karina Pedrique[1]
esmitt.ramirez@ciens.ucv.ve, kpedrique@gmail.com

[1] Escuela de Computación, Universidad Central de Venezuela, Caracas, Venezuela

Resumen: En el procesamiento digital de imágenes se aplican técnicas que tienen como objetivo mejorar la calidad de la imagen o extraer algún tipo de información. Entre las técnicas utilizadas se encuentran la segmentación, corrección de imagen, análisis de la imagen, reconstrucción de partes perdidas, entre otras. En la reconstrucción de partes de una imagen, existe una técnica llamada Inpainting, la cual modifica una imagen para reconstruir áreas deterioradas o eliminadas de ésta. En la actualidad, los algoritmos de Inpainting ocupan un amplio campo de investigación y desarrollo. En este trabajo se plantea la implementación de un novedoso algoritmo de Inpainting llamado k-Inpainting, el cual está basado en un proceso iterativo para lograr la reconstrucción total de la imagen. El algoritmo k-Inpainting se presenta como una técnica híbrida que une dos enfoques distintos de investigación. Así, se demuestra su utilización al implementar una versión del algoritmo de forma secuencial y una versión paralela en la GPU. Se realizaron diversas pruebas variando los parámetros de entrada del algoritmo, para obtener valores óptimos. Las pruebas comprueban su eficiencia y efectividad en la reconstrucción de segmentos de imágenes a recuperar.

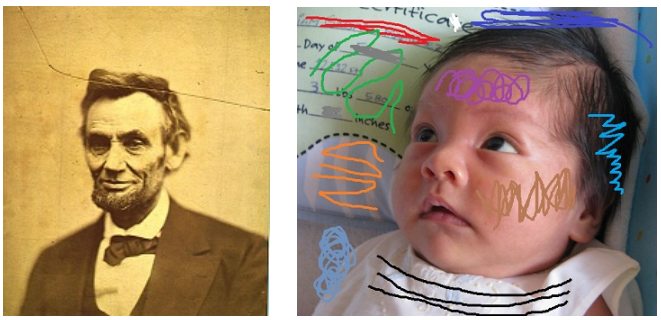
Palabras Clave: Inpainting; Reconstrucción de Imágenes; Síntesis de Textura; Difusión de Textura; Propagación de Textura; Auto-Similitud de Textura.

Abstract: In the digital image processing, techniques are applied in order to improve the quality of the image or to extract some kind of information. Among techniques used are the segmentation, image correction, image analysis, reconstruction of missing parts, and others. In the image reconstruction of missing parts, there is a technique called Inpainting, which modifies an image to reconstruct deteriorated areas of removed from it. At the present, Inpainting algorithms are studied in several research and development areas. In this paper, we present an implementation of a novelty algorithm named k-Inpainting, which is an iterative process to perform the total reconstruction of the image. The k-Inpainting algorithm is presented as a hybrid technique joining two different approaches. Thus, the use of k-Inpainting is probe with a sequential (using the CPU) and parallel version (using the GPU). We performed several tests varying the input parameters of algorithm, to obtain the optimal values. Tests checked its efficiency and effectiveness in the reconstruction of missing segments on images.

Keywords: Inpainting; Image Reconstruction; Texture Synthesis; Texture Diffusion; Texture Propagation; Texture Self-Similarity.

I. Introducción

La restauración de imágenes consiste en recuperar ciertas zonas en imágenes donde visualmente existe una interrupción de la continuidad estética. Un claro ejemplo consiste en el proceso de retoque de una fotografía antigua que ha sido digitalizada, removiendo artefactos notables u objetos, reconstruyendo áreas de un objeto en específico dentro de una imagen. El conjunto de técnicas asociadas con esta restauración se conoce como *Inpainting*. La Figura 1 muestra dos ejemplos de imágenes a ser recuperadas.



(a) (b)

Figura 1: Ejemplos de Restauración de Imágenes con (a) Fotografías Antiguas y (b) Fotografía con Interrupciones Visuales

Remover a un turista de una fotografía en un lugar histórico, o una señal de tránsito, o un ente inesperado en una imagen, es posible hacerlo con la técnica de *Inpainting*. De manera formal, el problema de remoción o restauración de imágenes

consiste en que dada una región Ω a ser restaurada, se emplea la información conocida que rodea dicha región, para regenerar de la manera más aceptable los datos en Ω [1].

El algoritmo de *Inpainting* debe buscar el mejor píxel q en Ω^c que sustituya el valor desconocido de un píxel p en Ω, tal como se muestra en la Figura 2. Es claro que $\Omega \cup \Omega^c = I$, donde I representa a la imagen.

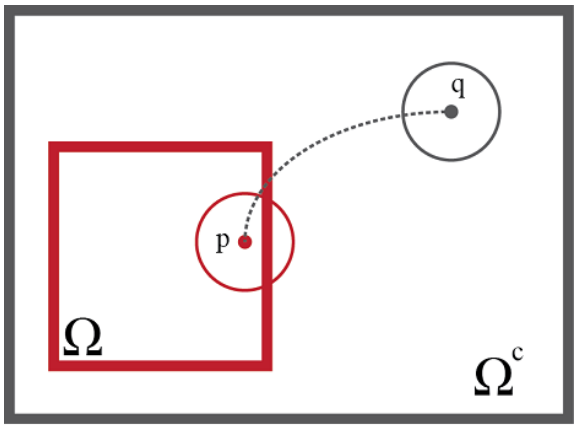


Figura 2: Representación de las Variables Involucradas en el Proceso de Inpainting. El Recuadro Rojo Representa el Área a Restaurar

Las diferentes aplicaciones de procesamiento digital de imágenes que se encargan de mejorar una imagen, tienen en común que los píxeles contienen información acerca de los datos reales y el ruido mientras que en *Inpainting*, no hay información significativa en la región a ser reconstruida. La información se encuentra principalmente alrededor de las áreas a ser tratadas [2]. En consecuencia, se han desarrollado técnicas específicas para estudiar este problema teniendo en común el sistema de vecindad de los píxeles [3].

El proceso de *Inpainting* surge a partir de la importancia de restaurar y modificar imágenes y videos, pero también es empleado para entender la validez de diferentes modelos de imagen (e.g. modelos de imagen estocásticos o determinísticos). Considerando estos modelos de imágenes, en [4] se plantea que la técnica de *Inpainting* puede ser dividida en tres grupos básicos: auto-similitud y síntesis de textura; difusión y propagación (basada en emplear ecuaciones diferenciales parciales de orden superior); y coherencia. Además, se puede incluir un nuevo grupo [5] donde se contemplan métodos basados en la continuación de bordes. Adicionalmente, existen diversos trabajos que combinan técnicas y crean nuevos algoritmos para *Inpainting*. Un resumen de las técnicas principales de *Inpainting* para la reconstrucción, fue presentado por Suthar y Patel [6], y por Joshua y Darsan [7].

En este trabajo, se presenta *k-Inpainting* como una propuesta híbrida que resulta de la combinación de dos trabajos de investigación: *A Comprehensive Framework for Image Inpainting* [4] y *The Generalized PatchMatch Correspondence Algorithm* [8], en conjunto con diferentes aportes presentes en este trabajo. Nuestro algoritmo tiene como entrada dos imágenes y una serie de parámetros: la primera imagen I de entrada es la destinada a ser procesada, la cual tiene una región desconocida Ω a ser reconstruida y la segunda imagen es una máscara M que permite diferenciar en coordenadas cartesianas los píxeles que deben ser procesados. Esta propuesta es llamada híbrida, ya que usa una aproximación del algoritmo *k-Nearest Neighbor* de [8] para una preselección de N candidatos para cada píxel p y aplicar la combinación de energías en [4] entre los N candidatos escogidos previamente para cada píxel p.

Este artículo se organiza de la siguiente forma: la Sección II presenta una visión general de nuestra propuesta. En la Sección III se muestra el proceso de interacción del usuario con la zona a reconstruir, para poder aplicar el algoritmo de *k-Inpainting* que se explica detalladamente en la Sección IV. Dos etapas de particular importancia, *k-Propagación* y *Búsqueda Aleatoria* son presentadas en la Sección V. La Sección VI presenta la experimentación aplicada a nuestra propuesta. Finalmente, en la Sección VII se muestran las conclusiones y trabajos a futuro.

II. Enfoque Global del Algoritmo

La Figura 3 muestra el proceso de reconstrucción aplicado en *k-Inpainting*. La región Ω es reconstruida de afuera hacia adentro, siguiendo un esquema de capas. El color de cada píxel p en Ω es reemplazado por el color de algún píxel q en Ω^c. Este píxel q es escogido aplicando la técnica de mapa de correspondencia junto a otros algoritmos iterativos. Además, se utilizan diferentes métricas para lograr un mejor resultado.

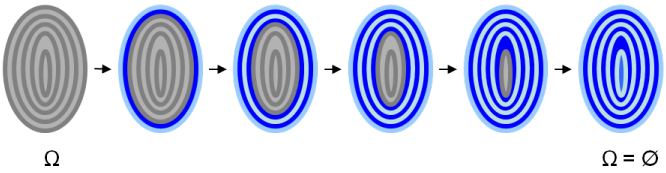


Figura 3: Proceso de Reconstrucción: La Región Ω es Reconstruida de Afuera Hacia Adentro Siguiendo un Sistema de Capas

El enfoque basado en capas permite que el conjunto Ω^c vaya siendo mayor iteración a iteración, de forma tal que exista una mayor cantidad de píxeles q, para un píxel p. El algoritmo *k-Inpainting* puede ser implementado en un enfoque secuencial y en ambientes paralelos. Las métricas presentadas en este trabajo están aplicadas sobre el espacio de color L*a*b [10].

En la Figura 4 se muestra la estructura general del algoritmo de *k-Inpainting* en seis pasos, donde los pasos 4-6 se ejecutan en diversas iteraciones hasta que se converja al resultado final. Primero, se obtienen los datos se selecciona la región a ser reconstruida. A partir de la selección, se aplica el algoritmo DFS (*Depth-First Search*), y se construye una pirámide de Gauss de dos niveles conformada por las imágenes I e I'. La imagen I' constituye una imagen de menor tamaño para poder operar de forma rápida sobre una aproximación de I.

Sobre la imagen I' se aplica la etapa de inicialización que proporciona valores preliminares a Ω a partir del procesamiento de los píxeles recolectados mediante un muestreo. Así, sobre I' se aplica el procesamiento de textura para refinar la zona a

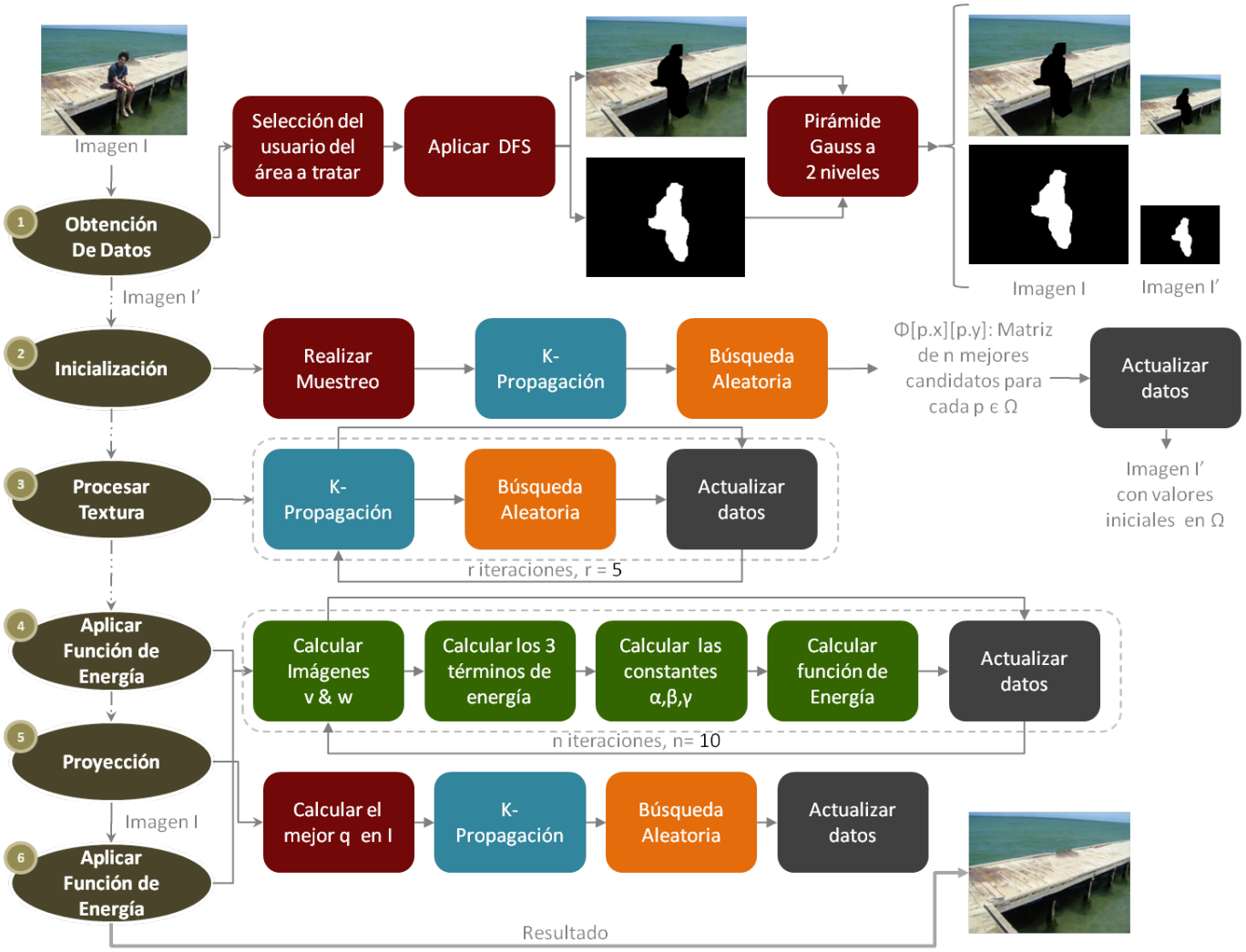


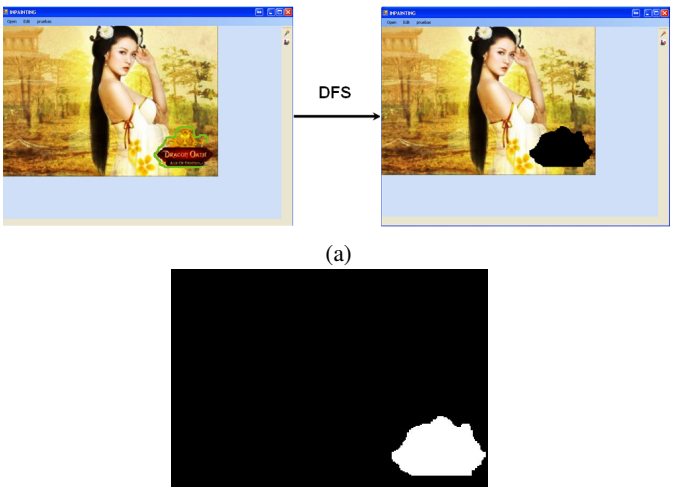
Figura 4: Estructura General del Algoritmo Propuesto

tratar y la prepara para la siguiente etapa, donde se aplica una función de energía que considera las tres técnicas de *Inpainting*. Después, se realiza una proyección de los píxeles procesados en I' a la imagen original I. Finalmente, se aplica la función de energía sobre I y se obtienen el resultado. Realizar el procesamiento en la textura de menor tamaño I' y luego realizar la proyección a a I es menos costoso en tiempo y cómputo que sobre la imagen I de forma directa.

Las sub-etapas *K-Propagación* y *Búsqueda Aleatoria* (etapa 2, 3 y 5) son algoritmos que trabajan en conjunto y buscan encontrar los mejores candidatos para cada píxel que constituye la parte desconocida de la imagen. Se denomina candidato a cualquier píxel $q \in \Omega^c$ que pueda reemplazar el valor incógnita de algún píxel $p \in \Omega$. Estos algoritmos emplean métricas diferentes dependiendo de la etapa donde se ejecuten.

III. Obtención de Datos

El usuario selecciona un área a tratar dentro de la imagen, y se busca un punto dentro del área a reconstruir para aplicar el algoritmo de DFS para llenar dicha área (ver Figura 5a). También se construye una máscara binaria (ver Figura 5b).



(a)

(b)

Figura 5: (a) Selección del Área a Reconstruir en la Imagen y (b) Máscara de la Imagen que Permite Diferenciar el Área a ser Tratada (Ω)

En *k-Inpainting* se utiliza un esquema de multi-resolución [11][12][13], donde se construye una pirámide gaussiana [14] de dos niveles, es decir, se obtiene una nueva imagen I' dos veces más pequeña que la imagen I. Para cada imagen se recopila la máscara y puntos máximos y mínimos del área a reconstruir, así como los puntos máximos y mínimos con respecto a la imagen anterior.

Para el paso de proyección es importante ubicar correctamente la correspondencia de los puntos entre las imágenes, especialmente para las coordenadas ubicadas dentro de Ω. En algunos casos las dimensiones de la imagen no son múltiplo de 2, por ello debe almacenarse el resto de la división. El proceso de corresponder un punto de I a I' se estudiará posteriormente.

IV. Algoritmo *k-Inpainting*

Como se mencionó anteriormente, para aplicar *k-Inpainting* se requiere realizar un muestreo sobre la parte conocida de la imagen para seleccionar píxeles candidatos que pueden sustituir los valores desconocidos de los píxeles que constituyen a Ω. Del mismo modo, se debe emplear una estructura de *heap* con algunas variaciones que permita almacenar los mejores candidatos para cada píxel $p \in \Omega$.

En las diferentes etapas de *k-Inpainting* se emplean diferentes métricas, algunas denominadas términos de energía, mientras que en otras sub-etapas se emplean una ecuación que acopla dichos términos. Así, se deben conocer tales métricas antes de explicar con detalle cada etapa que conforma el algoritmo. A continuación, se describirá la forma como se efectúa el muestreo, así como las características del heap. También se discutirá los términos de energía y de la combinación de los mismos en una función de energía.

1) Implementación del Heap: Para la aplicación del algoritmo, es necesario realizar un muestreo sobre la parte conocida de la imagen para realizar una selección inicial de los candidatos que pueden sustituir los valores desconocidos de los píxeles que constituyen a Ω; por ello se emplea un heap con algunas variaciones que permita almacenar los mejores candidatos para cada píxel $p \in \Omega$.

La formulación de esta estructura está basada en [8], donde implementan un *max-heap* que almacena la distancia D entre dos vectores de color, teniéndose la mayor distancia en el tope de la estructura. Cuando se examinan los candidatos, se construye una tabla *hash* que verifica rápidamente si el candidato ya ha sido almacenado en la lista.

Se presenta la implementación de un *max-heap* sin repetición basado en arreglos con variaciones. La estructura almacena dos datos, un tipo de dato *punto* que identifica cual píxel $q \in \Omega^c$ es candidato a reemplazar algún $p \in \Omega$ y un tipo de dato *float* que es la diferencia entre las vecindades de p y q para alguna métrica. Además, se dispone de una tabla *hash* para guardar los puntos que ya han sido escogidos. El ordenamiento de los puntos viene dado la distancia entre los colores de la vecindad de p y q (ver Figura 6).

Antes de actualizar los datos en la imagen en cada paso del algoritmo, se debe conocer cuál de los N candidatos es más

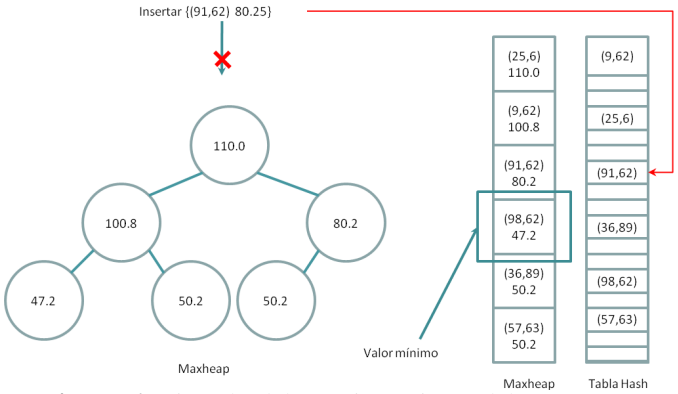


Figura 6: Ejemplo del Funcionamiento del *Max-Heap* Mostrando el Proceso de Inserción del Nodo con Distancia 80.2 y Posición 91,62. Sin embargo, Antes de Insertar se Compara con la Tabla *Hash*, Como la Posición 91,62 Existe, Entonces no se Inserta

conveniente para cada píxel p. El mejor candidato es el píxel que tiene menor diferencia entre su vecindad y la vecindad del píxel para el cual es candidato (i.e. menor valor flotante).

Se estimó que cuando el parámetro N corresponde al 0.05% de los píxeles totales de la imagen, se obtienen resultados satisfactorios. El algoritmo maneja un atributo $16 \leq K \leq 32$ para dos algoritmos que aquí hemos llamado *K-Propagación* y *Búsqueda Aleatoria*. Si el valor de N es menor que el máximo valor que toma K, entonces N toma el máximo valor de K. Dicho esto, N es definido como lo indica la Ecuación 1, dónde $\rho = (I.Width * I.Height * 0.05)/100$.

$$N = \begin{cases} \rho & si \quad \rho \geq 2 \times K \\ 2 \times K & \text{de otro modo} \end{cases} \tag{1}$$

La definición de N es formulada de esta manera para que la dimensión del *heap* no sea muy pequeña con respecto a una selección previa de K candidatos.

2) Muestreo: El paso de inicialización (ver Figura 4) realiza un muestreo uniforme sobre la imagen a procesar, para escoger aleatoriamente los posibles candidatos de cada píxel $p \in \Omega$. Para llevar a cabo tal muestreo se requiere de un generador de números pseudo-aleatorios con distribución uniforme, en los que cada valor tiene la misma probabilidad. En *k-Inpainting* se emplea el método de congruencia lineal [15] y un algoritmo de distribución uniforme [16].

Es importante realizar un muestreo sobre la parte conocida de la imagen, porque realizar un algoritmo de fuerza bruta (cada píxel desconocido por cada píxel de la parte conocida de la imagen) tiene un alto costo computacional, lo cual implica horas de procesamiento en PCs convencionales.

3) Combinación de los Términos de Energía: Los términos de energía se pueden clasificar en técnicas de auto-similitud y síntesis de textura; difusión y propagación; y coherencia. En [4] se presenta una combinación de dichas técnicas que seleccionan un candidato no solo por la similitud de sus vecindades en cuanto a los colores que la constituyen, sino

a la forma como están organizados los colores (coherencia espacial) y el cambio del tono del color de un vecino a otro (difusión), lo que hace que los resultados sean exitosos. Cada uno de estos métodos fue planteado como un término de energía, que serán explicados a continuación.

Auto-similitud y Síntesis de Textura

En [17] [1][4] se presenta una formulación variacional del método de síntesis y textura propuesto por Efros y Leung [18]. Esta variación consiste en calcular el mapa de correspondencia φ que minimiza la función de energía expresada en [17], para un píxel $p \in \Omega \subset \iota$, como:

$$E(\varphi) = \sum_{\tau \in N_0} \|\iota(\varphi(p+\tau)) - \iota(q+\tau)\|^2 \quad (2)$$

La síntesis de textura suele ser un proceso de refinamiento iterativo, y la Ecuación 2 debe ser expresada en función a las r iteraciones aplicadas para procesar la textura (ver Ecuación 3).

$$E_1(\varphi, \varphi^{r-1}) = \sum_{\tau \in N_0} \left\|\iota(\varphi^{r-1}(p+\tau)) - \iota(q+\tau)\right\|^2 \quad (3)$$

Difusión y Propagación

Se toma la difusión Laplaciana como segundo término de energía, entonces dado un píxel $p \in \Omega$, la difusión Laplaciana de p se define como:

$$\upsilon(p, \varphi) = \frac{1}{4}\left[\iota(\varphi(p^N)) + \iota(\varphi(p^S)) + \iota(\varphi(p^E)) + \iota(\varphi(p^O))\right] \quad (4)$$

Basados en la Ecuación 4, el segundo término de energía es:

$$E_2(\varphi, \varphi^{r-1}) = \sum_{\tau \in N_0} \left\|\upsilon(p+\tau, \varphi^{r-1}) - \iota(q+\tau))\right\|^2 \quad (5)$$

Coherencia

El tercer y último termino de energía se define como:

$$E_3(\varphi, \varphi^{r-1}) = \sum_{\tau \in N_0} \left\|\omega(p, \tau, \varphi^{r-1}) - \iota(q+\tau))\right\|^2 \quad (6)$$

, siendo ω el cálculo que favorece la similitud de los parches correspondiente a los píxeles vecinos. Para hacer esta estimación menos sensible a valores que no se encuentran en la imagen (valores aberrantes), en [4] proponen emplear la mediana de los valores generados tal como sigue:

$$\omega(p, \tau, \varphi^{r-1}) = median_{l \in N_0}\left\{\iota(\varphi^{r-1}(p+l) - l + \tau)\right\} \quad (7)$$

El cálculo de la mediana en el espacio L*a*b se hace comparando primero el componente de luz, en caso que para dos colores estos valores sean iguales, se ordena con respecto al componente *a, y por último por el componente *b. Se ordenaron de esta forma, debido a la sensibilidad del ojo humano a la luz. Nótese que todos los términos de energía se basan en la suma de diferencias cuadradas.

4) Combinación de los Tres Términos de Energía: Se deben escoger los N mejores candidatos para cada $p \in \Omega$, los cuales son almacenados en un conjunto $\Phi(p)$. Para cada conjunto $\Phi(p)$ se aplican los tres términos de energía combinados con respecto a cada píxel p en cada iteración r, quedando como:

$$\varphi^{r-1}(p) = argmin_{q \in \Phi(p)}(\alpha E_1 + \beta E_2 + \gamma E_3) \quad (8)$$

Las constantes α, β y γ definen la influencia de cada uno de los tres términos en el resultado final. Estos pesos han sido definidos dependiendo de las propiedades de cada píxel en la imagen según las métricas determinadas. El cálculo de dichos pesos viene dado en primer lugar por el valor mínimo de cada uno de los términos de energía (ver Ecuación 9) En [4] estos pesos han sido definidos dependiendo de las propiedades de cada píxel en la imagen según las métricas determinadas. El cálculo de dichos pesos viene dado en primer lugar por el valor mínimo de cada uno de los términos de energía (ver Ecuación 9)

$$m_i(p) = min_{q \in \Phi(p)} E_i \quad (9)$$

Estos mínimos son considerados en la estimación de las constantes, ya que se asume que los valores de sus pesos podrían depender de la validez del mejor parche candidato (píxel q con su vecindad definida según el parámetro L). Esto es porque si el parche candidato para un determinado píxel en un tiempo t no es suficientemente bueno, el valor del correspondiente peso podría ser pequeño, es decir, se disminuirá en función del valor $m_i(p)$. En relación a lo antes descrito, siendo α, β y γ definidos como:

$$\alpha(p) = \epsilon^{\frac{-m_1}{\sigma}}, \beta(p) = \epsilon^{\frac{-m_2}{\sigma}}, \gamma(p) = \epsilon^{\frac{-m_3}{\sigma}}, \quad (10)$$

donde $\sigma = \frac{m_1+m_2+m_3}{3}$.

Investigaciones previas de *Inpainting* han demostrado que usar el filtro de la ecuación laplaciana arroja mejores resultados que otros tipos de filtros, debido a ello, hemos usado la ecuación laplaciana aplicada sobre cada píxel $p \in \omega \subset \iota$ para cenerar genera una imagen denominada imagen de difusión v y el cálculo de ω sobre el mismo conjunto de píxeles produce una imagen de coherencia w. Por ello es conveniente construir la imagen coherencia y difusión con las ecuaciones 7 y 4 respectivamente, y luego realizar todos los cálculos pertinentes para cada término de energía.

V. K-Propagación y Búsqueda Aleatoria

En varias etapas del algoritmo que se muestra en el esquema de la Figura 4, se aplican los algoritmos de *K-Propagación y Búsqueda aleatoria* para obtener los candidatos de cada píxel desconocido de la imagen. A continuación, se describe cada uno de estos.

A. Obtener Candidatos

Dependiendo de la etapa del algoritmo que se esté aplicando, se puede tener una estructura Υ (lista de muestras) o Φ (matriz de *heaps*) con candidatos. Dichas estructuras son datos de entrada a dos algoritmos que encuentran los mejores N píxeles candidatos a sustituir el valor desconocido de cada píxel p.

Estos dos algoritmos reciben el nombre de *K-Propagación* y *Búsqueda Aleatoria*, el primero toma cada muestra y la compara con K vecinos en una dirección x y en una dirección y. El segundo algoritmo hace una búsqueda aleatoria en varias direcciones a diferentes radios de distancia del píxel escogido y lo compara con sus vecinos. Cabe destacar que estas funciones forman parte del algoritmo *NNF* (*Nearest Neighbors Field*) y *K-NNF* (*K-Nearest Neighbors Field*) presentados en [19] y [8] respectivamente, los cuales están enfocados en la síntesis de textura basada en parches. En esta investigación, éstos se modificaron para sintetizar la textura basada en píxeles.

Las métricas que permiten conocer cuál píxel es mejor candidato que otro, están basadas en la suma de las diferencias cuadradas de las vecindades del píxel incógnita y el píxel seleccionado. Suponiendo la utilización de alguna métrica que calcula una distancia D, se muestra el funcionamiento de estos algoritmos.

1) K-Propagación: Dado un píxel desconocido $p \in \Omega$, *K-Propagación* es aplicado a todos los píxeles $q \in \Upsilon$ ($q \in \Phi$ dependiendo de la etapa que esté ejecutando *k-Inpainting*). Sin embargo, antes se deben escoger las direcciones en el eje x y y. En [19] (algoritmo base de *K-Propagación*) se establece que los candidatos $f(x, y)$ tratan de ser mejorados haciendo desplazamientos $f(x-1, y)$ y $f(x, y-1)$ y en algunas iteraciones los examinan de forma inversa $f(x-1, y)$ y $f(x, y-1)$. En esta propuesta, las direcciones son escogidas aleatoriamente como $q_{kx} = (x \pm k, y)$ y $q_{ky} = (x, y \pm k)$, donde $0 \leq k < K$.

La aleatoriedad de las direcciones viene dada por la elección del signo. En x, si el número aleatoriamente seleccionado es par, el signo es negativo y es positivo en caso que el número sea impar. Para y se sigue la misma política.

Se debe acotar que K es un parámetro global del algoritmo. Los estudios arrojados en [8] del algoritmo *k-PatchMatch*, indican que el valor ideal para K es el valor constante 16. Por lo que se tomará un rango de valores [8,32) para esta variable global en el algoritmo *k-Inpainting*.

Una vez comprendido como se escogen las direcciones x y y y la influencia de la variable K, se puede describir el funcionamiento completo de la subetapa *K-Propagación*, que se resume en el Algoritmo 1.

Algorithm 1 K-Propagación

```
1: procedure KPROPAGACION(Point p ∈ Ω, Point q ∈ Υ,
   Int K)                                  ▷ Siendo Υ ⊂ Ω^c
2:     MaxHeap H(K) ▷ H máximo almacena K candidatos
3:     Escoger una dirección aleatoria en el eje de las x
4:     Escoger una dirección aleatoria en el eje de las y
5:     Evaluar y obtener D para K candidatos electos y
   almacenarlos en H
6:     List λ = H.List()
7:     return λ
8: end procedure
```

2) Búsqueda Aleatoria: En *K-Propagación* se obtuvo la lista λ con los K mejores candidatos. Para evitar un mínimo local, se aplica el algoritmo de *Búsqueda Aleatoria* propuesto en [19], en donde para cada $q \in \lambda$, se aplica que dado $v_0 = \varphi(p)$, se intenta mejorar tal valor probando una secuencia de desplazamientos de candidatos en un decremento de distancia exponencial desde v_0, tal como se muestra en la Figura 7, descrito como $u_i = v_0 + w\alpha^i R_i$.

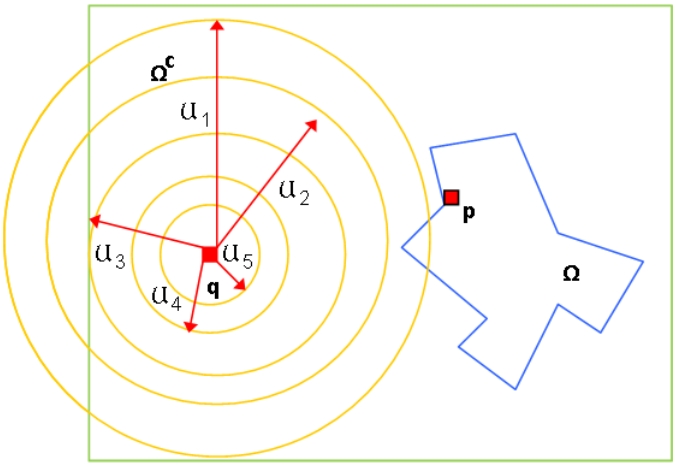


Figura 7: Búsqueda Aleatoria de Puntos Dada una Posición q

La variable R_i es una variable aleatoria uniforme sobre la ventana $[-1, 1] \times [-1, 1]$, w es el máximo radio de búsqueda y α es un valor fijo que varía la distancia del radio de búsqueda en alguna dirección. Se examinan las vecindades de los nuevos puntos generados para i = 0,1,2, ..., hasta que $w\alpha^i < 1$. Entonces, w es la máxima dimensión de la imagen y $\alpha = 1/2$. Sin embargo cuando un nuevo punto generado está fuera de la imagen, este se descarta y se realiza la siguiente iteración.

B. Inicialización

El principal objetivo de este paso es encontrar desde un punto de vista espacial los mejores candidatos para los valores incógnitas de I. Se busca el argumento mínimo producto de aplicar los tres términos de energía.

Una forma de encontrar este argumento, que es empleado por diferentes trabajos [17], [12], [11], [20], es aplicar un modelo basado en los Campos Aleatorios de Markov (MRF). Sin embargo, este mecanismo puede ser costoso en el tiempo para nuestro propósito, por ello en [12] se agiliza el tiempo de respuesta usando esta técnica con el algoritmo de Label Pruning.

En el 2009, Barnes et al. [19] propusieron un algoritmo que busca rápidamente $\varphi(p)$ que recibe el nombre de *Nearest Neighbors Field* (NNF). Sobre este fue realizado una optimización en [8] que permite obtener resultados más precisos, dicha optimización recibe el nombre de *K-Nearest Neighbors Field* (*K-NNF*). En cualquier caso, Barnes et al. [19] emplean en su algoritmo de *PatchMatch* un paso de inicialización, donde realiza pocas iteraciones iniciales del algoritmo *NNF* o *K-NNF* con una inicialización aleatoria.

Basados en la este concepto, sobre $\Omega^c \in I'$ se realiza un muestreo uniforme, a partir del cual se aplicará el algoritmo *NNF*, pero con algunas variantes de las funciones *Propagation* y *Random Search*.

La función de muestreo que aplica el Algoritmo de Las Vegas recibe dos parámetros importantes: el porcentaje de píxeles que serán tomados por fila o columna, y un valor δ que indica la cantidad de filas o columnas a muestrear. Si el ancho de la imagen es mayor que el alto de la misma, el porcentaje va dirigido a las filas y el valor δ indica cada cuantas filas se toma el muestreo de las mismas, en caso contrario aplica igual pero para las columnas.

El porcentaje de píxeles tomados por cada fila o columna es del 25%, mientras que el δ toma el valor constante 2. Estos valores fueron estimados de forma empírica, realizando diferentes pruebas. El algoritmo probabilístico aplicado devuelve una lista Υ con píxeles seleccionados. En dicha lista todos los píxeles son diferentes entre sí en cuanto a ubicación espacial en la imagen.

El muestreo es realizado cada vez que se detecta un borde $\delta\Omega$, por lo que los píxeles en Υ_j son los candidatos a sustituir a los píxeles $p_i \in \delta\Omega_j$ por cada $\delta\Omega_j \subset \Omega$, la Figura 8a presenta el muestreo realizado para cada capa sobre la máscara de la imagen a procesar y la Figura 8b muestra los valores que toma Υ_j para $\delta\Omega_j$.

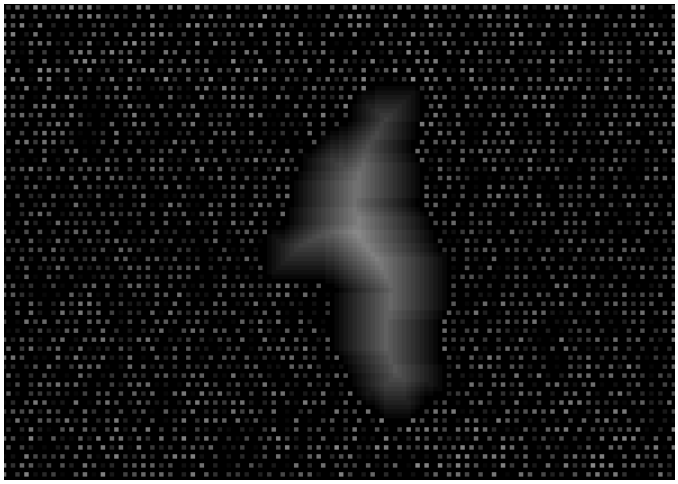
Los N mejores candidatos para cada píxel $p \in \Omega$ son almacenados en un *max-heap* , por ello se construye una matriz de tipo *max-heap* de las dimensiones de la zona a reconstruir. Como la forma de la zona, puede ser irregular, dentro de la matriz pueden existir estructuras vacías para los píxeles que no pertenecen a Ω.

Con los mejores N candidatos para cada píxel desconocido del agujero, se asigna el mejor candidato al píxel p_i de la zona a tratar en la imagen. Por ser el paso de inicialización, se está reconstruyendo el área Ω de I' y se indica en una máscara auxiliar cuales píxeles fueron sustituidos. Dado que este proceso es iterativo, siempre se usa una máscara auxiliar para no perder la localidad espacial del área de la imagen que se está procesando (ver el Algoritmo 2).

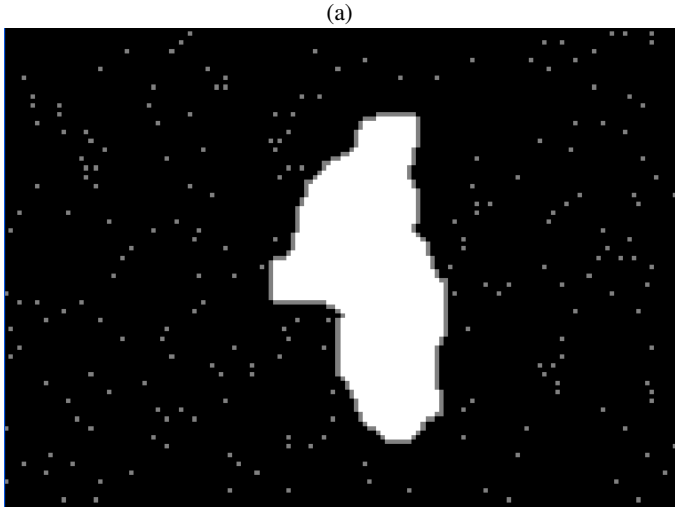
La *K-Propagación* y *Búsqueda Aleatoria* son aplicados sobre todas las muestras con respecto a cada píxel p_i y encuentran los mejores N píxeles a sustituir el valor desconocido de cada p. Así,ya se ha calculado $\varphi(p)$ $\forall p \in \Omega \subset I'$. Para calcular $\varphi(p)$, se utiliza una variación sobre la suma de diferencias cuadradas (i.e. ssd), de manera similar a como lo plantea Komodakis [12], donde la diferencia de los vecinos en posiciones desconocidas, no son calculadas (ver Ecuación 11).

$$ssd(N_p, N_q) = \sum_{\tau \in N_0} M(p+\tau) * \|\iota(p+\tau) - \iota(q+\tau)\|^2 \quad (11)$$

donde $p \in \Omega$, $q \in \Omega^c$ e $\iota = \Omega \cup \Omega^c$.



(a)



(b)

Figura 8: Representación del (a) Muestreo Realizado por Cada Capa, y (b) Un Ejemplo del Muestreo Sobre una Sola Capa (*la más externa*)

Algorithm 2 Algoritmo para inicializar Ω

```
Array Υ
MaxHeap Φ[widthHole][heightHole]
Image M = mask ▷ Se asigna la máscara (mask) de I' a una máscara auxiliar
while Ω ≠ ∅ do
    Seleccionar δΩ_j
    RealizarMuestreo(Υ_j, Mask, ...)
    for cada píxel p ∈ δΩ_j do
        Hallar los n mejores candidatos a partir de Υ_j y almacenarlos en Φ[p.X][p.Y]
        I(p) = φ(p)
        M(p) = 0
    end for
end while
```

C. Procesamiento de Textura

Este paso tiene como entrada el conjunto Φ, compuesto por los candidatos electos para cada píxel p. Estos píxeles han sido escogidos mediante la Ecuación 11, siendo estos los primeros valores que toma Ω en el paso de inicialización. Así, se llama $\varphi^0(p)$ al mejor candidato q que sustituyen a cada píxel p. Nótese que que el algoritmo *k-Inpainting* es iterativo y a partir de $\varphi^0(p)$ es cuando los términos de energía son estimados.

Para procesar una textura hay que sintetizarla. Para esto, se utiliza una variación de la versión original de K-NNF (Obtener candidatos) con r iteraciones, donde se utiliza la Ecuación 2 en lugar de la Ecuación 11. Esto es para refinar el procesamiento de la textura y obtener un mejor acabado.

Los candidatos a ser evaluados provienen del conjunto $\Phi(p)$ para cada p, como se mencionó anteriormente y estos son actualizados en cada iteración (ver Algoritmo 3).

Algorithm 3 Algoritmo para procesar textura

```
1: r = 0                                  ▷ Sea valor el número de iteraciones
2: while r < valor do ▷ Aplicar el algoritmo para Obtener candidatos usando como candidatos a los q ∈ Φ(p)
3:     while Ω ≠ ∅ do
4:         Seleccionarδ Ω_j
5:         for cada píxel p ∈ δΩ_j do
6:             λ = Kpropagacion(Φ(p))
7:             BusquedaAleatoria(λ, Φ(p))        ▷ Φ(p) se actualiza en la Búsqueda Aleatoria
8:             ι = φ(p)
9:         end for
10:    end while
11:    r = r + 1
12: end while
```

Basados en una experimentación realizada por los autores, cuando r toma como valor máximo 5, se obtienen resultados satisfactorios. Sin embargo este valor puede variar de acuerdo a característica particulares de la imagen. Los puntos p pertenecientes a Ω siempre se toman de afuera hacia dentro.

D. Aplicar Función de Energía

Para aplicar la función de energía, se generan las imágenes de difusión v y coherencia w. Para este momento debe existir una matriz o conjunto de *max-heap* Φ como se expresa en la Ecuación 12 a continuación.

$$\Phi|\Phi[p.X][p.Y] \vee \Phi(p) \tag{12}$$

Dicha ecuación es equivalente a los mejores N candidatos para un $p \in \Omega$. Estos candidatos han sido escogidos empleando pasos que emplean solo el primer término de energía para mejorar los candidatos mediante un refinamiento de la textura, tal como el paso 3 (i.e. Procesar textura) y paso 5 (i.e. Proyección) de la Figura 4.

Luego, se intenta mejorar estos candidatos calculando las magnitudes de las ecuaciones 3, 5 y 6 para todos los píxeles $q \in \Phi(p)$, y las constantes α, β y γ. Posteriormente, se calcula la Ecuación 8 actualizando la lista de $\Phi(p)$, tomándose como valor de distancia el dato calculado por la función de energía.

Seguidamente, se toma el mejor valor de cada $\Phi(p)$ para cada píxel $p \in \Omega$, es decir $\varphi(p)$, y se actualiza la imagen a tratar. Una vez actualizada la imagen, se vuelve a calcular las imágenes de difusión (v) y coherencia (w) y se repite el proceso antes descrito, y así sucesivamente hasta que se hayan cumplido un número de n iteraciones. Para diferenciar las iteraciones del procesamiento de textura de éstas iteraciones, aquí se habla de n iteraciones en vez de r iteraciones.

La manera como los píxeles p son seleccionados, es de afuera hacia adentro, siguiendo un esquema de capas y contando con una máscara auxiliar que permita distinguir los píxeles que faltan por procesar de los que no. Una vez procesados todos los píxeles, a la máscara auxiliar se le asigna la máscara original para repetir el proceso de aplicar la función de energía en la siguiente iteración. En este punto, es donde el proceso iterativo es fundamental en nuestra propuesta.

El número de iteraciones n para esta parte del algoritmo es un parámetro fijo que puede ser manipulado por el usuario, sin embargo para $n = 10$ se obtienen buenos resultados [4].

En *k-Inpainting* esta variable puede tomar otros valores que serán mostrados en la sección VI. Es importante destacar, que para la primera iteración del algoritmo la Ecuación 3 no es calculada, si no que se toma el valor que se obtuvo del paso anterior, el cual ha buscado los mejores candidatos mediante este primer término de energía.

E. Proyección

En una proyección es natural proyectar cuatro puntos a una imagen grande a partir de un punto de la imagen de menor tamaño. Sin embargo, ese tipo de proyección puede provocar que se procesen puntos de Ω en un orden inadecuado, esto causa que se presenten problemas en el algoritmo *k-Inpainting*.

Entonces, la proyección se hace a partir de la imagen de mayor tamaño a la de menor tamaño, es decir, se toma un punto $p \in \Omega \subset I$ y se busca el punto correspondiente en I'. Una vez encontrado ese punto en I', se proyectan ahora de la forma $I' \Longrightarrow I$ los candidatos $q' \in \Phi(p'), p' \in \Omega \subset I'$, donde cada q' proyecta cuatro puntos $q \in \Omega^c \subset I$.

Como no es deseable aumentar la cantidad de candidatos mediante la Ecuación 11, se calcula cuál de los cuatro puntos proyectados es el nuevo candidato para p. El mejor q para cada p, viene a ser $\varphi(p)^0$ para esta imagen y reemplazan en una iteración inicial a los valores incógnitas $p \in \Omega \subset I$, es decir a $I(p) = \varphi(p)^0$.

Posteriormente, como indica la Figura 9, se aplica una síntesis de textura con el algoritmo para *Obtener candidatos* con el primer término de energía. Sin embargo, como ya se tiene una buena estimación espacial, el algoritmo es realizado una única vez, es decir, no se itera r veces.

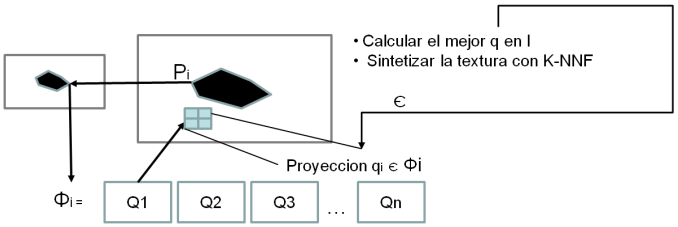


Figura 9: Esquema de Procesamiento de la Etapa de Proyección

Tabla I: Descripción de los Sistemas Utilizados

Sistema	Procesador	RAM
1	Intel Core i3 3.07Ghz	4 GB
2	Intel Core 2 Quad 2.40 GHz	4 GB
3	Intel Pentium 4 3.20 GHz	1 GB

Tabla II: Descripción Tarjetas Gráficas Utilizadas

Sistema	Tarjeta	Memoria	Núcleos
1	GeForce GTX 470	1280 MB	448
2	GeForce GT 8800	640 MB	112
3	GeForce GT 9600	512 MB	64

Anteriormente se menciona que para cada imagen son almacenados los puntos máximos y mínimos con respecto a la imagen anterior, así como sus módulos. Esto es para hacer una buena correspondencia de puntos en el momento de proyectar. Sea p un punto dentro de $\Omega \subset I$ y sea $pmin$, $pmax$, $pminmod$ y $pmaxmod$ los puntos máximos y mínimos de Ω con sus correspondientes módulos, para buscar el punto p' correspondiente dentro de Ω de I', se calcula como $p' = (p - m - pminmod)/2$, donde m es un punto dentro de la ventana $[0, 0] \times [1, 1]$.

Luego de aplicar la proyección, Φ ha sido redimensionado y actualizado con los mejores candidatos para $\Omega \subset I$, que fueron escogidos primero con la Ecuación 11 y luego mejorados con la Ecuación 2. Buscando mejorar los candidatos, se aplica nuevamente la función de energía, tal como se ilustra en el paso 6 de la Figura 4, y finalmente se obtienen los resultados de la imagen.

F. k-Inpainting en la GPU

El algoritmo de *k-Inpainting* obtiene buenos resultados en una versión secuencial, sin embargo su tiempo de ejecución es extenso. Por ello, se planteó realizar una versión paralela en la GPU empleando la arquitectura CUDA provista por las tarjetas gráficas de Nvidia. Entonces, cada etapa del algoritmo se ejecuta en paralelo en sus sub-etapas, a excepción de *Obtener datos de entrada*.

Una diferencia importante con respecto al diagrama secuencial, es la encapsulación de las sub-etapas *K-Propagación* y *Búsqueda Aleatoria* en una etapa llamada *K-NNF*. En la etapa de *Procesamiento de textura*, aplicando un ciclo de r iteraciones para que sea ejecutado una vez si se procesa con todas las muestras o r veces si se paraleliza.

VI. Pruebas y Resultados

En esta sección se presentan las diferentes pruebas realizadas para reconstruir el área desconocida de cada imagen, así como el ámbito de ejecución de las mismas, para finalmente realizar un análisis de los resultados obtenidos.

El algoritmo fue probado en tres equipos con especificaciones de hardware distintas (ver Tabla I). Igualmente, el hardware gráfico constó de tarjetas Nvidia con soporte CUDA (ver Tabla II).

El lenguaje de programación C# fue empleado para la GUI y el algoritmo *k-Inpainting* sobre C++. El ambiente de trabajo fue Visual Studio sobre el sistema operativo Windows. Para la manipulación inicial de las imágenes se empleó la biblioteca EmguCv. Igualmente, se utiliza el API de CUDA para C++.

A. Escenarios

Se efectuaron experimentos en diferentes niveles. Un primer nivel hace un estudio a fondo de los parámetros, realizando tantas ejecuciones como resulten de la combinación de todos éstos. Luego, en un segundo nivel se elaboran pruebas con las mejores combinaciones obtenidas de los experimentos de primer nivel en 4 imágenes distintas.

Existen diferentes parámetros que afectan el resultado luego de aplicar el algoritmo en cualquiera de sus versiones. Algunas de éstas variables tienen influencia significativa en el producto final. Dichos valores son: el lado de la ventana (L); el parámetro de propagación (K); la cantidad de candidatos (N); la cantidad de iteraciones para el procesamiento de textura (r); y la cantidad de iteraciones para aplicar la función de energía (n).

Por otro lado, se implementa el algoritmo secuencial (totalmente en la CPU); otro algoritmo en la GPU donde se ejecuta cada etapa a la vez, pero teniendo como entrada las muestras que se emplearon en la etapa de inicialización (GPU T1); e iterando r veces en las sub-etapas que la componen, teniendo como entrada a dicha etapa los mejores candidatos seleccionados en la etapa anterior (GPU T2).

B. Experimentos (Nivel 1)

Los experimentos de nivel 1 prueban el algoritmo considerando todas las combinaciones posibles de los diferentes parámetros descritos anteriormente. Para este estudio se realizaron 7680 ejecuciones, producto de las 384 combinaciones que surgen a partir de la composición de dichos parámetros, multiplicado por 20 para obtener un promedio.

El tiempo de ejecución del algoritmo depende de tres factores: la complejidad del algoritmo, el tamaño de la imagen y el tamaño del agujero o zona a reconstruir. Debido a la cantidad de pruebas a realizar y a la complejidad del algoritmo, se decide emplear una imagen relativamente pequeña de dimensiones 228×250 píxeles con un orificio pequeño. El objetivo de este experimento es elegir el sub-conjunto de las mejores combinaciones de los diferentes valores, para luego aplicar estas combinaciones a otros experimentos.

Entonces, se hace un estudio métrico donde se necesita una imagen de referencia que sea correcta, tomando una imagen

sin ningún tipo de pérdida de información, como la Figura 10a y se edita para causar pérdida de información como se ve en la Figura 10b. Luego, se hace una selección del área dañada como se observa en la Figura 10c y esa es la región a reconstruir. Con la Figura 10a, es posible tener un punto de referencia adecuado y realizar una diferencia píxel a píxel con cada imagen resultado.

(a) Original (b) Editada (c) Selección

Figura 10: Procedimiento Realizado Para los Experimentos en la Imagen Tomada del Nivel 1

Para evaluar los resultados de la imagen se calcula la diferencia promedio entre la imagen 10a y la imagen reconstruida a partir de 10c en la región tratada con la técnica. Dada la imagen original O y la imagen resultante R, se calcula en el espacio L*a*b la diferencia euclidiana promedio de los η píxeles sobre Ω como:

$$Dif(\varphi) = \frac{\sum_{i \in \Omega} \|\iota(O_i) - \iota(Ri)\|^2}{\eta} \quad (13)$$

En la Tabla III, se muestra el tiempo que tardó cada versión en alcanzar el resultado visual presentado en la Figura 11.

Tabla III: Tiempos de Ejecución para cada Versión

Versión	Tiempo	Parámetros
Secuencial	3 m y 19 s	$L = 9$, $K = 8$, $r = 3$, $n = 10$
GPU T1	1 m y 18 s	$L = 9$, $K = 32$, $n = 5$
GPU T2	33 s	$L = 5$, $K = 64$, $r = 3$, $n = 10$

(a) (b) (c)

Figura 11: Resultados Visuales de los Diferentes Algoritmos *k-Inpainting*

En la Figura 12 se puede visualizar el tiempo que tardó la versión secuencial en ejecutar las 384 combinaciones, el tiempo en ejecutarse las 192 combinaciones de la implementación *GPU T1* de la versión paralela y el tiempo en ejecutarse las 384 combinaciones en la implementación *GPU T2* de la versión paralela.

C. Experimentos (Nivel 2)

En los experimentos nivel 2, se tratarán las imágenes agrupadas en la Figura 13. Dichas imágenes serán procesadas

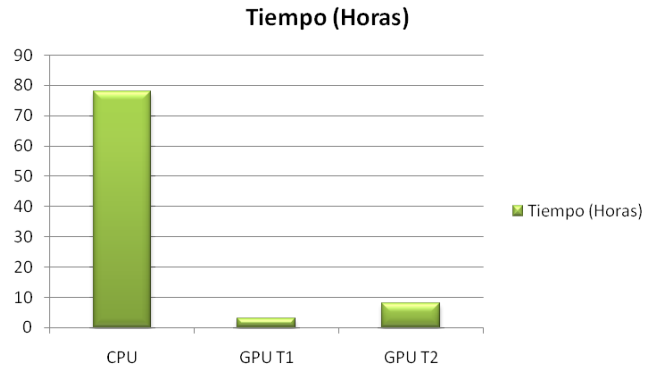


Figura 12: Tiempos de Ejecución de las Versiones de *k-Inpainting* en Reconstruir el Área Desconocida

con los parámetros escogidos en los experimentos de nivel 1. Sin embargo, solo se presentará la implementación GPU T1 dada la extensión de los resultados obtenidos por las otras implementaciones. Sin embargo, en [21] se puede explorar todas las pruebas realizadas. Como las imágenes son distintas, pueden existir casos especiales, en los cuales se deba variar el valor del tamaño de la ventana L o el valor de propagación K. Por otro lado, la Tabla IV resume las características de las imágenes a procesar.

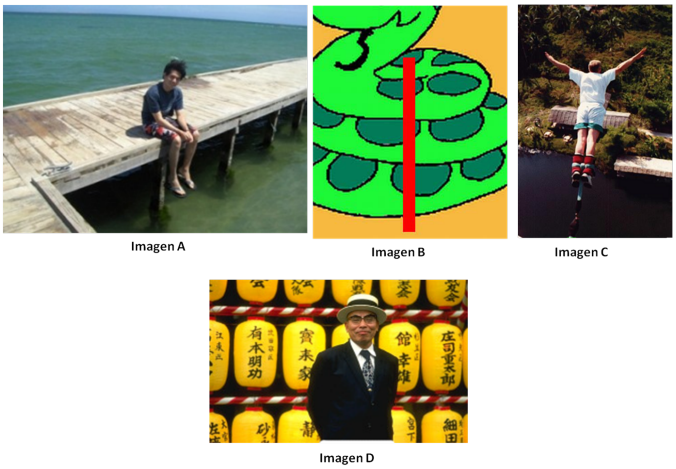


Figura 13: Imágenes de Prueba Para los Experimentos Nivel 2

Tabla IV: Características de las Imágenes a Procesar en los Experimentos Nivel 2

Imagen	Dimensiones en píxeles
A	301×226
B	292×291
C	206×308
D	256×162

Las imágenes serán tratadas por el orden indicado en la Figura 13, es decir, siguiendo la secuencia alfabética A,B,C,D. La primera imagen a procesar es la imagen A, la segunda imagen con la cual experimentar es la B y así sucesivamente.

1) Imagen A: En la imagen A, se observa una persona sentada en un puente. La idea es remover a dicha persona para

ver el paisaje de la fotografía. Luego que el usuario selecciona el área a restaurar rodeando al individuo en la foto, en la imagen queda un agujero. Este agujero está constituido por 6.398 píxeles, que forman el 9,4% de la imagen.

En la Tabla V se despliega el subconjunto de parámetros para procesar la región desconocida de la imagen con su respectivo tiempo de ejecución. La Figura 14 muestra los resultados obtenidos para cada combinación, donde se observa que los resultados son muy parecidos entre las imágenes 2, 3, 4, 5 y 6, por consiguiente el mejor resultado es aquel que tomó menos tiempo de ejecución, es decir, para este caso, el mejor resultado corresponde a la imagen 6.

Tabla V: Diversos Parámetros y Tiempo de Ejecución de la Imagen A

Resultado	Porcentaje	L	K	n	Tiempo
1	0,1%	9	16	10	6 minutos
2	0,05%	9	32	5	10 minutos
3	0,05%	5	64	5	8 minutos y 40 segundos
4	0,05%	9	32	10	10 minutos y 10 segundos
5	0,05%	9	64	5	27 minutos
6	0,1%	5	16	10	1 minuto y 40 segundos

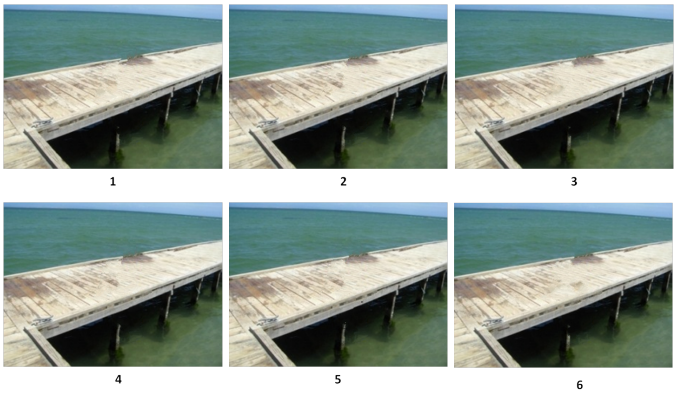


Figura 14: Resultados Luego de Procesar la Imagen A con los Parámetros de la Tabla V

Nótese la percepción visual de cada una de las imágenes con respecto a la restauración del área que se desea eliminar. La capacidad de reconstrucción visual es notable, mostrando un buen resultado el algoritmo.

2) Imagen B: La imagen B tiene un área dañada (rectángulo rojo) en la parte central de la imagen, afectando de forma transversal la estructura de la serpiente en dicha imagen. En este caso se busca restaurar la región. Luego que el usuario selecciona el área a tratar rodeando el área dañada, en la misma queda un orificio. Este orificio está constituido por 5.617 píxeles, que forman el 7,9% de la imagen B.

El subconjunto de parámetros se presentan en la Tabla VI para esta imagen. En la Figura 15 se observa el resultado para cada conjunto de parámetros en la tabla. Los resultados son muy parecidos entre las primeras cuatro imágenes, por lo que se selecciona como mejor resultado la que se ejecuta en menor tiempo entre ellas, que corresponde a la imagen 2.

Tabla VI: Diversos Parámetros y Tiempo de Ejecución de la Imagen B

Resultado	Porcentaje	L	K	n	Tiempo
1	0,1%	11	8	10	5 minutos
2	0,05%	11	8	10	3 minutos 45 segundos
3	0,05%	9	32	5	9 minutos
4	0,05%	9	32	10	9 minutos y 20 segundos
5	0,05%	5	64	10	8 minutos y 17 segundos
6	0,1%	9	16	10	5 minutos y 25 segundos

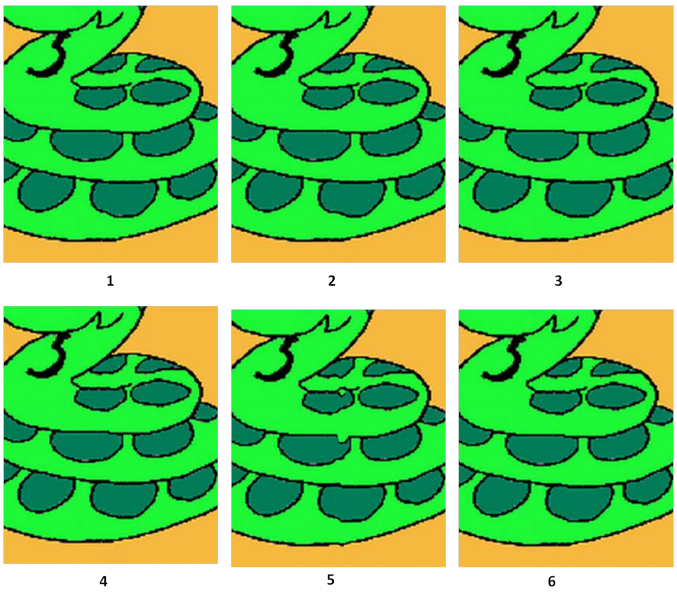


Figura 15: Resultados Después de Aplicar la Primera Versión del Algoritmo Paralelo con los Parámetros de la Tabla VI

3) Imagen C: En la imagen C se busca remover la persona que bloquea el paisaje, la cual forma el agujero que tiene 12.117 píxeles y constituye el 19,09% de la imagen.

La recopilación de los parámetros de esta ejecución se muestra en la Tabla VII, mostrando sus resultados visuales en la Figura 16.

Tabla VII: Diversos Parámetros y Tiempo de Ejecución de la Imagen C

Resultado	Porcentaje	L	K	n	Tiempo
1	0,1%	5	16	10	2 minutos y 41 segundos
2	0,1%	11	8	10	7 minutos
3	0,1%	9	32	10	16 minutos

4) Imagen D: Se busca remover el individuo presente en la foto y reconstruir el patrón detrás de él. Esta imagen es compleja, ya que el tamaño del agujero representa un porcentaje importante y tiene un fondo complicado. El agujero que queda en la imagen luego de seleccionar el área a tratar ocupa el 26,67% de la imagen con 11.061 píxeles. A continuación se despliega en la Tabla VIII los parámetros utilizados para procesar esta imagen con cada variante del algoritmo.

A diferencia de las imágenes anteriores, solo la primera combinación es mostrada, debido a que la imagen arroja mejores resultados con una ventana $L = 11$ para cualquier versión. Los parámetros de la segunda y tercera combinación de la tabla se escogieron a partir del resultado obtenido para la

1 2 3

Figura 16: Resultados Luego de Procesar la Imagen C con los Parámetros Indicados en la Tabla VII

Tabla VIII: Diversos Parámetros y Tiempo de Ejecución de la Imagen D

Versión	Porcentaje	L	K	r	n	Tiempo
CPU	0,05%	11	16	5	10	2 horas
GPU T1	0,05%	11	32	-	10	44 minutos
GPU T2	0,05%	11	32	5	5	50 minutos

primera. En la Figura 17 se observan las imágenes obtenidas a partir de estos parámetros en su proceso iterativo.

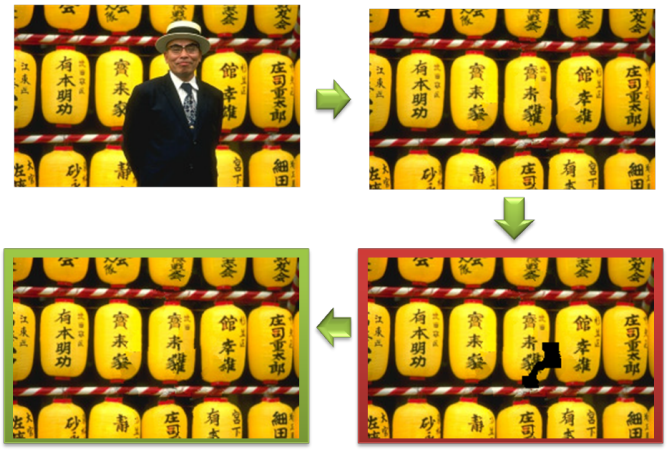


Figura 17: Diversas Etapas Iterativas Aplicadas en la Imagen D

Los resultados aquí presentados, fueron corroborados a su vez por una serie de encuestas para determinar la percepción visual de usuarios regulares antes las imágenes presentadas. Dicho estudio ayudo a distinguir qué imágenes (y bajo qué parámetros) se puede considerar a una imagen "visualmente agradable" con discontinuidades no notorias a simple vista.

VII. Consideraciones Finales

En este trabajo se presentó un nuevo algoritmo híbrido denominado *k-Inpainting* que se basa en dos trabajos concebidos de manera distinta, incluyendo modificaciones a dichos trabajos para acoplarse en un mismo algoritmo. Los trabajos base, manejan la síntesis de textura basada en parches [19], [8], o basada en píxeles [4], y adoptándo para el proceso de inicialización un campo aleatorio de Markov, la cual es una característica común en los algoritmos de síntesis de textura basada en píxeles. Básicamente, se sustituye la inicialización mediante un campo aleatorio de Markov, por un algoritmo de síntesis de textura para procesar la misma mediante parches, pero adaptándo este último a su vez a píxeles.

k-Inpainting provee buenos resultados y logra el objetivo de la técnica. La implementación funciona eficientemente cuando no hay partes importantes del agujero en los bordes de la imagen, completa bien los contornos, realiza una buena síntesis de textura, y no es estrictamente necesario aplicar una técnica de difusión al área que ha sido tratada o un post-procesamiento adicional. Adicionalmente, su rapidez depende más de las dimensiones de la región Ω a ser reconstruida y de los parámetros que del tamaño mismo de la imagen. Los parámetros que más influyen en el tiempo son las dimensiones del sistema de vecindad, el parámetro de propagación y el parámetros N. Hasta ahora no existe para *k-Inpainting* una combinación que funcione para todas las imágenes, por lo que sería necesario estimar estos valores también de forma automática. Un aspecto importante es que la selección a mano alzada no es la ideal, pues se pierden píxeles que podrían ayudar a reconstruir mejor el área a tratar.

En esta propuesta solo se contemplan las imágenes, sin embargo, pensando en una extensión para video, cuando se desea retirar un cuerpo en movimiento, se proveen más datos a las imágenes del video a reconstruir, ya que a través de las imágenes en otros momentos del mismo se puede extraer información. Por ello se considera que la reconstrucción de imágenes estáticas tiene una mayor complejidad al reconstruir un área donde había un cuerpo en movimiento.

El cálculo de la diferencia de vecindades requiere tiempo computacional cuando se procesa una cantidad considerable de píxeles. Este fenómeno ocurre incluso en las variantes paralelas, ya que esta función es secuencial en sí misma (i.e. un hilo a la vez).

Trabajos Futuros

Diversos cambios se proponen con el objetivo de mejorar nuestra propuesta. Hasta ahora, se ha construido el algoritmo de forma secuencial y dos variantes paralelas. También se ha construido una interfaz sencilla para que el usuario seleccione el área a tratar. Entre dichos cambios se encuentran:

- Elaborar otras herramientas para el usuario, como ofrecer una herramienta precisa de selección de la región a tratar.
- Hallar un método automático para detectar los parámetros más adecuados de una imagen de entrada.
- Incluir el uso de OpenMP o Numa en el algoritmo secuencial para acelerar los tiempos de respuesta.
- Emplear estructuras de datos que exploten más la capacidad de la tarjeta gráfica.

Para acelerar el tiempo de respuesta, se propone elaborar una versión basada en parches, en vez de basada en píxeles. En cuanto a la implementación, se propone migrar el código actual desarrollado en CUDA a un ambiente independiente de la arquitectura como Compute Shaders.

REFERENCIAS

[1] J. F. Aujol, S. Ladjal, and S. Masnou, *Exemplar-Based Inpainting From a Variational Point of View*, SIAM Journal on Mathematical Analysis, vol. 43, no. 3, pp. 1246-1285, 2010.

[2] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, *Image Inpainting*, in Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00), pp. 417-424, 2000.

[3] S. Li, *Markov Random Field Modelling in Image Analysis*, pp. 353-354, Springer, Londres, 1999.

[4] A. Bugeau, M. Bertalamio, V. Caselles, and G. Shapiro, *A Comprehensive Framework For Image Inpainting*, IEEE Transactions on Image Processing, vol. 19, no. 1, pp. 2634-2645, 2010.

[5] M. Taschler, *A Comparative Analysis of Image Inpainting Techniques*, Tesis de PhD, The University of York, 2006.

[6] R. Suthar and K. Patel, *A Survey on Various Image Inpainting Techniques to Restore Image*, International Journal of Engineering Research and Applications, vol. 4, no. 2, pp. 85-85, 2014.

[7] J. Joshua and G. Darsan, *Digital Inpainting Techniques - A Survey*, International Journal of Latest Research in Engineering and Technology (IJLRET), vol. 2, no. 1, pp. 34-36, 2016.

[8] C. Barnes, E. Shechtman, D. Goldman, and A. Finkelstein, *The Generalized Patchmatch Correspondence Algorithm*, in Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III, pp. 29-43, 2010.

[9] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, *PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing*, ACM Transactions on Graphics, vol. 28, no. 3, pp. 24:1, 24:11, 2009.

[10] K. Plataniotis and A. Venetsanopoulos, *Color Image Processing and Applications*, pp.32-39, Springer-Verla, Berlín, 2000.

[11] L-Y. Wei and M. Levoy, *Fast Texture Synthesis Using Tree-Structured Vector Quantization*, in Proceedings of the 27th annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00), pp. 479-488, 2000.

[12] N. Komodakis, *Image Completion Using Global Optimization*, in Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 442-452, 2006.

[13] K. Cao, K. Ding, G. Christensen, M. Raghavan, R. Amelon, and J. Reinhardt, *Unifying Vascular Information in Intensity - Based Non-Rigid Lung CT Registration*, Biomedical Image Registration, pp. 5-6, Springer, Alemania, 2010.

[14] P. Burt and E. Adelson, *The Laplacian Pyramid as a Compact Image Code*, IEEE Transactions On Communications, vol. 31, pp. 532-540, USA, 1983.

[15] R. Sedgewick, *Algoritmos en C++*, pp. 555-561, Addison - Wesley / Diaz de Santos, USA, 1995.

[16] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C*, 2da edición, Cambridge, USA, 2002.

[17] L. Demanet, B. Song, and T. Chan, *Image Inpainting by Correspondence Maps: a Deterministic Approach*, reporte técnico, Instituto Tecnológico de California y Universidad de California, Los Angeles, USA, 2000.

[18] A. Efros and T. Leung, *Texture Synthesis by Non-parametric Sampling*, in Proceedings of the Seventh IEEE International Conference on Computer Vision, IEEE Computer Society, 1999.

[19] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, *PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing*, ACM Transactions on Graphics, vol. 28, no. 3, 2009.

[20] S. Shin, T. Nishita, and S. Shin, *On Pixel-Based Texture Synthesis by Non-Parametric Sampling*, Computer & Graphics, vol. 30, no. 5, 2006.

[21] K. Pedrique, *Un Enfoque Híbrido del Algoritmo Inpainting bajo CPU y GPU*, Tesis de Pregrado, Universidad Central de Venezuela, 2011.