
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Computación Gráfica

# Aplicación de la técnica de *Inpainting* en un entorno Web empleando WebCL

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
Por el Bachiller
**Victor M. Meneses P.**
para optar el título de
**Licenciado en Computación**

Tutor: Prof. **Esmitt Ramírez**

Caracas, Octubre de 2016

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
**Centro de Computación Gráfica**

ACTA

Quienes suscriben, miembros del Jurado designado por el Consejo de Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, para examinar el Trabajo Especial de Grado titulado: "Aplicación de la técnica de Inpainting en un entorno web empleando WebCL", presentado por el bachiller Victor M. Meneses P. portador de la cédula de identidad V.- 21.374.434, **a los fines de optar al título de Licenciado en Computación,** dejan constancia de lo siguiente:

**Dicho trabajo, leído por cada uno de los miembros del jurado, se fijó el día martes 25** de Octubre de 2016, a las 8:00 a.m., para que su autor lo defendiera en forma pública en el CCG, mediante una presentación oral de su contenido, luego de lo cual se respondieron las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con la nota de 16 puntos.

En fe de lo cual se levanta la presente Acta, en Caracas a los veinticinco (25) días del mes de octubre del año dos mil dieciséis (2016), dejando constancia que actuó como coordinador del jurado el Profesor Esmitt Ramírez.

Prof. Esmitt Ramírez (Tutor)

Profa. Adélis Nieves (Jurado)

Prof. Héctor Navarro (Jurado)

# Resumen

En el procesamiento digital de imágenes, se aplica un conjunto de técnicas que tienen como objeto 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 imágenes una de las técnicas empleadas se denomina *Inpainting*, la cual modifica una imagen para reconstruir áreas deterioradas o eliminadas. En la actualidad, las técnicas de *Inpainting*, ocupan un amplio campo de investigación y desarrollo en el área de la Computación Gráfica.

Actualmente existen técnicas de *Inpainting* que hacen sus soluciones en paquetes comerciales/libres y algoritmos. Sin embargo, muchos ofrecen resultados muy diversos y variados basados en el uso de múltiples parámetros.

Por ello, seria ideal poder contar con una herramienta que ofrezca la parametrización y obtención de resultados en una plataforma independiente del sistema operativo. En este trabajo se presenta la solución de la técnica *Inpainting*, pensada para un navegador web, donde se pueda realizar una variedad de procesamientos a una imagen (además de la técnica *Inpaiting*). Las pruebas demuestran el comportamiento de la aplicación web, más los trabajos que se deben hacer para completar la solución para la técnica *Inpaiting* en el navegador.

**Palabras Claves**: *Inpainting*, Procesamiento digital de imágenes, reconstrucción de imágenes, Imagen digital, WebCL.

# Tabla de Contenidos

# Introducción

La restauración de imágenes es un arte, técnica y práctica tan vieja como la creación artística misma, en donde el restaurador busca recuperar o corregir la zona de una imagen, de tal forma que no sea detectada por el usuario. Actualmente esta técnica es utilizada con mucha frecuencia para restaurar una fotografía antigua que previamente se halla digitalizado, remover objetos no deseados y reconstruir áreas de un objeto específico.

Los restauradores profesionales de fotografías están reemplazando los métodos manuales por los digitales, y la técnica de *Inpainting* es una de estas nuevas tendencias. Esta técnica con lleva un alto costo computacional, ya que para ésta solución se necesita el uso de una gran variedad de operaciones matemáticas, con las cuales se permite reconstruir la zona de interés de forma que no sea detectado el cambio por el usuario.

Este documento tiene como finalidad presentar el desarrollo de una página web donde el usuario pueda realizar la restauración a una imagen deseada mediante la técnica *Inpainting*, utilizando la tecnología WebCL, la cual permite la integración de OpenCL en una aplicación web y con ello se aprovecha todos los beneficios que nos provee la GPU y multi-core CPU. Con esto se realiza procesamiento paralelo en un navegador web, así logrando una aceleración significativa en los procesamientos de imágenes que se realizan.

En el Capítulo 1 están los conceptos básicos, las características más importantes de la técnica *Inpainting*, el estado donde se encuentra la técnica a lo largo de los últimos años, la justificación y motivación del tema, y los objetivos generales y específicos. En el Capítulo 2 ésta la implementación del algoritmo *Inpainting* paso a paso, junto a los distintos problemas que tiene al realizar una implementación de WebCL, en el Capítulo 3 están las pruebas y resultados obtenidos por la aplicación web y finalmente el Capítulo 4 describe las conclusiones y posibles trabajos a futuro.

# Capítulo 1

# Marco Teórico

## 1. Procesamiento Digital de Imágenes (PDI)

En este capítulo se explican los conceptos básicos que son necesarios para comprender más a fondo este documento.

El Procesamiento Digital de Imágenes (PDI) es el conjunto de técnicas que se aplican a las imágenes digitales por medio de un computador, con el objetivo de mejorar la calidad o facilitar la búsqueda de información [1].

### 1.1. Imagen digital

El concepto de imagen está asociado a una función bidimensional $f(x, y)$, cuya amplitud o valor será el grado de iluminación (intensidad de la luz), en el espacio de coordenadas $(x, y)$ de la imagen para cada uno de sus puntos. Cuando el valor de amplitud de $f$ y su dominio son cantidades finitas y discretas, se dice que es una imagen digital [1]. En la Figura 1.1 se muestra una de las herramientas mas empleadas para generar imágenes digitales.

Figura 1.1: La cámara es una de las herramientas más empleadas, que genera imágenes digitales del ambiente.

## 1.2. Interpolación

El proceso de interpolación es la obtención de nuevos puntos partiendo del conocimiento de un conjunto de valores discretos, como el caso de un usuario que conoce el valor de una función $f(x)$ en una serie de puntos $x_1, x_2, ..., x_N$ (en la Figura 1.2, se ve un ejemplo de aplicar interpolacion sobre 7 puntos), los cuales son obtenidos por muestreo o a partir de un experimento, pero no se conoce una expresión analítica de $f(x)$ que permita calcular el valor de la función para un punto arbitrario, y por ello se pretende construir una función que ajuste esa serie de puntos con el objetivo de obtener el valor deseado [2].

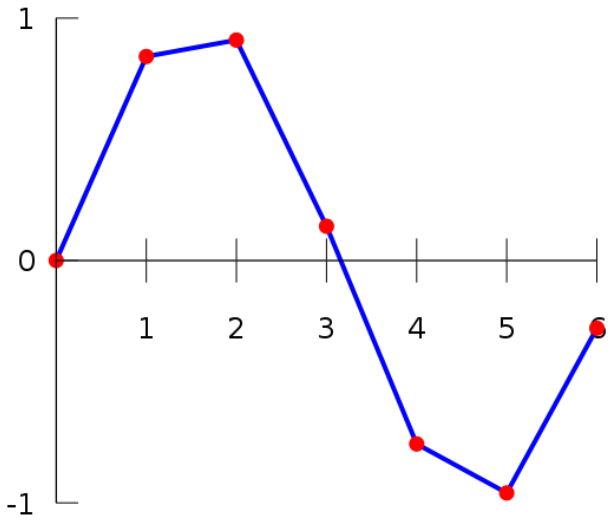


Figura 1.2: La línea azul representa la interpolación lineal entre los puntos rojos.

## 1.3. Píxel

Un píxel es la menor unidad de color que conforma una imagen digital, ya sea una fotografía, video o fotograma; esta palabra proviene de la unión o abreviatura de las palabras inglesas *Picture Elements* (elementos de una imagen).

Ampliando lo suficiente una imagen (aplicando zoom a la zona deseada), pueden observarse los píxeles que componen la figura. Las imágenes se forman como una sucesión de píxeles. La sucesión marca la coherencia de la información presentada, siendo su conjunto una matriz coherente de información para el uso digital [3].

## 1.4. RGB

RGB (estas siglas en inglés son *red*, *green*, *blue*) es un término empleado para la composición del color en términos de intensidad de los colores primarios de la luz [4].

El modelo RGB se basa en la síntesis aditiva de color. Empleando la luminosidad del rojo, el verde y el azul en diferentes proporciones, se produce el resto de los colores. Como se puede evidenciar en la Figura 1.3.

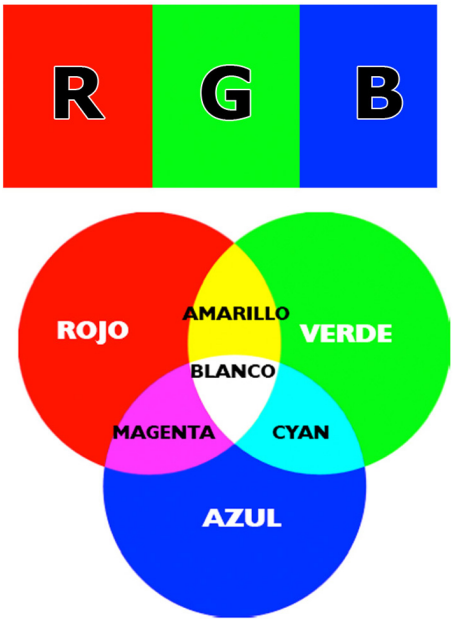


Figura 1.3: Modelo aditivo de colores rojo, verde, azul.

## 1.5. Comisión Internacional de la Iluminación (CIE)

La Comisión Internacional de la Iluminación es también conocida por las siglas CIE, éstas vienen de su nombre en francés *Commission internationale de l'éclairage*; fundada en 1913. Estos son la autoridad internacional en luz, iluminación, color y espacios de color.

CIE esta compuesta por 8 divisiones principales visión y color, medida de luz y radiación, ambiente interior y diseño de iluminación, iluminación y señalización para el transporte, iluminación exterior y otras aplicaciones, fotobiología y fotoquímica, aspectos generales de la luz (Inactivo) y tecnología de la imagen [5].

## 1.6. XYZ (CIE 1931)

En el modelo XYZ, X es una mezcla tendiente a la curva de sensibilidad del rojo al verde, donde Y significa luminosidad y Z es aproximadamente igual al estímulo de azul (Como se puede ver en la Figura 1.4); este fue creado en 1931 por CIE. El modelo CIE XYZ se basa en 3 primarios imaginarios con caracterización espectral (X, Y y Z), estos son los que representan el color. Éstos se combinan para formar todos los colores visibles por el observador [6].

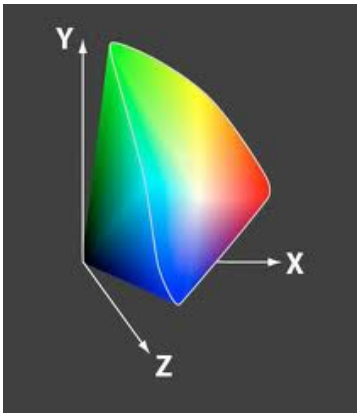


Figura 1.4: Diagrama de cromaticidad del espacio de color XYZ.

## 1.7. LAB (CIE 1976)

Las siglas LAB se refieren al espacio de color tridimensional, en donde L o L* es luminosidad de negro a blanco, A o a* va de rojo a verde y B o b* es la gradiente del azul (Como se puede ver en la Figura 1.5). El propósito de este, es producir un espacio de color que sea más perceptivamente lineal que otros espacios de color y con ello obtener un resultado más exacto al momento de sacar la distancia entre dos colores [6].

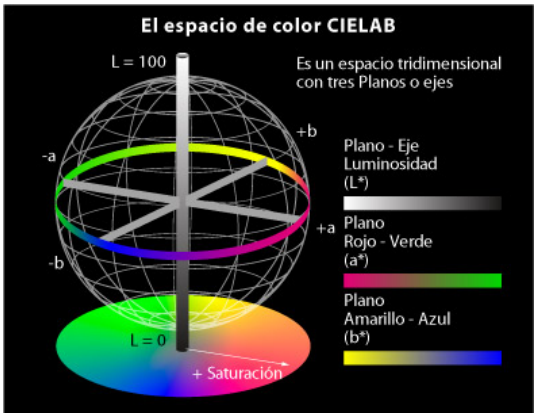


Figura 1.5: Espacio de color CIELAB.

## 1.8. LUV (CIE 1976)

El espacio de color CIELUV 1976 es un intento de proporcionar un espacio de color perceptualmente uniforme, como se puede ver en la Figura 1.6. Este, junto a CIELAB, es uno de los espacios de color más usados, su importancia es conseguir con mayor exactitud la distancia entre dos colores; LUV (CIE 1976), es derivado del CIE 1931 XYZ o las coordenadas de cromaticidad (x, y) [7].

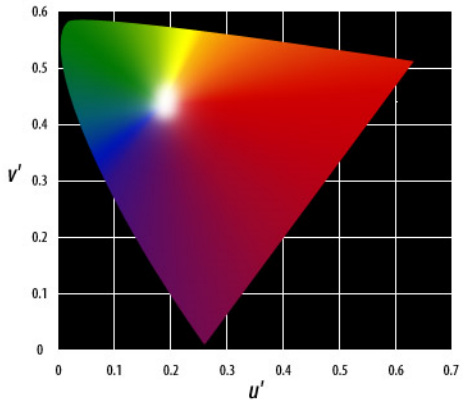


Figura 1.6: Diagrama de cromaticidad del espacio de color LUV.

## 1.9. LCH (CIE 1976)

Las siglas LCH se refieren al espacio de color tridimensional, en donde L o L* es luminosidad de negro a blanco, C o c* representa croma o saturación y H o h* es el ángulo de matiz (ver Figura 1.7). Ellos determinan el color de un objeto dentro del espacio de color, y muestran los valores para cada coordenada L*, C*, y h* [8].

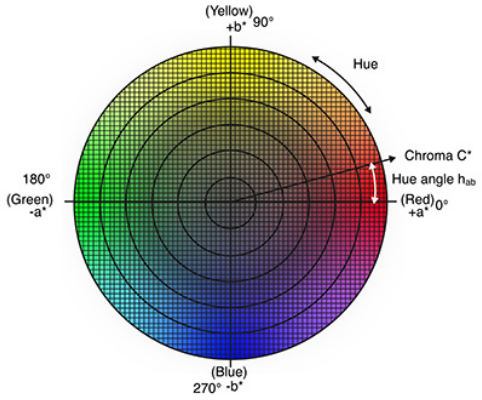


Figura 1.7: Diagrama de cromaticidad del espacio de color LCH.

## 1.10. Convolución

En procesamiento de imágenes, la convolución corresponde a la extensión del caso unidimensional, mediante la cual una señal cualquiera podía ser procesada con un filtro arbitrario, y con una respuesta impulsiva conocida.

Para aplicar un filtro arbitrario se multiplica sobre todos los píxeles de la imagen, una matriz como la de las Figuras 1.9, 1.11 y 1.12; y en PDI ésta matriz es llamada como máscara.

La máscara de convolución, tiene por lo general un número impar de filas y columnas, y su tamaño frecuentemente es 3 x 3 píxeles. Adicionalmente, define el tamaño de la vecindad dentro de la imagen de manera que sea del mismo tamaño que la máscara [9].

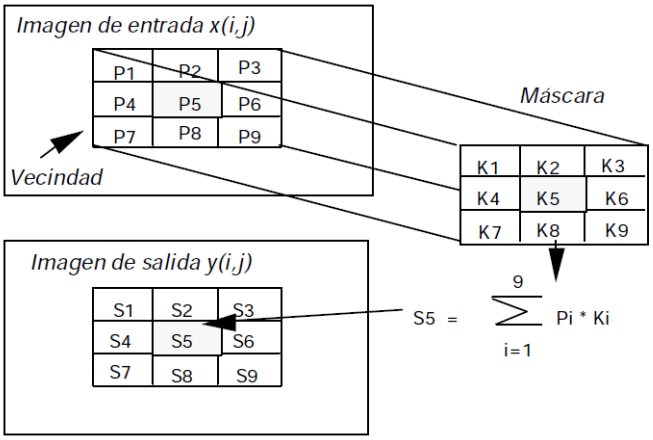


Figura 1.8: Ilustración del proceso de convolución con una máscara.

En la Figura 1.8 se ilustra el proceso de convolución, según el cual, para un píxel dado dentro de la imagen de entrada *x(i,j)*, cada píxel de la vecindad es multiplicado por el píxel correspondiente en la máscara de convolución, así mismo cada uno de estos productos es sumado, de manera que el nuevo valor del píxel en la imagen de salida *y(i,j)* estará dado por la suma de todos estos productos. El procesamiento de toda la imagen se realiza desplazando la máscara y repitiendo para cada punto el mismo procedimiento.

## 1.11. Sistema de vecindad

Las técnicas de procesamiento basadas en una región tienen muchas aplicaciones en la obtención de primitivas características de la imagen, como por ejemplo la extracción de contornos, para realzar los bordes, para suavizar una imagen, para introducir borrosidad y para

atenuar el ruido aleatorio; y así poder extraer información relevante de la imagen.

El grupo de píxeles que se estudia en este caso, se denomina usualmente vecindad. Por lo general la vecindad es una matriz bidimensional de valores de píxeles con un número impar de filas y columnas. El píxel de interés usualmente se ubica en el centro de la vecindad, el cual es reemplazado por un nuevo valor, producto de la aplicación de un algoritmo en nuestra vecindad a tratar.

Al utilizar una vecindad en el procesamiento, se puede aprovechar la información acerca del comportamiento regional de la imagen en cuestión, mejor conocida como frecuencia espacial la cual, podría definirse como la tasa de cambio de la intensidad de los píxeles, dividido por la distancia sobre la cual ocurre el cambio. La frecuencia espacial tiene componentes en las direcciones horizontal y vertical dentro de la imagen.

En la implementación de estas técnicas de procesamiento, se utilizan métodos lineales como la convolución, y métodos no lineales, como el filtro de mediana (los métodos no lineales serán explicados en la siguiente sección). El procedimiento que se sigue es el siguiente:

a) Se realiza una sola pasada sobre la imagen de entrada realizando un barrido píxel por píxel, según las filas y columnas.

b) Cada píxel de la imagen de entrada es procesado, considerando una vecindad del mismo y utilizando un algoritmo apropiado.

c) El nuevo valor del píxel, obtenido de acuerdo a lo especificado en b), es ubicado en la imagen de salida, ocupando la misma posición que ocupaba en la imagen de entrada.

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 3 | 4 | 5 |
| 4 | 2 | 1 | 2 | 4 |
| 3 | 1 | **X** | 1 | 3 |
| 4 | 2 | 1 | 2 | 4 |
| 5 | 4 | 3 | 4 | 5 |

Figura 1.9: Comportamiento del sistema de vecindad.

El hecho de considerar los píxeles de una vecindad, hace que las técnicas de procesamiento basadas en una región tengan un mayor costo de cálculo numérico que las técnicas basadas en un solo punto. Este costo dependerá del tamaño de la vecindad a considerar, así como del tipo de representación numérica utilizada [9].

## 1.12. Filtros

Los filtros son operaciones que se aplican a los píxeles de una imagen digital para optimizarla, conseguir un efecto especial en ella o enfatizar cierta información [1], como:

- **Suavizar la imagen**: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.
- **Eliminar ruido**: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen, como en la transmisión.
- **Realzar bordes**: destacar los bordes que se localizan en una imagen.
- **Detectar bordes**: detectar los píxeles donde se produce un cambio brusco en la función intensidad.

## 1.13. Filtro Sobel

El filtro Sobel detecta los bordes horizontales y verticales de manera separada sobre una imagen en escala de grises. Las imágenes en color se convierten a RGB en niveles de grises.

En la Figura 1.10 se muestra el resultado de aplicar el filtro de Sobel, cuyo resultado muestra una imagen con líneas blancas y algunos restos de color, la cual representa los cambios de color que tiene la imagen [10].

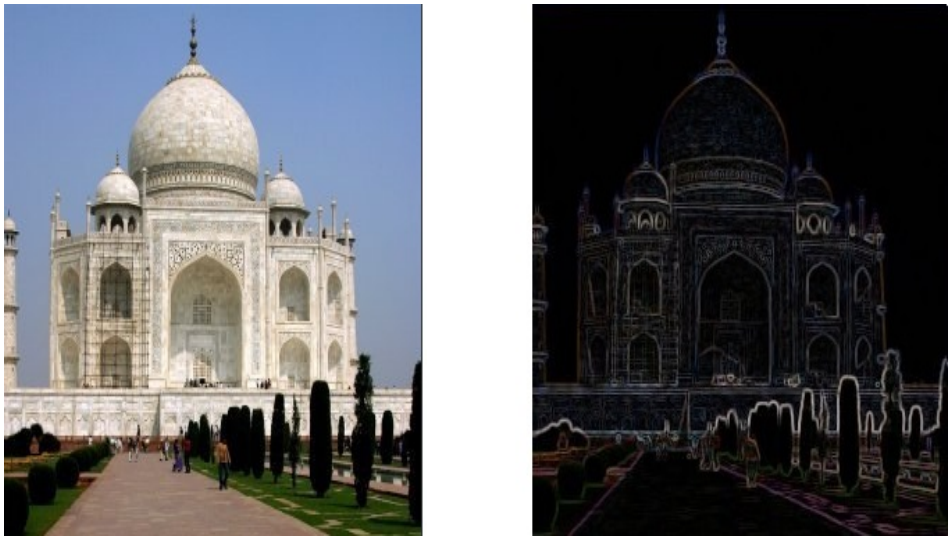

Figura 1.10: Ejemplo de aplicar el filtro sobel.

- **Sobel en *x***: Aplicar el filtro de Sobel en sentido horizontal.
- **Sobel en *y***: Aplicar el filtro de Sobel en sentido vertical.

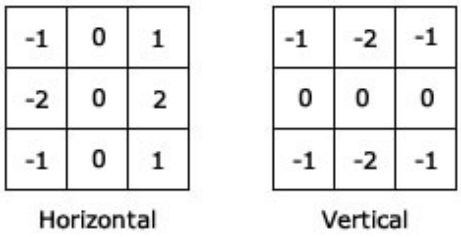


Figura 1.11: Máscara Sobel.

## 1.14. Filtro Laplaciano

El filtro Laplaciano es una medida 2D isotrópica de la segunda derivada espacial de una imagen. El Laplaciano de una imagen destaca las regiones donde hay cambios bruscos de intensidad y por lo tanto se suele utilizar para detección de bordes.

El Laplaciano se aplica frecuentemente a una imagen que previamente ha sido suavizada mediante un filtro Gaussiano de suavizado, con el fin de reducir su sensibilidad al ruido [11].

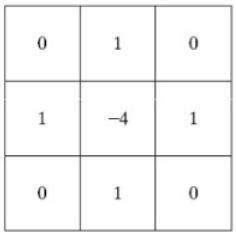


Figura 1.12: Máscara del filtro Laplaciano.

## 1.15. Filtro Mediana

La función principal del filtro mediana es forzar a los puntos con valores de intensidad muy distintos a sus vecinos, puesto que así se obtendrá valores mas próximos entre ellos, así se logra eliminar los picos de intensidad que aparecen en áreas aisladas [11]. Un ejemplo de la utilidad de este tipo filtro se muestra en la Figura 1.13.

Figura 1.13: Ejemplo de aplicar el filtro Mediana en una imagen con ruido.

## 1.16. Kernel

El kernel se entiende como una matriz de coeficientes, donde el entorno del punto $(x, y)$ que se considera en la imagen para obtener $g(x, y)$ está determinado por el tamaño y forma del kernel seleccionado. Entonces, la suma ponderada de estas funciones es un estimador para aproximar la función de densidad desconocida [12].

## 1.17. Kernel Gaussiano

Este kernel es un caso especial, ya que cada kernel influye en todos los otros kernels colocados en los puntos de la muestra, dando así que la suma resultante sea continua y suave [12].

$$G_1D(x;\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{x^2}{2\sigma^2}}, x \in (-\infty, +\infty) \tag{1.1}$$

La ecuación 1.1 representa el cálculo del Kernel Gaussiano ($G_1D(x;\sigma)$) unidimensional ($x$) y $\sigma$ es la anchura que va a tener este kernel.

$$G_ND(\vec{x};\sigma) = \frac{1}{(\sqrt{2\pi}\sigma)^N} e^{\frac{|x|^2}{2\sigma^2}} \tag{1.2}$$

La ecuación 1.2 representa el cálculo del Kernel Gaussiano ($G_ND(\vec{x};\sigma)$) en N dimensiones, $N$ es la cantidad de dimensiones representadas en $\vec{x}$ y $\sigma$ es la anchura que va a tener este kernel.

## 1.18. Poisson Blending

Poisson Blending es una técnica que mezcla los objetos deseados de dos o mas imágenes, para así obtener una imagen final que unifica todos esos elementos; principalmente este algoritmo selecciona el área deseada de una primera imagen, para luego escoger en que área se desea colocar de una segunda imagen, como se puede ver en el panel fuentes/destino de la Figura 1.14. Después de haber seleccionado las distintas áreas, se mezcla las imágenes en una sola como se puede ver en el panel Mezcla de la Figura 1.14. Para así finalmente empezar la mezcla entre los píxeles del entorno de la nueva imagen con la agregada, así obteniendo el resultado del panel Integración de la Figura 1.14 [13].

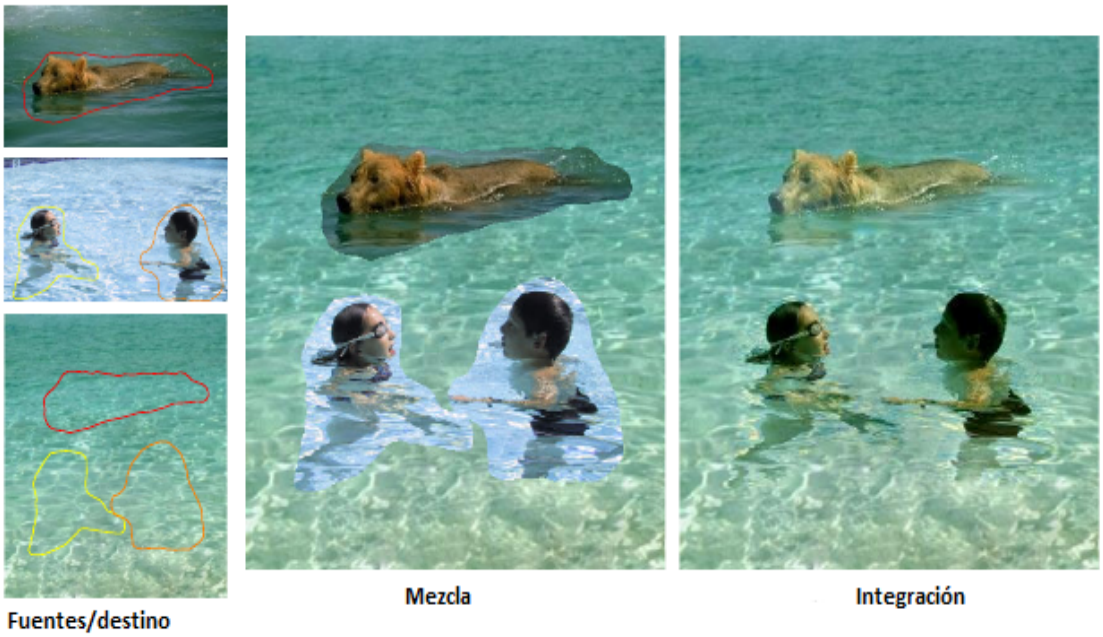


Figura 1.14: Ejemplo Poisson Blending.

Viendo el resultado de la Figura 1.14 se puede observar una diferencia muy notable en las regiones seleccionadas, ya que el color del agua en las tres fuentes son diferentes. Incluso, si se hacen coincidir los fondos también podemos ver las costuras entre estas regiones.

Esto ocurre porque la vista es mucho más sensible al gradiente de la intensidad global de una imagen, eso quiere decir, que se desconoce el valor exacto de cada píxel, pero si cambia el valor en algún lugar, podemos saber fácilmente que cambia.

Para evitar este tipo de problemas, lo recomendable es que los píxeles de la imagen original tengan una mayor probabilidad, de este modo se evita que se genere un cambio tan notorio.

# 2. *Inpainting*

*Inpainting* o también llamado restauración de imagen, es un proceso que permite recuperar la parte deteriorada de una imagen o esconder algo indeseable en ésta. Es decir, que se desea mejorar la calidad de la imagen dependiendo de la necesidad del usuario.

Es decir que *Inpainting* o retoque, se refiere a la restauración de imágenes mediante la aplicación de una o más técnicas numéricas, que permiten recuperar datos de una imagen a partir de la información disponible en su entorno.

En este capítulo se dará a conocer en qué consiste dicho proceso y también se realizará una revisión de los últimos avances de esta técnica [14].

## 2.1. Definición

Es un proceso por el cual se modifica una imagen o video, de tal manera que sea difícil detectar el cambio realizado, esta es una tecnología muy utilizada para la restauración de viejas pinturas y ha sido un área fundamental de la investigación en el procesamiento digital de imágenes.

Para explicar el principio de *Inpainting* usaremos la Figura 1.15, la cual trabaja sobre una región $\Omega$ a ser restaurada, este método se le aplica sobre el punto *p* situado en el límite $\delta\Omega$. De modo que, se toma una pequeña área $B_\epsilon(p)$ de tamaño $\epsilon$ con respecto a la imagen conocida alrededor de *p* (como se puede notar en la Figura 1.15 (a)), así pues con ello podremos determinar el valor final del punto *p* en la imagen. Considerando como prioridad los valores de escala de grises del área $B_\epsilon(p)$, así se pueden denotar los cambios de colores [14, 15, 16].

De esta pequeña área $\epsilon$, se considera el primer punto de aproximación $I_q(p)$ de la imagen para el punto *p*, para esta aproximación se usa $I(q)$ y el gradiente $\nabla I(q)$ con respecto al punto *q* ( ver en Figura 1.15 (b)):

$$I_q(p) = I(q) + \nabla I(q)(p - q) \tag{1.3}$$

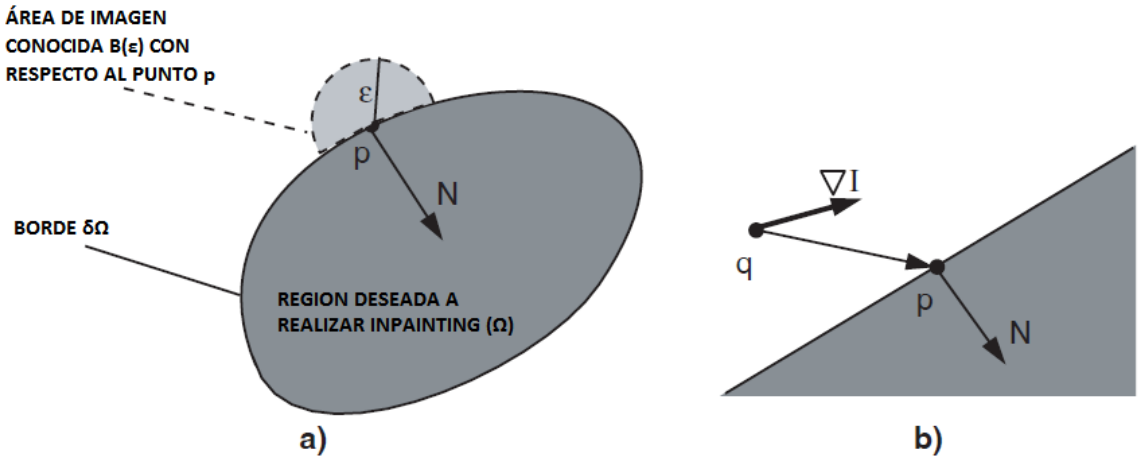


Figura 1.15: Principio de *Inpainting*.

## 2.2. Síntesis de textura

Una textura en computación gráfica, es definida como un conjunto de píxeles que se encuentran sobre un espacio 2D finito, que muestran características de superficies tales como terrenos, plantas, minerales, pelaje, piel, etc. La síntesis de textura es un mecanismo para generar texturas, mediante el uso de la información de textura contenida en su vecindad, este método puede producir imágenes de alta definición con el manejo adecuado de los bordes [17].

El problema de síntesis de textura puede ser definido como sigue: dada una muestra de textura, se sintetiza una nueva que al ser percibida por un observador, parece generada por el mismo (Figura 1.16).

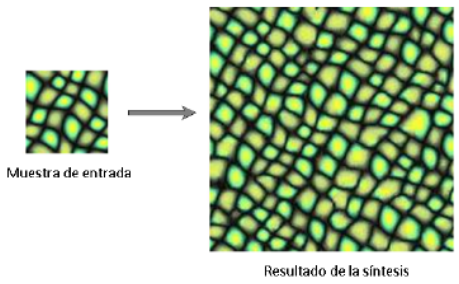


Figura 1.16: Resultado de aplicar síntesis de textura.

Este proceso ha sido un tópico activo de investigación, empleado como mecanismo para

verificar el método de análisis de textura. Una textura puede ser sintetizada basada en píxeles o en parches.

### Síntesis de textura basada en píxeles

Diversos métodos sintetizan una textura haciéndola crecer de adentro hacia fuera, píxel por píxel, a partir de una semilla inicial o replicando un píxel a la vez. Esta forma de sintetizar la textura fue introducida por Efros y Leung en su trabajo *"Texture Synthesis by Non-Parametric Sampling"* [17].

Este método es un poco lento en comparación con el de parches, puesto que este método se realiza el procesamiento píxel por píxel, lo cual acarrea una mayor carga de procesamiento, y por cómo se trabaja es más probable que se noten cambios abruptos entre píxeles (dependiendo de la secuencias de píxeles que se propaguen).

### Síntesis de textura basada en parches

La idea de síntesis de textura por parches, se introduce con el trabajo de Xu titulado *"Chaos Mosaic: Fast and Memory Efficient Texture Synthesis"* [18] en el cual se busca construir texturas procedurales, sintetizando grandes texturas a partir de una muestra inicial. Para ello, utilizan como base el algoritmo *chaos mosaic*, el cual permite sintetizar texturas con una distribución equilibrada y estocástica visual, de las características locales de la muestra de entrada.

En la Figura 1.17 se sigue los siguientes pasos de la técnica *chaos mosaic*:

- En la parte superior izquierda de la fotografía mostrada, no coinciden las características en el borde límite del bloque, este se encuentra marcado por líneas negras.
- En la parte superior derecha de la imagen, es la misma que la figura de la izquierda, pero las líneas negras se retiran para obtener una mejor visualización de las características que no coinciden.
- En la imagen inferior izquierda, un par capas de píxeles cerca de la orilla están oscurecidas para formar un problema de síntesis de textura limitada.
- La imagen inferior derecha, representa el resultado de la técnica *chaos mosaic*.

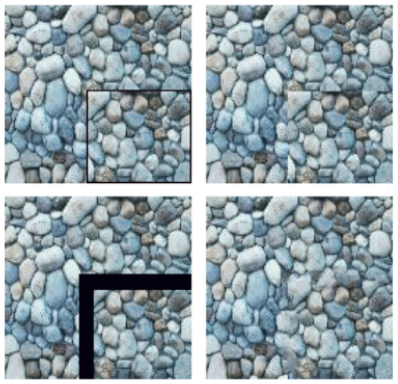

Figura 1.17: Resultado de aplicar *chaos mosaic*.

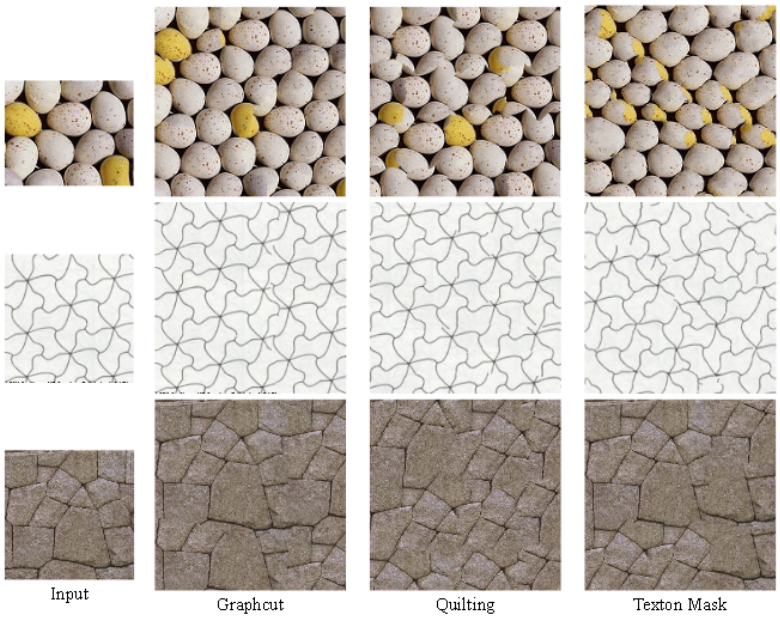


Figura 1.18: Resultado de varias técnicas de síntesis de textura basada en parche (*Graphcut, Quilting y Texton Mask*).

Se explicaran algunas de las técnicas de síntesis de textura basadas en parches:

*Graphcut*: La técnica *Graphcut* es usada para resolver de manera eficiente una amplia variedad de problemas de visión por computador de bajo nivel, tales como el suavizado de imagen, el problema de correspondencia estéreo, entre otros. Estos se pueden formular en

términos de minimización de la energía. En esta técnica se aplica a los modelos una optimización de las técnicas *max-flow*, y *min-cut*.

*Quilting*: La técnica *Quilting* se basa principalmente en buscar para cada parche, la textura de entrada que satisface las restricciones de solapamiento (arriba e izquierda) dentro de cierta tolerancia de error, luego aleatoriamente recoger uno de esos bloques candidatos, y finalmente calcular la superficie de error entre el bloque recién elegido, y los antiguos bloques en la región de superposición. Con esto se encuentra el mínimo coste de la ruta a lo largo de esta superficie.

*Texton Mask*: La técnica *texton mask*, permite calcular de forma automática desde las texturas originales, cuales son los patrones con mayor grado de similitud entre ellos. Estos elementos, una vez identificados, son usados para propagarse a lo largo de la imagen.

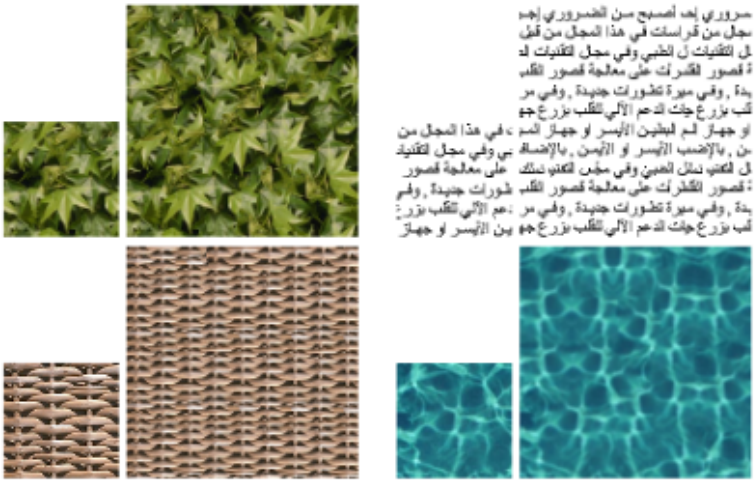

Figura 1.19: Resultado de la técnica basada en parche de Wu.[19]

La mayoría de los métodos basados en parches, han utilizado sólo la información del color, cuando se busca una ubicación adecuada de un parche. Sin embargo Wu en su investigación "*Feature matching and deformation for texture synthesis*", introduce un método híbrido que considera las características geométricas y de color simultáneamente, para minimizar la discontinuidad visual entre parches adyacentes. En los casos donde las discontinuidades generen artefactos inevitables, se remueve explícitamente la deformación del parche [19].

El cual obtiene los resultados vistos en la Figura 1.19, donde se puede notar un mejor resultado que las técnicas descritas en la Figura 1.18.

# 3. Estado del arte

El estado del arte o estado de la técnica, consiste en la descripción de los trabajos más importantes o llamativos que se han realizado hasta la fecha sobre *Inpainting*, estos serán las bases para este trabajo especial de grado.

## 3.1. Efros y Leung (1999)

Efros empieza su algoritmo definiendo una máscara, y un área específica de origen. A estos se le detectan los bordes del área ocluida, y con ellos se ordenan de forma descendente todos los píxeles en el borde por el número de vecinos conocidos. Después, se define una ventana centrada en cada píxel elegido a restaurar [17].

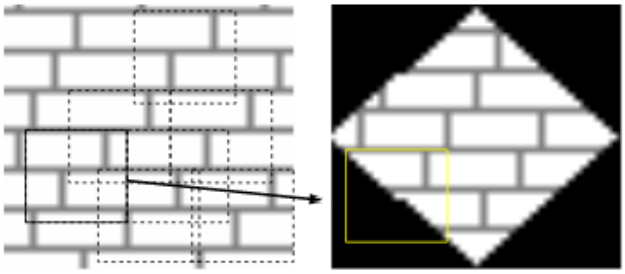

Figura 1.20: Síntesis de textura por píxel.

Esta ventana tiene un tamaño que se rige por un parámetro y se utiliza en la búsqueda de bloques similares, esta medida de similitud viene dada por la suma de las diferencias al cuadrado (SSD).

Para preservar el carácter local de la textura, se utiliza un kernel gaussiano, que tiene como objetivo controlar la influencia de los píxeles situados una mayor distancia de la zona ocluida. Considerando:

$$d = d_{SSD} * G. \tag{1.4}$$

Dependiendo del valor de la suma de las diferencias al cuadrado, se obtiene una colección de bloques candidatos. Considerando:

$$\begin{aligned} w_{best} = arg_w min\ d(w(p), w) \subset I_{smp}, \\ d(w(p), w) < (1+\epsilon) d(w(p), w_{best}), \epsilon = 0{,}1, \end{aligned} \tag{1.5}$$

En la ecuación 1.5 el píxel que será procesado es *p*, $I_{smp}$ representa el área específica de origen y $d(w(p), w)$ describe la distancia desde una ventana de tamaño *w* centrado en píxel *p* a un bloque del mismo tamaño $w_{best}$, donde $w_{best}$ es el que tiene la menor diferencia al cuadrado.

Los bloques candidatos serán elegidos al azar y la información de color de su píxel central se asigna al píxel en el borde de la ventana [17].

## 3.2. Marcelo Bertalmio y Guillermo Sapiro (2000)

La técnica que propone Bertalmio para el año 2000 es la más innovadora, ya que propone una técnica que no requiere la intervención del usuario, el usuario solo selecciona la región deseada a hacer *Inpainting* [14].

Figura 1.21: Imagen resultante de aplicar el algoritmo de Bertalmio, sobre el bastón que sostiene el señor en la primera imagen, como se puede notar el algoritmo tiene un fallo, que no le permite reproducir bien la textura.

Este algoritmo es capaz de cubrir simultáneamente regiones rodeadas de diferentes orígenes, sin que el usuario especifique lo que desea colocar. Para el año 2000 este método es el más innovador y el comienzo de la técnica *Inpainting*, para este trabajo se usa un enfoque basado en las técnicas de síntesis de textura.

Considerando el área ocluida $\Omega$ y el contorno de la región $\delta\Omega$ (descritos en el Capítulo 1 Sección 2.1), el método que propone Bertalmio propaga la información a lo largo de las líneas alrededor del contorno $\delta\Omega$; su algortimo opera iterativamente y crea una familia de

imágenes, cada imagen representa una versión mejorada de la anterior.

$$I^{n+1}(i,j) = I^n(i,j) + \Delta t I_t^n(i,j), \forall (i,j) \in \Omega, \tag{1.6}$$

Donde $I^{n+1}(i,j)$ es la intensidad del píxel que tiene las coordenadas $(i,j)$ en el momento $n+1$, $\Delta t$ es la mejora o la tasa de cambio y $I_t{}^n(i,j)$ corresponde a una actualización de la imagen en el momento $n$.

Esta actualización incluye la información a ser propagada y la dirección de propagación se calcula en la ecuación 1.7.

$$I_t{}^n(i,j) = \left( \overrightarrow{\delta L^n}(i,j) \frac{\vec{N}(i,j,n)}{\left|\vec{N}(i,j,n)\right|} \right) \left|\nabla I^n(i,j)\right|, \tag{1.7}$$

Donde $\delta \vec{L}^n$ es un vector que indica el cambio de intensidad en la imagen, este vector es obtenido después de aplicar el operador de Laplace sobre la imagen. Y $\frac{\vec{N}(i,j,n)}{\left|\vec{N}(i,j,n)\right|}$ representa la dirección de la línea, esta se expresa:

$$\frac{\vec{N}(i,j,n)}{\left|\vec{N}(i,j,n)\right|} = \frac{(-I_y^n(i,j), I_x^n(i,j))}{\sqrt{(I_x^n(i,j))^2 + (I_y^n(i,j))^2 + \epsilon}}, \tag{1.8}$$

Donde $\epsilon$ es un valor pequeño (mayor que cero) destinado a evitar la división entre 0 e $I_x^n$, $I_y^n$ son intensidades que son determinadas por la diferencia entre las intensidades del siguiente píxel y el anterior. La norma del gradiente tiene como objetivo mejorar la estabilidad ($|\nabla I^n(i,j)|$) y para calcular la norma del gradiente se usa la ecuación 1.9.

$$\begin{aligned} |\nabla I^n(i,j)| = \begin{cases} \sqrt{(I_{xbm}^n)^2 + (I_{xfM}^n)^2 + (I_{ybm}^n)^2 + (I_{yfM}^n)^2}, & \text{si } \beta^n > 0, \\ \sqrt{(I_{xbM}^n)^2 + (I_{xfm}^n)^2 + (I_{ybM}^n)^2 + (I_{yfm}^n)^2}, & \text{si } \beta^n < 0, \end{cases} \\ \beta^n(i,j) = \overrightarrow{\delta L^n}(i,j).\frac{\vec{N}(i,j,n)}{\left|\vec{N}(i,j,n)\right|} \end{aligned} \tag{1.9}$$

Los índices $b$ y $f$ especifican la diferencia entre las intensidades del píxel actual y el que está en la dirección inversa o la siguiente, en los ejes de coordenadas $x$ o $y$.

Mientras que los índices m y M es una notación que define, si se selecciona el valor máximo que tome (M) o si se selecciona el valor mínimo que tome (m) .

El método propuesto por Bertalmio, entrelaza una serie de pasos $A$ con medidas de difusión anisotrópica $B$, donde $A$, $B$ y $T$ (número total de iteraciones) son parámetros de entrada. El cual utiliza la difusión anisotrópica propuesto por Perona y Malik [20], ellos presentan una función de limitación del proceso de difusión a regiones homogéneas, como se puede ver en la ecuación 1.10.

$$I_{i,j}^{t+1} = I_{i,j}^{t} + \lambda \left[ c_N . \nabla_N I + c_S . \nabla_S I + c_E . \nabla_E I + c_W . \nabla_W I \right]_{i,j}^{t}, \tag{1.10}$$

Donde $I_{i,j}^{t+1}$ es la intensidad de un píxel en las coordenadas (i,j) en el momento t+1, $\lambda$ es un valor constante en el rango de [0, 0.25] para dar estabilidad a el algoritmo, y $\nabla_N I, \nabla_S I, \nabla_E I, \nabla_W I$ representa la diferencia de intensidades de los píxeles con respecto a la dirección que indica su índice (Norte, Sur, Este u Oeste) y el píxel actual, como se puede ver en la ecuación 1.11.

$$\begin{aligned} \nabla_N I_{i,j} &\equiv I_{i,j-1} - Ii, j, \\ \nabla_S I_{i,j} &\equiv I_{i,j+1} - Ii, j, \\ \nabla_E I_{i,j} &\equiv I_{i+1,j} - Ii, j, \\ \nabla_W I_{i,j} &\equiv I_{i-1,j} - Ii, j \end{aligned} \tag{1.11}$$

En el cual $c_N, c_S, c_E, c_W$ son llamados los coeficientes de conducción, estos valores son determinados por el cálculo del gradiente ($\nabla I$).

Existen varios métodos para calcular estos valores, pero los métodos en los que se basa Bertalmio son los dos siguientes:

$$c(\|\nabla I\|) = e^{-(\|\nabla I\|/K)^2}, \tag{1.12}$$

$$c(\|\nabla I\|) = \frac{1}{1 + (\|\nabla I\| / K)^2}, \tag{1.13}$$

Los coeficientes son determinados utilizando los ecuaciones 1.13 o 1.12, donde el gradiente ($\nabla I$) corresponde a la dirección descrita por el índice $K$, con este valor se controla el proceso de detección de bordes.

Tanto el método *inpainting* y la difusión anisotrópica se aplica a los componentes RGB de cada píxel [14].

## 3.3. M. Oliveira (2001)

Basado en el método de Bertalmio y Sapiro en el 2001, Oliveira propuso un método rápido para la técnica *Inpainting*, en este nuevo método se realiza un procesamiento previo a la imagen, así eliminando información como el color dentro de la máscara y afianzando los bordes de la misma [15].

A partir de los píxeles en el borde del área $\Omega$, es decir en el límite $\delta\Omega$, se aplica una operación de convolución, utilizando un kernel que tiene un peso cero en el centro del núcleo (ver Figura 1.22). Los aspectos positivos son que el algoritmo es muy rápido y funciona bien para las imágenes que no tienen muchos bordes de alto contraste o componentes de alta frecuencia.

| | | |
|---|---|---|
| 0.073235 | 0.176765 | 0.073235 |
| 0.176765 | 0 | 0.176765 |
| 0.073235 | 0.176765 | 0.073235 |

| | | |
|---|---|---|
| 0.125 | 0.125 | 0.125 |
| 0.125 | 0 | 0.125 |
| 0.125 | 0.125 | 0.125 |

Figura 1.22: Los dos kernel de difusión usados por Oliveira.

Finalmente el algoritmo produce un resultado en pocos segundos después de más de 100 iteraciones, siendo así para el 2001 la técnica más rápida de *Inpainting*, pero que produce desenfoque en la imagen.

## 3.4. Criminisi (2004)

El algoritmo de Criminisi tiene como objetivo lograr la síntesis de textura [21], teniendo en cuenta la información estructural de la imagen, tales como las líneas que cruzan el borde de la zona ocluida $\epsilon$.

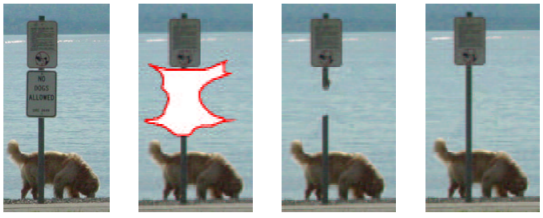

Figura 1.23: Resultado de aplicar el algoritmo de Criminisi.

Este algoritmo está compuesto por tres pasos y se comienza trabajando con los píxeles $\delta\Omega$ que se encuentran en el borde de la máscara $\Omega$. Primero se calcula, para todas las ventanas centradas en píxeles del borde una prioridad $P$, donde esta representa el píxel procesado en un momento determinado.

$$P(p) = C(p) * D(p), \tag{1.14}$$

Donde $P(p)$ es el píxel procesado, $C(p)$ representa un término de confianza asociado con un bloque $\psi p$ (cuanto mayor sea el número de píxeles conocidos en la ventana, mayor será la confianza), $D(p)$ es un término que procesa la información estructural contenida en la ventana $\psi p$ y el cual eleva la prioridad de un bloque. Estos dos términos se definen como se muestra en la ecuación 1.15.

$$\begin{aligned} C(p) &= \frac{\sum_{q \in \psi_p \cap \Omega} C(q)}{|\psi_p|}, \\ D(p) &= \frac{\left|\nabla I_p^{\perp} n_p\right|}{\alpha} \end{aligned} \tag{1.15}$$

Donde $|\psi_p|$ es la superficie de la ventana $\psi_p$ centrada en $p$ (píxel perteneciente a $\delta\Omega$), en el cual $\alpha$ es un factor de normalización con el valor 255, $n_p$ es el vector normal del contorno $\delta\Omega$ en el punto $p$ y $\nabla I_p^{\perp}$ es el gradiente de la normal.

Para las prioridades $P(p)$ se requiere una etapa de inicialización, en la cual todos los píxeles pertenecientes a la máscara seleccionan el valor de confianza $C(p) = 0$ y los restantes seleccionan el valor de confianza $C(p) = 1$. La siguiente etapa es la del método $Inpainting$, en esta etapa el píxel que tiene la prioridad más alta es el primero a ser procesado y la prioridad se escoge mediante la distancia mínima SSD:

$$\psi_{\bar{q}} = \arg_{\psi_{q \in \phi}} mind(\psi_{\bar{p}}, \psi_q), \tag{1.16}$$

Donde $d(\psi_{\bar{p}}, \psi_q)$ representa el valor SSD (entre todos los píxeles conocidas de la ventana $\psi_p$ y los que están en las posiciones correspondientes en un bloque $\psi_q$, pertenecientes a la banda de la fuente) [21]. El último paso consiste en actualizar los valores de confianza asociados a los píxeles de la ventana restaurada, con la fórmula:

$$C(p) = C(\bar{q}), \forall p \in \psi_{\bar{p}} \cap \Omega. \tag{1.17}$$

## 3.5. Wexler (2007)

Wexler en su artículo *"Space-time video completion"* [22] presenta el algoritmo que desarrollo para la variedad de tareas en la post-producción o restauración de un vídeo. En el cual se hacen los mismos pasos que en los trabajos anteriores, es decir se selecciona el área a la cual se aplica el algoritmo *Inpainting*.

El corazón de su algoritmo es una medida de similitud entre las distintas imágenes del vídeo en un espacio tiempo específico. La suma de las diferencias al cuadrado (SSD), que es tan ampliamente utilizado en los distintos algoritmos de *Inpainting* para completar la imagen, pero este método no es suficiente para obtener los resultados deseados en el vídeo (independientemente de la elección del espacio de color). La razón principal de esto es que el ojo humano es muy sensible al movimiento. Mantener la continuidad de movimiento es más importante que encontrar la combinación exacta dentro de una imagen del vídeo.

Cuyo algoritmo tiene como objetivo, poder crear una medida de similitud porcentual. Por lo tanto se añade una medida que es similar a la de flujo normal para obtener una aproximación rápida de la información de movimiento. Sea $Y$ la secuencia que contiene la intensidad en escala de grises (información obtenida a partir de la secuencia de colores). En cada punto del espacio tiempo, se calcula las derivadas parciales y temporales $(Yx, Yy, Yt)$. Si la moción fuera sólo horizontal, entonces $u = Yt/Yx$ capturaría el movimiento instantáneo en la dirección $x$. Si la moción fuera sólo vertical, entonces $v = Yt/Yy$ sería capturar el movimiento instantáneo en la dirección y.
Las fracciones factorizan la mayor parte de la dependencia entre el territorio y los cambios temporales, mientras que la captura de objetos, velocidades y direcciones se escalan y se añaden a las mediciones RGB para obtener una representación en 5 dimensiones (5-D), para cada punto del espacio tiempo: $(R, G, B, \alpha u, \alpha v)$ .

Teniendo en cuenta que Wexler en su algoritmo no calcula el flujo óptico, si no que el aplica norma L2 (SSD) o también conocida como norma euclidiana, a esta representación 5-D, con el fin de capturar similitudes de espacio tiempo para las piezas estáticas y dinámicas simultáneamente. Es decir, para dos ventanas de espacio tiempo $W_p$ y $V_q$ como se puede ver en la ecuación 1.18.

$$d(W_p, V_q) = \sum_{x,y,t} \|W_p(x, y, t) - V_q(xyt)\|^2 \tag{1.18}$$

Donde para cada $(x, y, t)$ dentro de $W_p(x, y, t)$ denota su vector de medición 5D. La distancia se traduce a la medida de similitud como se puede ver en la ecuación 1.19.

$$sim(W_p, V_q) = e^{-\frac{d(W_p, V_q)}{2\sigma^2}} \tag{1.19}$$

La elección de $\sigma$ es importante, ya que este valor controla la suavidad de la superficie de error inducido. En lugar de utilizar un valor fijo, se elige que sea el 75 percentil de todas las distancias en la búsqueda actual. De esta manera, la mayoría de los lugares se tienen en

cuenta y por tanto, hay una alta probabilidad de que el error global se reduzca [22].

## 3.6. Barnes (2009)

Barnes desarrollo un algoritmo cuya idea principal es que algunos buenos valores a sustituir se pueden encontrar a través de un muestreo aleatorio, y que la coherencia natural en las imágenes permita propagar tales partidos rápidamente a las zonas circundantes [23].

Por ello el núcleo principal de este algoritmo es el cálculo de la correspondencia de conexión, es decir el define un campo de vecinos más cercanos (NNF) como una función $f : A \mapsto R^2$, definidas sobre todas las posibles coordenadas a sustituir en la imagen A, para alguna función distancia de dos puntos de $\delta\Omega$.

Dado el valor a sustituir $a$ en la imagen $A$ y el vecino correspondiente más cercano $b$ en la imagen $B$, es decir que $f(a)$ es simplemente $b - a$.

Se refiere a los valores de $f$ como los valores deseados, y se almacenan en una matriz cuyas dimensiones son las de $A$. Por lo cual Barnes presenta un algoritmo aleatorio para calcular un valor aproximado de NNF.

Este algoritmo tiene tres componentes principales, las cuales son: inicialización (se genera la imagen $B$ utilizando valores aleatorias de la imagen original), propagación (se revisa la correspondencia de la imagen $A$ con la imagen $B$, y se verifica si la imagen $A$ mejora con la correspondencia en $B$) y búsqueda (se realiza búsquedas al azar en la imagen $B$ para sustituir un punto en la imagen original), el campo vecino más cercano está lleno de cualquiera de los valores aleatorios o alguna información previa.

A continuación, un proceso de actualización iterativo se aplica a la NNF, donde los valores con buenos resultados se propagan a los píxeles adyacentes, seguido se aplica una búsqueda al azar en la vecindad de los mejores valores encontrado hasta ahora.

Después de la inicialización, se lleva a cabo un proceso iterativo de mejora de la NNF. Cada iteración del algoritmo procede de la siguiente manera: Los resultados se examinan en orden de exploración (de izquierda a derecha y de arriba a abajo), y cada uno se somete primero a la técnica de propagación y luego a la búsqueda aleatoria. Estas operaciones se intercalan en el nivel de parche: si $P_j$ y $S_j$ denotan, respectivamente propagación y búsqueda aleatoria en el parche $j$, a continuación, se procede en el orden: $P_1, S_1, P_2, S_2, \ldots, P_n, S_n$.

Cuando se realiza el método de propagación lo que se intenta mejorar es $f(x, y)$ con el resultado conocido de $f(x-1, y)$ y $f(x, y-1)$, asumiendo que los resultados sean los mejores. Utilizando la función $D(v)$ como la función encargada de minimizar el error resultante entre

el resultado $(x, y)$ en $A$ y el resultado $(x, y) + v$ en $B$; con ello tomamos nuestro nuevo valor para $f(x, y)$ como el $\arg min$ de:

$$\{D(f(x, y)), D(f(x - 1, y)), D(f(x, y - 1))\} \tag{1.20}$$

El efecto resultante es si $(x, y)$ tiene una asignación correcta y está en una región coherente de $R$, entonces los valores de abajo y a la derecha de $(x, y)$ se llena con la asignación correcta. Por otra parte, en algunas iteraciones se propaga la información en sentido contrario utilizando las funciones inversas ($f(x + 1, y)$ y $f(x, y + 1)$ ) como nuestros desplazamientos candidatos.

Con la búsqueda aleatoria se busca mejorar el valor de $f(x, y)$ probando una serie de resultados candidatos que nos consiga mejorar de manera exponencial la distancia de $f(x, y)$ utilizando la ecuación 1.21.

$$u_i = f(x, y) + w\alpha^i R_i \tag{1.21}$$

Donde $R_i$ es un valor aleatorio uniforme en $[-1, 1]$ x $[-1, 1]$, $w$ es el valor máximo de búsqueda, y $\alpha$ es una relación fija entre los tamaños de ventana de búsqueda.

Selecciona todos los resultados para $i = 0, 1, 2 \ldots$ hasta que la corriente $w\alpha^i$ este por debajo de 1 píxel. La ventana de búsqueda debe sujetarse a los límites de la imagen $B$.

## 3.7. M. Hadhoud, A. Moustafa y Z. Shenoda (2009)

En el 2009 Hadhoud y su equipo proponen una mejora del método de Oliveira, el cual principalmente afecta al resultado final y el tiempo de procesamiento, ya que ellos proponen seguir con casi todos los mismos pasos de Oliveira solo que cambian el kernel de convolución, con la idea de utilizar lo máximo posible de la información contenida en la región $\epsilon$ (Figura 1.15), teniendo en cuenta el proceso de restauración (Figura 1.24). Mediante el uso de los vecinos más conocidos, la restauración se puede lograr incluso dentro de una única iteración [24].

| | | | | | |
|---|---|---|---|---|---|
| 0.073235 | 0.176765 | 0.073235 | 0.125 | 0.125 | 0.125 |
| 0.176765 | 0.073235 | 0.176765 | 0.125 | 0.125 | 0.125 |
| 0.073235 | 0.176765 | 0 | 0.125 | 0.125 | 0 |

Figura 1.24: Los dos kernel de difusión usados por Hadhoud.

Ellos deciden cambiar el Kernel de convolución ya que con la técnica de Oliveira algunos de los píxeles son desconocidos, ya que el borde $\delta\Omega$ se encuentra en el centro del kernel de convolución, conteniendo así en el kernel píxeles dentro de $\Omega$ como se muestra en la Figura 1.25-a. Debido a esta razón Oliveira itera la convolución a más de 100 veces para así difuminar los colores hacia las áreas distantes, para así poder aproximarse al valor deseado.

Por ello Hadhoud coloca el núcleo de difusión en la esquina inferior derecha en lugar del centro, lo cual hace que no sea necesario de realizar tantas iteraciones ya que el área que no es conocida es inferior, como se puede percibir en la Figura 1.25-b.

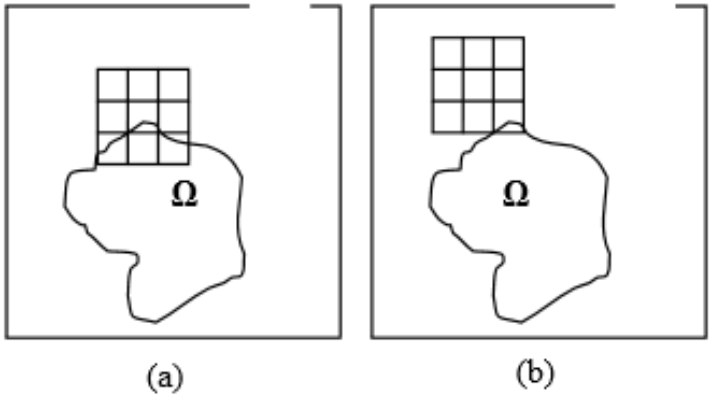


Figura 1.25: a) Propuesta de kernel de convolución segun Oliveira b) Propuesta de kernel de convolución segun Hadhoud.

## 3.8. Darabi (2012)

Darabi con su algoritmo desea sustituir el área $\Omega$, con el entorno $\delta\Omega$, en su algoritmo tiene en cuenta que las regiones de la imagen de entrada pueden ser inconsistente debido a diferentes tipos de iluminación o transformaciones geométricas.

Por ello él se plantea esta tarea como un problema y trata de optimizarlo con la siguiente función:

$$E(\Omega, \delta\Omega) = \sum_{q \subset \Omega} \min_{p \subset S} (D(Q, P) + \lambda D(\nabla Q, \nabla P)), \tag{1.22}$$

Donde $Q = N(q)$ es un kernel con el píxel objetivo $q$ en la esquina superior izquierda, y $P = f(N(p))$ es un kernel con el resultado de una transformación geométrica y fotométrica $f$ aplicada sobre una pequeña vecindad $N$ alrededor del píxel de origen $p$. Los cuales tiene

cinco canales, tres canales de color $L * a * b$ y dos canales del gradiente de luminancia de cada píxel $(L, a, b, \nabla_x Ly\nabla_y L)$.

Sin embargo, para simplificar la notación, se va a utilizar $P$ o $Q$ para denotar los tres canales de color, y $\nabla P$ o $\nabla Q$ para denotar los dos canales del gradiente de luminancia. D es la suma de distancias al cuadrado (SSD) sobre todos los canales, y las dimensiones del gradiente se ponderan por $\lambda w, r, t$.

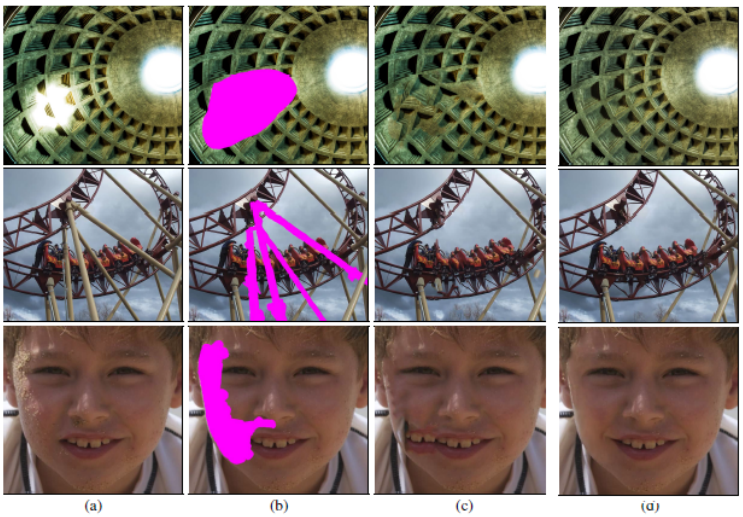


Figura 1.26: a) Es la imagen original, b) Es la imagen original con el área deseada a hacer *Inpainting*, c) Es la imagen resultante al aplicar el algoritmo de Wexler [22] y d) Es la imagen resultante al aplicar el algoritmo de Darabi [25].

Esta función de energía define el relleno óptimo en el que cada vecindario local es similar a algún valor local de la vecindad dentro de la fuente, bajo un conjunto restringido de transformaciones.

Darabi limita estos ajustes dentro de unos rangos predefinidos razonables. Estos se calculan de la siguiente manera, donde $c$ es el canal de color:

$$\begin{aligned} g(P^c) &= min\{max\{\sigma(Q^c)/\sigma(P^c), g_{min}\}, g_{max}\}, \\ b(P^c) &= min\{max\{\mu(Q^c) - g(P^c)\mu(P^c), b_{min}\}, b_{max}\}, \end{aligned} \tag{1.23}$$

Donde $c \in L, a, b, \sigma()$ y $\mu()$ es la desviación estándar y la media de entrada en cada canal c, y $[g_min, g_max]$ y $[b_min, b_max]$ son los rangos de ganancia y polarización.

Esta ganancia y el sesgo se utilizan para ajustar los colores [25]:

$$P^c : P^c \leftarrow g(P^c)P^c + b(P^c) \tag{1.24}$$

## 3.9. He y Sun (2012)

He y Sun aplican un algoritmo de vecino más cercano con una restricción, el cual es que antes de calcular la diferencia entre un par de parches primero se debe revisar su distancia espacial y rechazarlas si la restricción se desobedece. Ellos llevan a cabo este paso coincidente en una región rectangular en torno a la caja de contorno del agujero. Este rectángulo es 3 veces más grande (de longitud) que el cuadro de límite. El propósito de usar tal región rectangular es evitar estadísticas poco fiables si el agujero es demasiado pequeño.

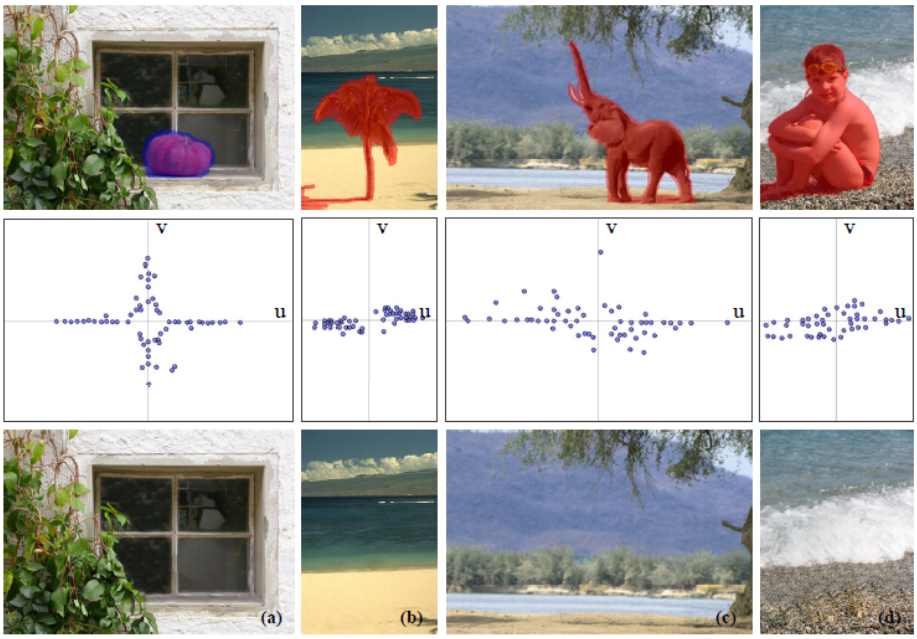


Figura 1.27: Resultado de aplicar el algoritmo de He y Sun. [26].

Dado el campo vecino más cercano $s(x)$, se calcula el histograma $h(u, v)$ y este histograma se suaviza con un filtro de Gauss ($\sigma = \sqrt{2}$). En este histograma suavizado, se considera un pico como un valor cuya magnitud es localmente máxima dentro de una ventana 9 x 9 píxeles. Los picos más altos de $K$ son recogidos, y sus correspondientes desplazamientos dan los candidatos de compensación $\{SI\}K_{i=1}$, los cuales se utiliza en los algoritmos de terminación de imagen.

Luego implementan lo que hizo Darabi, pero con la diferencia de que ellos ven cuales son los valores más parecidos mediante el histograma, con ello realizan una interpolación de los vecinos más cercano, He y Sun para culminar con su algoritmo aplican la función de Poisson sobre los resultados obtenidos para así ocultar los posibles cambios de color que genera el algoritmo [26].

## 3.10. Huang (2014)

Huang presenta un algoritmo de restructuración de una imagen, el cual extrae estructuras de escenas nivel medio para así poder guiar la finalización de bajo nivel, lo cual permite detectar múltiples planos y su correspondiente regularidad de traslación. El resultado del algoritmo supera en la mayoría de los casos a muchos de los algoritmos actuales, ya que con este algoritmo se consigue un mejor resultado en escenas complejas.

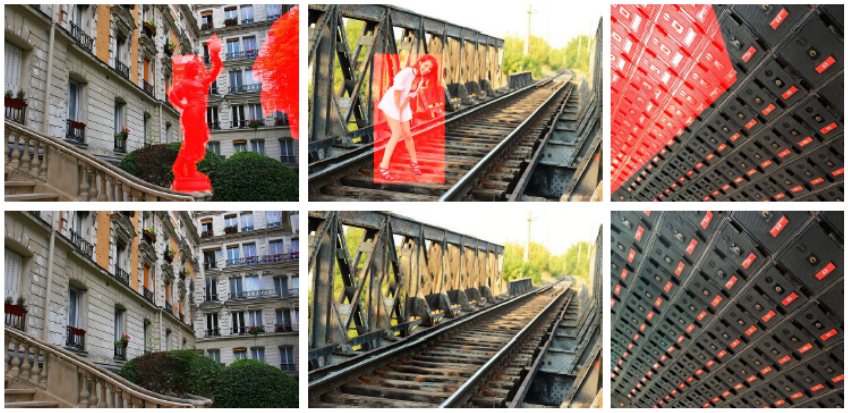

Figura 1.28: Resultado de aplicar el algoritmo de Huang. [27].

Este objetivo se logra gracias al análisis de la región conocida de la imagen ($\delta\Omega$) para detectar superficies planas y la regularidad de traslación. El análisis de estos valores es usado para restringir los valores a utilizar en la imagen final. Teniendo en cuenta que los parches de destino son imágenes alineadas sin transformación geométrica, es decir como una escala o rotación, mientras que los parches de origen tienen una transformación geométrica que se deriva implícitamente de la geometría del plano.

Huang utiliza el algoritmo de muestreo para una ubicación aleatoria de Barnes, para incorporar las probabilidades calculadas y la regularidad de traslación. Además de la ubicación aleatoria de muestreo regular, Huang en su implementación, utiliza 5 iteraciones de plano probabilidad y la regularidad, ambos guiados por el muestreo obtenido en la etapa de búsqueda y propagación.

Este método demuestra el beneficio de realizar un análisis previo a la imagen, es decir que con este artículo se demuestra que la calidad de la imagen se puede mejorar de manera significativa si se logra un equilibrio entre el análisis y la síntesis [27].

# 4. Introducción WebCL

En esta sección, se explicará todo lo referente a WebCL y el estado actual del mismo.

## 4.1. Conceptos básicos

WebCL, es una tecnología que permite la integración de OpenCL, mediante el uso del lenguaje JavaScript, con el cual, se puede usar la computación paralela heterogénea.

WebCL o Web Computing Language se ha desarrollado en estrecha cooperación con la comunidad web, y ofrece la posibilidad de ampliar las capacidades de los navegadores que utilizan HTML5; gracias a esta tecnología se puede acelerar las aplicaciones que requieran de un alto costo computacional.

Permite a las aplicaciones web poder aprovechar la GPU y multi-core CPU, con el objetivo de poder realizar procesamiento paralelo desde un navegador web, esto permite una aceleración significativa de las distintas aplicaciones [28]; como por ejemplo al momento de realizar procesamiento de imagen o también para poder utilizar física avanzada en juegos WebGL, en la Figura 1.29 se muestra una comparación entre una solución JavaScript y WebCL.

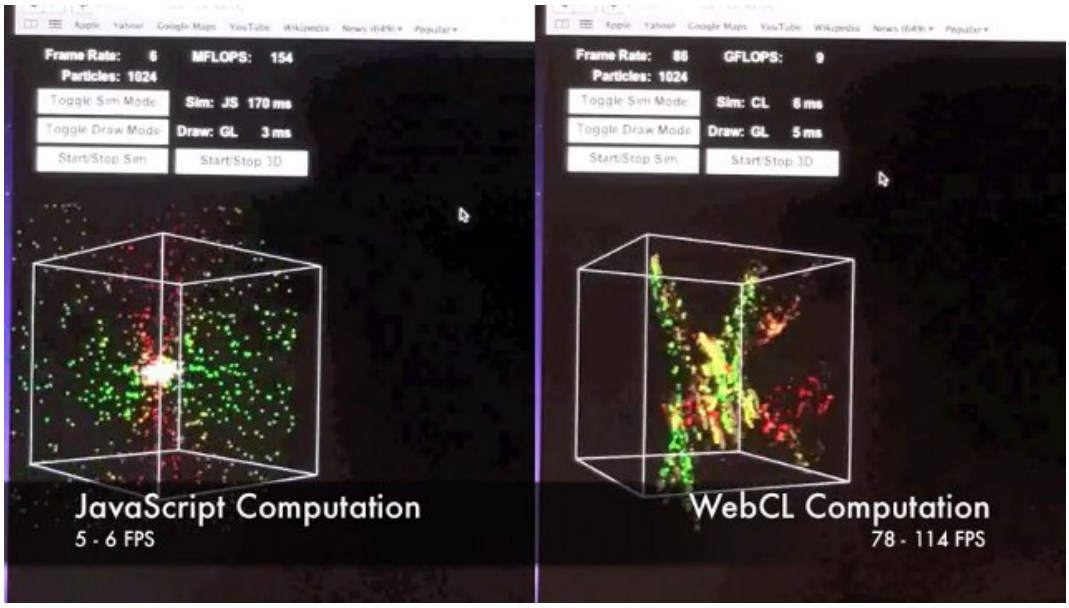


Figura 1.29: Demostración de partículas en el cálculo de la interacción física entre 1.024 elementos individuales utilizando WebCL dio 114 FPS, mientras que la versión tradicional JavaScript tiene 6 FPS.[29]

# 5. Justificación

En este trabajo se pueden ver los distintos avances que ha tenido la técnica *Inpainting*, y cómo a lo largo de los últimos años se consigue restaurar una sola imagen, con nada más que el entorno de la misma.

El estudio de la técnica *Inpainting* es muy llamativa, y hasta los momentos muy extensa, ya que actualmente no existe una solución para todos los casos donde se logre obtener una imagen resultante totalmente correcta, esto se puede evidencia en los distintos trabajos contenidos en este documento (Bertalmio, Wexler, Barnes, Darabi, He y Sun, etc), los cuales dan a entender que se necesita realizar varios tipos de procesamiento a la imagen, para así obtener el resultado esperado en un período de tiempo mínimo, y que se debe realizar una búsqueda en toda la imagen, con el objetivo de obtener las distintas secuencias de píxeles que tengan un mayor parecido al entorno del área $\Omega$.

Dado que no existe una aplicación web en la cual se pueda realizar la técnica *Inpainting*, este trabajo especial de grado tiene como objetivo realizar una aplicación web para la técnica *Inpainting* aprovechando que WebCL te permite acelerar la carga computacional.

# 6. Objetivo general

Desarrollar una aplicación Web que implemente la técnica *Inpainting*, utilizando WebCL.

Para lograrlo la aplicación debe cumplir con los Objetivos Específicos que se explicaran en el siguiente punto.

# 7. Objetivos específicos

- Elaborar una página web para la técnica *Inpainting* usando solo el lado cliente.
- Permitir flexibilidad en los formatos de imágenes permitidos.
- Elaborar una página web en la cual el usuario pueda manejar con total facilidad la interfaz de usuario.

# Capítulo 2

# Marco Aplicativo

En este capítulo se describirá el diseño planteado para nuestra solución, se explicara el desarrollo realizado junto con las herramientas que se utilizaron para nuestra aplicación web.

## 1. Diseño

En esta sección se plantea la solución web que se desea implementar para el algoritmo *Inpainting*, se realiza los Diagramas de Caso de Uso con su respectiva descripción, Modelo Objeto del Dominio y el diagrama de secuencia que seguirá la aplicación web. Todo esto se realiza con el objetivo de describir en detalle el funcionamiento o desenvolvimiento deseado para nuestra aplicación web de *Inpainting*, que emplea WebCL para acelerar el funcionamiento del mismo.

### 1.1. Diagrama de Caso de Uso

En la Figura 2.1 se puede ver el diagrama de caso de uso que se utilizara en nuestra aplicación web. La cual esta compuesta por 5 casos de usos principales: Cargar Imagen; Descargar Imagen; Descargas necesarias; Realizar Inpainting y Efecto extras. Todos estos son accesibles por el usuario al momento de utilizar la aplicación web.

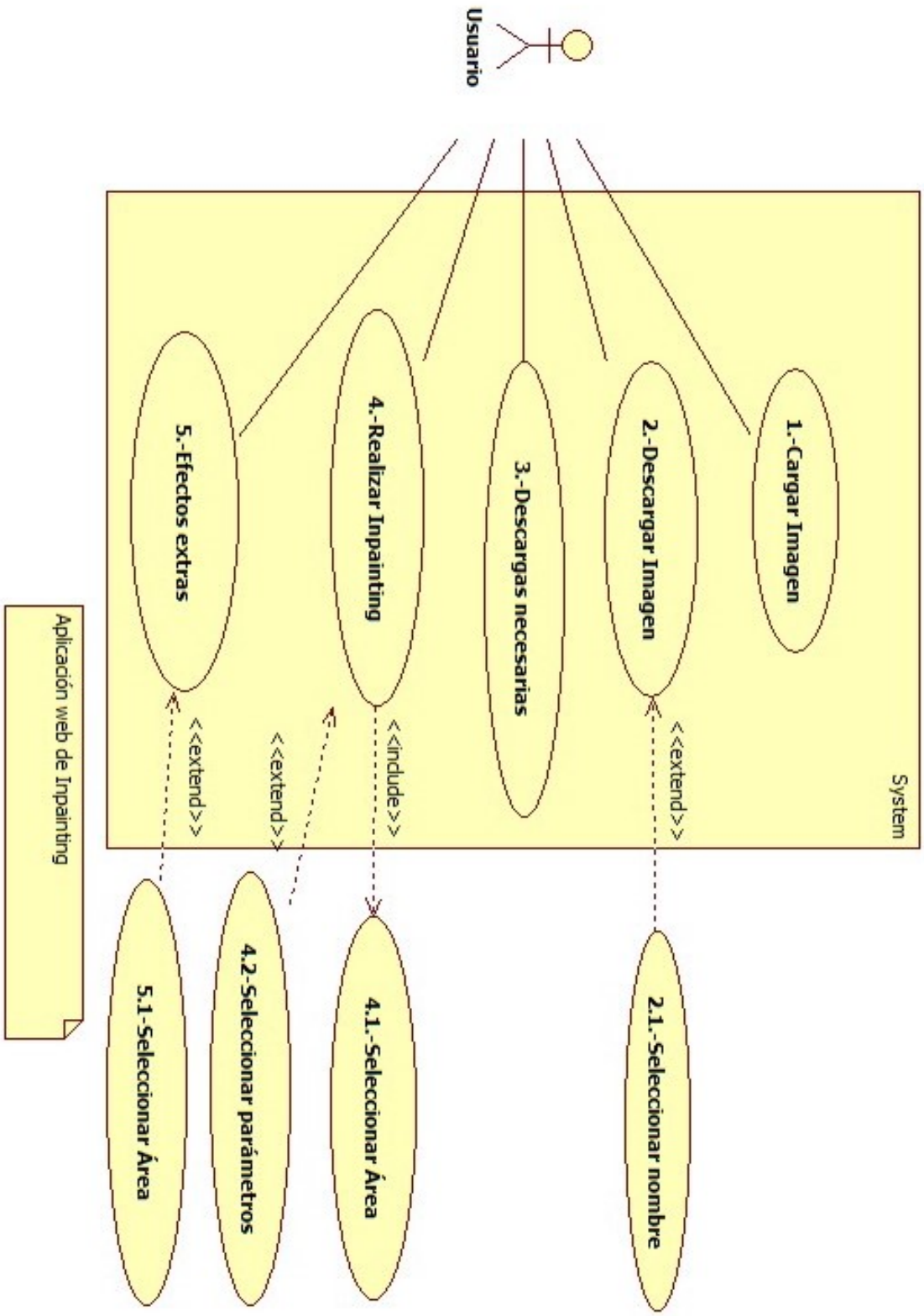


Figura 2.1: Diagrama de Caso de Uso

| Caso de uso | 1.-Cargar Imagen |
|---|---|
| Actor | Usuario |
| Descripción | Permite al usuario desplegar una nueva imagen en el canvas. |
| Pre condiciones | El usuario selecciona la opción Cargar Imagen y esta utilizando el navegador Firefox. |
| Post condiciones | El usuario cargara una nueva imagen en el canvas. |
| Flujo básico | 1.- Selecciona la opción Cargar Imagen.<br>2.- Escoge la opción Seleccionar Imagen.<br>3.- Escoge nueva imagen.<br>4.- Selecciona la opción Ok.<br>5.- Desplegar nueva imagen en el canvas. |
| Flujos alternos | **Cancelar:**<br>1.- Seleccionar la opción Cancel o cerrar la ventana/pestaña del navegador.<br>**No selecciono ninguna imagen:**<br>1.- Si no selecciona ninguna imagen y presiona sobre la opción Ok.<br>2.- Se despliega una alerta de color rojo diciendo que no ha introducido una nueva imagen. |
| Notas | |

En la tabla 1 Cargar Imagen, se puede ver como sería el flujo al cargar una imagen en el canvas.

| Caso de uso | 2.-Descargar Imagen |
|---|---|
| Actor | Usuario |
| Descripción | Permite al usuario descargar a la ruta especificada, la imagen mostrada en el canvas. |
| Pre condiciones | El usuario selecciono la opción Descargar imagen a la PC y esta utilizando el navegador Firefox. |
| Post condiciones | El usuario desea descargar una nueva imagen a su PC. |
| Flujo básico | 1.- Selecciona la opción Descargar Imagen a la PC.<br>2.- Selecciona la opción Descargar.<br>3.- Selecciona la opción salvar archivo y Ok.<br>4.- Así se almacena la imagen en la carpeta Descargas. |
| Flujos alternos | **Cancelar:**<br>1.- Cerrar la ventana/pestaña del navegador. |
| Notas | |

En la tabla 2 Descargar Imagen, se explica con mayor detalle lo que sucede al momento de descargar una imagen a la computadora.

| Caso de uso | 2.1-Seleccionar nombre |
|---|---|
| Actor | Usuario |
| Descripción | Permite al usuario colocar un nombre a la imagen que desea almacenar. |
| Pre condiciones | El usuario escribió un nombre en el campo Nombre: |
| Post condiciones | El usuario define un nombre para la imagen del canvas. |
| Flujo básico | 1.- Selecciona escribir en la opción Nombre:.<br>2.- Escribe el nombre deseado para su imagen.<br>3.- Selecciona la opción Descargar.<br>4.- Selecciona la opción salvar archivo y Ok.<br>5.- Se almacena la imagen en la carpeta Descargas con el nombre seleccionado. |
| Flujos alternos | **Cancelar:**<br>1.- Cerrar la ventana/pestaña del navegador. |
| Notas | |

En la tabla 3 Seleccionar nombre, en este apartado se puede evidenciar como funciona el flujo extend de la Figura 2.1, con esta opción se podrá almacenar la imagen del canvas con el nombre deseado por el usuario.

| Caso de uso | 3.-Descargas necesarias |
|---|---|
| Actor | Usuario |
| Descripción | Permite al usuario descargar los archivos necesarios para poder utilizar WebCL en el navegador. |
| Pre condiciones | El usuario selecciono la opción Descargar WebCL o Versión de Firefox. |
| Post condiciones | El usuario instala y configura su navegador. |
| Flujo básico | 1.- Selecciona la opción Descargar WebCL o Versión de Firefox.<br>2.- Esperar que termine de Descargar.<br>3.- Instalar.<br>4.- Reiniciar navegador. |
| Flujos alternos | |
| Notas | |

En la tabla 4 Descargas necesarias, aquí se explica el flujo que se tiene que realizar para instalar todos los programas pertinentes, que son necesarios para utilizar WebCL en el navegador Firefox ( mediante esta aplicación web se puede descargar el plugin WebCL y la versión de Firefox que lo soporta).

| | |
|---|---|
| Caso de uso | 4.-Realizar *Inpainting* |
| Actor | Usuario |
| Descripción | Permite al usuario aplicar *Inpainting* sobre el área seleccionada. |
| Pre condiciones | Haber seleccionado un área a editar. |
| Post condiciones | El programa aplica la técnica *Inpainting* sobre el área deseada. |
| Flujo básico | 1.- Selecciona la opción Seleccionar Área.<br>2.- Seleccionar área deseada a editar.<br>3.- Seleccionar la opción Realizar *Inpainting*.<br>4.- Vista previa de imagen.<br>5.- Seleccionar los parámetros para su edición.<br>6.- Seleccionar la opción Ok. |
| Flujos alternos | **Cancelar:**<br>1.- Seleccionar la opción Cancel o cerrar la ventana/pestaña del navegador. |
| Notas | |

En la tabla 5 Realizar *Inpainting*, en esta tabla se explica como es el flujo para aplicar la técnica *Inpainting* sobre el área seleccionada, en este apartado se menciona que se tiene que seleccionar unos parámetros, estos serán descritos en mayor detalle en el Capítulo 2 Sección 2.6.

| | |
|---|---|
| Caso de uso | 4.1.-Seleccionar Área |
| Actor | Usuario |
| Descripción | Permite al usuario seleccionar el área donde se desea aplicar *Inpainting*. |
| Pre condiciones | Haber seleccionado la opción Seleccionar Área. |
| Post condiciones | El usuario selecciona el área deseada. |
| Flujo básico | 1.- Escoge la opción Seleccionar Área.<br>2.- Seleccionar área deseada a editar.<br>3.- Escoger la opción realizar *Inpainting*. |
| Flujos alternos | |
| Notas | |

En la tabla 6 Seleccionar Área, se puede ver como sería el flujo al momento de seleccionar el área que se desea aplicar la técnica *Inpainting*.

| Caso de uso | 4.2.-Seleccionar parámetros |
|---|---|
| Actor | Usuario |
| Descripción | Permite al usuario seleccionar los parámetros que se emplea en la técnica *Inpainting* sobre el área seleccionada. |
| Pre condiciones | Haber seleccionado algún parámetro. |
| Post condiciones | El programa aplica la técnica *Inpainting* sobre el área deseada según los parámetros especificados. |
| Flujo básico | 1.- Seleccionar los parámetros con que se editara.<br>2.- Seleccionar la opción Ok. |
| Flujos alternos | **Cancelar:**<br>1.- Seleccionar la opción Cancel o cerrar la ventana/pestaña del navegador. |
| Notas | |

En la tabla 7 Seleccionar parámetros, en este apartado se detalla el flujo sobre los parámetros que delimitaran la técnica *Inpainting* sobre el área deseada, estos serán descritos en mayor detalle el Capítulo 2 Sección 2.6

| Caso de uso | 5.-Efecto extra |
|---|---|
| Actor | Usuario |
| Descripción | Permite al usuario aplicar un Efecto extra en la imagen. |
| Pre condiciones | Haber seleccionado la opción sobre el efecto extra deseado. |
| Post condiciones | El programa aplica el Efecto extra sobre la imagen. |
| Flujo básico | 1.- Selecciona la opción del efecto extra deseado. |
| Flujos alternos | |
| Notas | |

En la tabla 8 Efecto extra, se evidencia el flujo que se realiza al seleccionar alguno de los efectos extras que provee la aplicación web.

| Caso de uso | 5.1.-Seleccionar Área |
|---|---|
| Actor | Usuario |
| Descripción | Este caso de uso permite al usuario seleccionar el área donde se aplica un efecto extra (solo servirá en algunos). |
| Pre condiciones | Haber escogido la opción Seleccionar Área. |
| Post condiciones | El usuario selecciona el área deseada. |
| Flujo básico | 1.- Escoger la opción Seleccionar Área.<br>2.- Seleccionar área deseada a editar.<br>3.- Seleccionar la opción del Efecto extra deseado. |
| Flujos alternos | |
| Notas | |

En la tabla 9 Seleccionar Área, se explica el flujo al momento de seleccionar un área y luego aplicar algún efecto extra. Esta opción solo sera posible en algunos Efectos extra.

## 1.2. Modelo objeto del dominio

En la Figura 2.2 se puede ver el modelo objeto del dominio que utiliza en la aplicación web.

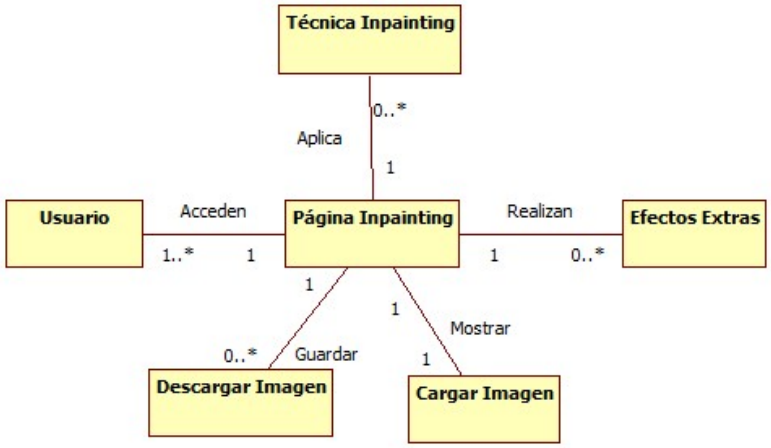


Figura 2.2: Modelo objeto del dominio

La cual esta compuesta por:

- **Usuario**: Objeto que representa el usuario que quiere acceder a la página (*Inpainting*).
- **Página *Inpainting***: Objeto que representa la página web.
- **Efectos Extras**: Objeto que representa los distintos efectos que se puede aplicar a la imagen (exceptuando la técnica *Inpainting* ).
- **Técnica *Inpainting***: Objeto que representa aplicar la técnica *Inpainting* sobre el área seleccionada.
- **Descargar Imagen**: Objeto que representa descargar la imagen a la PC.
- **Cargar Imagen**: Objeto que representa cargar una imagen de la PC al canvas.

## 1.3. Diagrama de secuencia

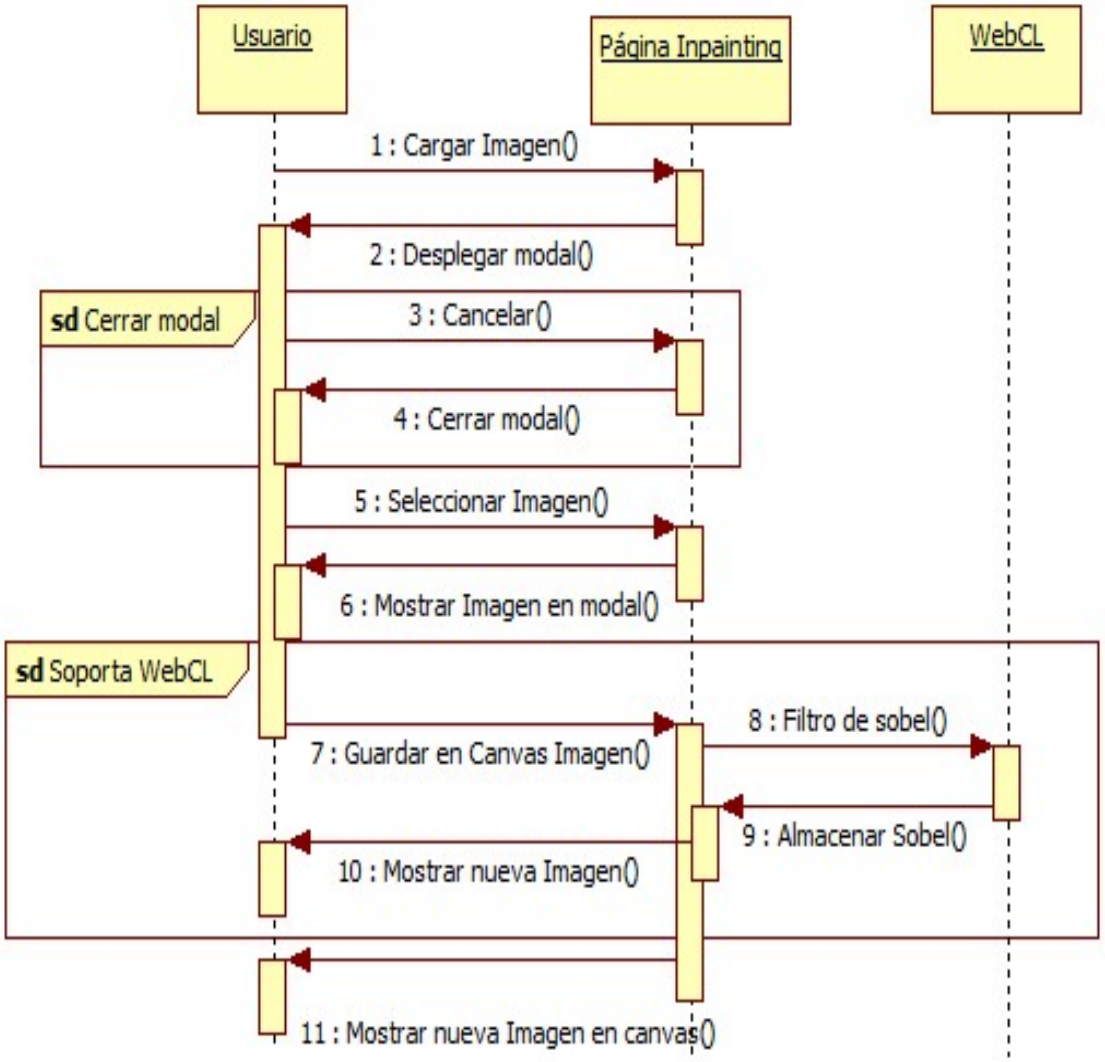


Figura 2.3: Diagrama de secuencia cargar imagen.

En la Figura 2.3 se puede ver el diagrama de secuencia que se implementa al Cargar una nueva imagen en el Canvas.

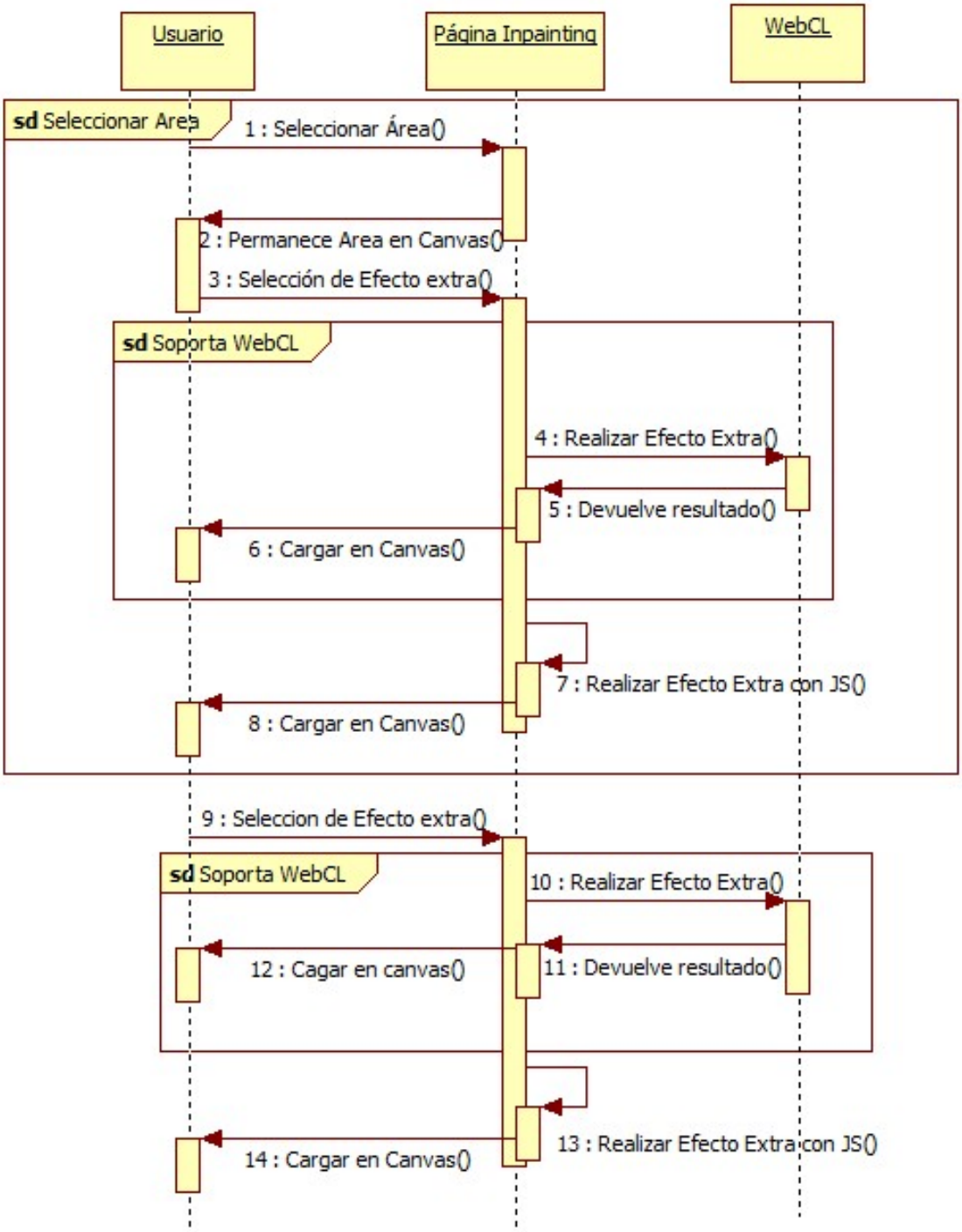


Figura 2.4: Diagrama de secuencia efectos extras.

En la Figura 2.4 se puede ver el diagrama de secuencia que se implementa al aplicar cualquier tipo de Efecto extra, algunos de los efectos extras son: Negativo, Ruido, Mediana, Gradiente de Sobel, etc.

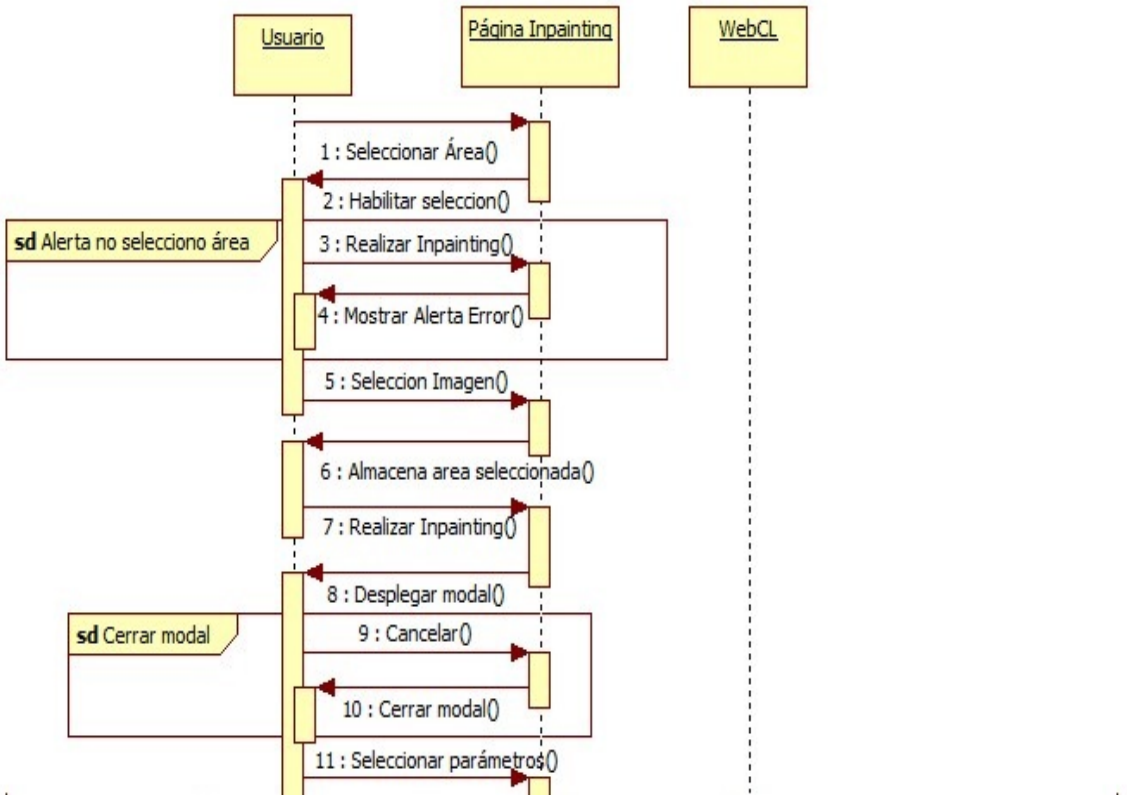


Figura 2.5: Diagrama de secuencia Realizar Inpainting Parte 1.

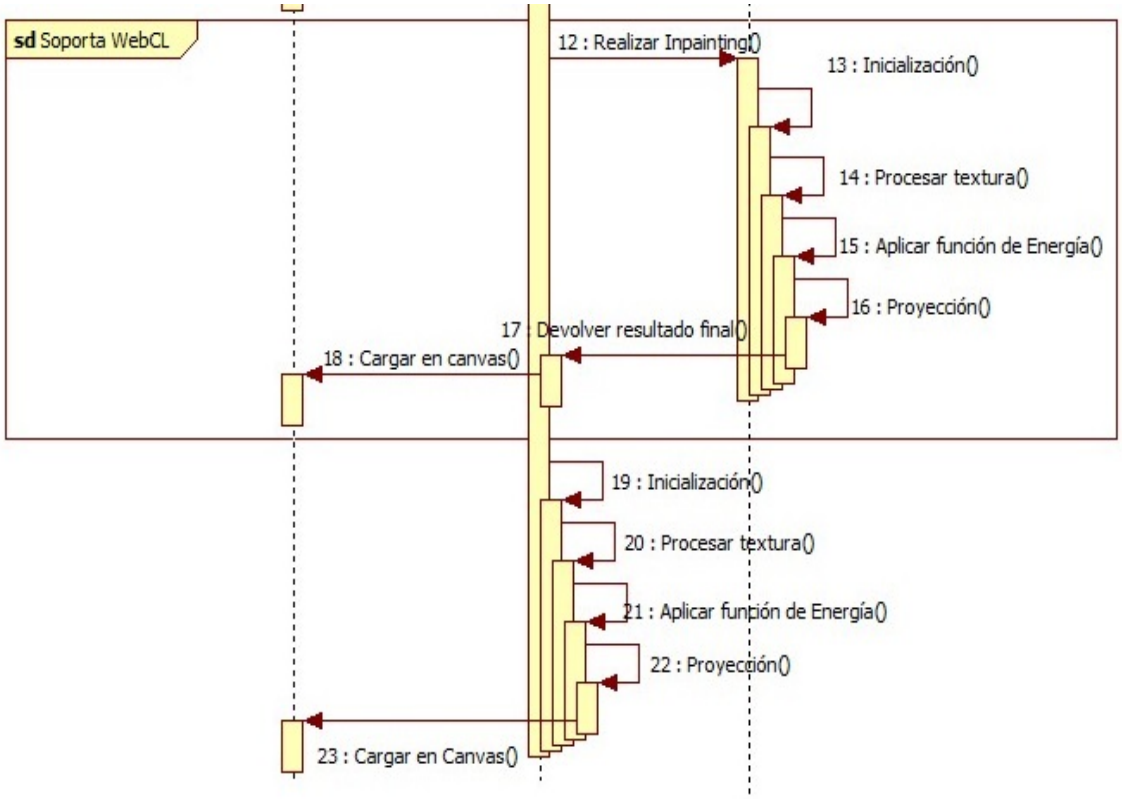


Figura 2.6: Diagrama de secuencia Realizar Inpainting Parte 2.

En la Figura 2.5 y 2.6 se puede ver el diagrama de secuencia que se implementa al Realizar *Inpainting* sobre el área $\Omega$.

# 2. Desarrollo

En esta sección se describe las herramientas a utilizar y la solución final para el algoritmo *Inpainting*, se tiene como entrada dos imágenes, una matriz y una serie de parámetros que son explicados en esta sección.

La primera imagen $P$ de entrada es la destinada a ser procesada, la cual tiene una región desconocida $\Omega$ a ser reconstruida, la segunda imagen es el resultado de aplicar en $P$ el filtro de Sobel, y por ultimo una matriz que contiene la posición de la región desconocida $\Omega$ junto a su entorno.

En la segunda parte del capítulo se describe el enfoque global del algoritmo, donde se resume todas las características del algoritmo.

## 2.1. Descarga y uso de WebCL en una aplicación web

Para usar WebCL en el navegador, lo primero que se debe hacer es descargar el plugin webcl-1.0.xpi el cual se puede descargar desde la aplicación web, también se debe descargar e instalar OpenCL. Si la tarjeta de video es NVIDIA, se debe usar la bibliografía [30] para descargarlo, si es AMD, utilizar la [31], esto permitirá utilizar WebCL en la aplicación web.

La primera prueba que se realiza para saber si su navegador posee todo lo necesario para usar WebCL, es la siguiente:

**Algorithm 1** Primera prueba WebCL

```
1: function WEBCL
2:     if window.webcl != undefined then          ▷ El sistema soporta WebCL
3:         platforms = webcl.getPlatforms(); ▷ Devuelve todas las plataformas disponibles.
4:         ctx = webcl.createContext(WebCL.DEVICE_TYPE_GPU);   ▷ Crea un contexto
5:     else                                       ▷ El sistema no soporta WebCL
6:     end if
7: end function
```

Si el algoritmo anterior da correctamente en el navegador, ya se puede utilizar WebCL sin problema alguno, y por ello ya se puede empezar a trabajar con el.

Por último, la mejor documentación que se encuentra en la bibliografía [32], esa página es la mejor documentación para entender las distintas funciones que provee WebCL y con algunos ejemplos sencillos.

## 2.2. Problemas al momento de utilizar WebCL

Existen varios problemas al momento de utilizar WebCL, el primero y más importante es que la página que tenía el plugin webcl-1.0.xpi [33], desde Noviembre del 2015 se encuentra inhabilitado, por ello se genera un conflicto con el navegador al momento de querer utilizarse, ya que el mismo trata de certificar este plugin, y en vista de que la página no esta en funcionamiento, no logra hacer el certificado.

Otro de los problemas que se pueden encontrar, es el manejo de los hilos por parte de WebCL, donde si se generan muchos hilos con una alta carga computacional, al momento de procesar cada hilo se puede generar alertas, inhibir el navegador o inhabilitar OpenCL de su maquina, lo cual por transitividad hace que se inhabilite el navegador, esperando una respuesta de OpenCL.

**Problemas con Firefox**

Como ya se dijo anteriormente, la página principal donde se encontraba el plugin de Firefox ya no se encuentra en funcionamiento, esto hace que las últimas versiones de Firefox busquen una actualización de este plugin en esa página, y en vista de que no la encuentra inhabilita el plugin en el navegador.

La solución que se encontró a este problema fue instalar una versión más antigua de Firefox, exactamente la 33, con ello no se inhabilita el plugin en el navegador, y WebCL puede ser utilizado con total normalidad.

También, desde nuestra aplicación web se puede descargar el plugin para que funcione con total normalidad en Firefox. Si se genera siguiendo los pasos de la bibliografía [34] el plugin no va a funcionar, y fallaría al momento de generar un contexto WebCL, por eso se recomienda utilizar el que trae esta aplicación web, el cual se descargó cuando [33] aún estaba operativa.

**Problemas con Chromium**

Cuando se buscó la manera de utilizar WebCL en Chromium, se encontró con la siguiente bibliografía [35], se siguió los pasos descritos para instalarlo, pero después de seguirlos no se logró utilizar WebCL en Chromium, por ello esta aplicación no está hecha para ser utilizada en este navegador.

## 2.3. Implementación utilizada de WebCL

Por los problemas descritos anteriormente, y que no se encontró la manera de utilizar WebCL en cualquier otro navegador, esta aplicación web será orientada a una solución enfocada en el navegador web de nombre Firefox en su versión 33, y utilizando el plugin que provee está aplicación (estas condiciones se deben a las limitaciones existentes de WebCL descritas anteriormente).

## 2.4. Herramientas

En esta sección se describe los distintos lenguajes, API, framework y bibliotecas que se utilizo para la solución de *Inpainting* planteada.

### WebGL

Es una API para mostrar gráficos 2D y 3D en el navegador Web, de gran rendimiento computacional y controlando directamente la unidad de procesamiento gráfico del equipo, (GPU). En el cual parte del código se escribe en Javascript y la otra, en un lenguaje similar a C denominado GLSL, (OpenGL Shading Language) [36], en la Figura 2.7 se puede ver un ejemplo de utilizar WebGL.


Figura 2.7: Resultado de utilizar WebGL.

### Bootstrap

Es un framework diseñado para simplificar el proceso de creación de diseños web. Para ello, nos ofrece una serie de plantillas CSS y de archivos JavaScript [37] (como el de la Figura 2.8).


Figura 2.8: Ejemplo de página utilizando Bootstrap.

### OpenCL

OpenCL (Open Computing Language, en español lenguaje de computación abierto) es el primer estándar sin derechos de autor abierto para multiplataforma, el cual mejora en gran medida la velocidad y capacidad de respuesta para una amplia gama de aplicaciones en numerosas categorías de mercado de los juegos y el entretenimiento a software científico y médico.

Es decir que con OpenCL se puede crear aplicaciones con paralelismo a nivel de datos y de tareas, que pueden ejecutarse tanto en unidades centrales de procesamiento como unidades de procesamiento gráfico [38].

### HTML

HyperText Markup Language (HTML) es el lenguaje básico de casi todo el contenido web, por ello la mayoría de las páginas web que visitamos a diario están basadas en este lenguaje [39]. Es considerado un "estándar vivientez técnicamente siempre está bajo construcción.

### JavaScript

Es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Utilizado para construir sitios Web y para hacerlos más interactivos. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM)[40]. Aunque este comparte muchas de las características y de las estructuras del lenguaje Java, fue desarrollado independientemente del mismo; en la Figura 2.9 se puede ver un ejemplo de código JavaScript.

```
<HTML>
<TITLE>Ejemplo03.htm</TITLE>

<SCRIPT LANGUAGE="JavaScript">
//Recoger un dato por teclado y visualizarlo
var nom;
nom=prompt("Escribe tu nombre","NOMBRE");
alert("Mucho gusto "+ nom);

</SCRIPT>

</HTML>
```

Figura 2.9: Ejemplo de código JavaScript, que muestra un alert con el nombre introducido por el usuario.

### jQuery

Jquery es una biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web (un ejemplo de código jQuery se puede ver en la Figura 2.10), lo cual la convierte en la biblioteca más utilizada de JavaScript [41].

```
Encadenamiento:
$('content')
      .find('h3')
      .eq(2)
      .html('nuevo texto para el tercer elemento h3');

Establecer propiedades CSS
$('h1'.css({
      'fontSize' : '100px',
      'color' : 'red'
});
```

Figura 2.10: Ejemplo de código jQuery, en el primer ejemplo se concatena al texto contenido en h3 un nuevo texto y en el segundo ejemplo cambia el CSS establecido de h1 por uno nuevo.

### Random Javascript

Javascript provee un random al aplicar la función Math.Random(), el cual es lo suficientemente bueno para proporcionar una manera fácil de generar animaciones, juegos, etc. Sin embargo, si se quiere usar para una programación más seria, esta función suele seguir un mismo patrón al momento de aplicarse, lo cual termina arrojando casi siempre los mismos resultados.

Por ello se recomienda utilizar otros patrones como:

- **Alea**: Este generador implementa una variación de Marsaglia o también llamado como multiplicación con acarreo, el cual se adaptada a la noción pintoresca de Javascript: los acarreos son exactamente las partes enteras de números, con exactamente 32 bits de parte fraccionaria.
- **LFIB4**: En este generador de Fibonacci el período se encuentra muy cerca de $2^{308}$, y se realiza una combinación de cuatro en lugar de dos, lo cual hace que sea más robusto que los otros generadores de Fibonacci habituales. Pero al realizar combinaciones de 4 se convierte en una solución un poco más lenta que las otras, aunque esta es mucho más rápida en JavaScript que los algoritmos basados en enteros.

- **Xorshift03**: Esta solución se encuentran entre los más rápidos disponibles en la aritmética de enteros, pero se vuelven mucho más lento en Javascript. Además, la mayoría de las variantes tienden a fallar en pruebas de L'Ecuyer que son bastante más estrictas que las de la propia Marsaglia.
- **MRG32k3a**: Esta solucion es notable por su ingenioso uso de la aritmética de enteros de 53 bits (implementado originalmente en C), esto hace que sea muy adecuado para Javascript.
- **Mash**: Al aplicar Mash lo que principalmente se hace es una modificación que depende de la naturaleza del elemento de estado, si se trata de un número entero, se añade simplemente el hash, si es un solo bit, es XOR, si es una fracción entre 0 y 1, se resta el hash y el elemento se incrementa en 1, si este se ha convertido negativo.
- **KISS07**: Esta solución se basa en añadir las salidas de tres pequeños generadores ( Xorshift03, MRG32k3a y Mash) que por separado no tendría la más mínima posibilidad de transmitir una buena prueba de aleatoriedad. Sin embargo, el resultado de estos tres generadores se pasa a BigCrush. Con un período de aproximadamente $2^{121}$ [42].

## 2.5. Enfoque global del algoritmo

La región $\Omega$ es reconstruida de afuera hacia adentro siguiendo un patrón de espiral que empieza en el parche superior izquierdo, como se muestra en la Figura 2.11. El color de cada píxel $p$ en $\Omega$ es reemplazado por el color de algún píxel $q$ del mejor parche obtenido en cada interacción, comparado con su entorno. El cual es seleccionado mediante los parámetros y técnicas que se explican en este capítulo, para así obtener un mejor acabado en el resultado final.

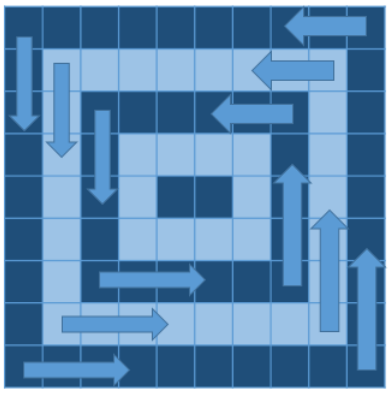

Figura 2.11: Proceso de reconstrucción área $\Omega$. Comienza en el parche superior izquierdo

Las métricas presentadas en este trabajo están aplicadas sobre el espacio L*a*b, L*c*h o L*u*v, ya que estos espacios son mas uniformes perceptiblemente con respecto al espacio RGB y por ello se obtiene una buena estimación de la diferencia entre dos vectores de color (distancia).

En la Figura 2.12, se diagrama la estructura general del algoritmo, esto sirve de guía para ubicar y comprender la funcionalidad de los pasos que serán descritos en el resto del capítulo.

El primer paso de la Figura 2.12 es cargar la imagen en el canvas, al momento de ejecutar el programa se revisara si el navegador (Firefox) soporta WebCL, si lo soporta se aplicara el filtro de Sobel sobre toda la imagen, este resultado se almacenara y luego finalmente se cargara la imagen deseada en el canvas; en caso contrario el programa solo cargara la imagen en el canvas.

Luego en el paso Obtener datos, se obtiene la región a ser reconstruida y sus parámetros (mediante la participación del usuario), para así con estos datos luego poder realizar el procesamiento previo de la solución.

En la selección del área se utiliza el plugin *imgareaselect*, este plugin devuelve las coordenada inicial y final del área rectangular escogida por el usuario, con esta información se genera la matriz de trabajo para esa ejecución. Seguidamente se aplica la etapa de inicialización, donde se almacena los posibles $N$ parches para el área desconocida $\Omega$, los cuales son seleccionados mediante un muestreo aleatorio, para luego aplicar la etapa de propagación en un sentido x o y aleatorio, y por ultimo realizar una búsqueda aleatoria en el rango especificado por el usuario.

Luego se aplica el procesamiento de textura para refinar la zona a tratar, con ello se prepara los mejores parches, en la siguiente etapa del algoritmo (Aplicar función de energía), se aplica el filtro mediana y la técnica Poisson Blending en el área $\Omega$, para luego aplicar la etapa de proyección sobre esta nueva imagen.

Las sub-etapas Propagación y búsqueda aleatoria, son utilizadas en diferentes etapas del algoritmo. Ya que estos son algoritmos que trabajan en conjunto y tienen como objetivo encontrar los mejores candidatos para cada parche correspondiente al área $\Omega$, el cual constituye la parte desconocida de nuestra imagen.

A continuación se explica detalladamente cada etapa, junto con otros aspectos importantes que serán discutidos en el siguiente apartado, los cuales describe en mayor detalle el algoritmo propuesto.

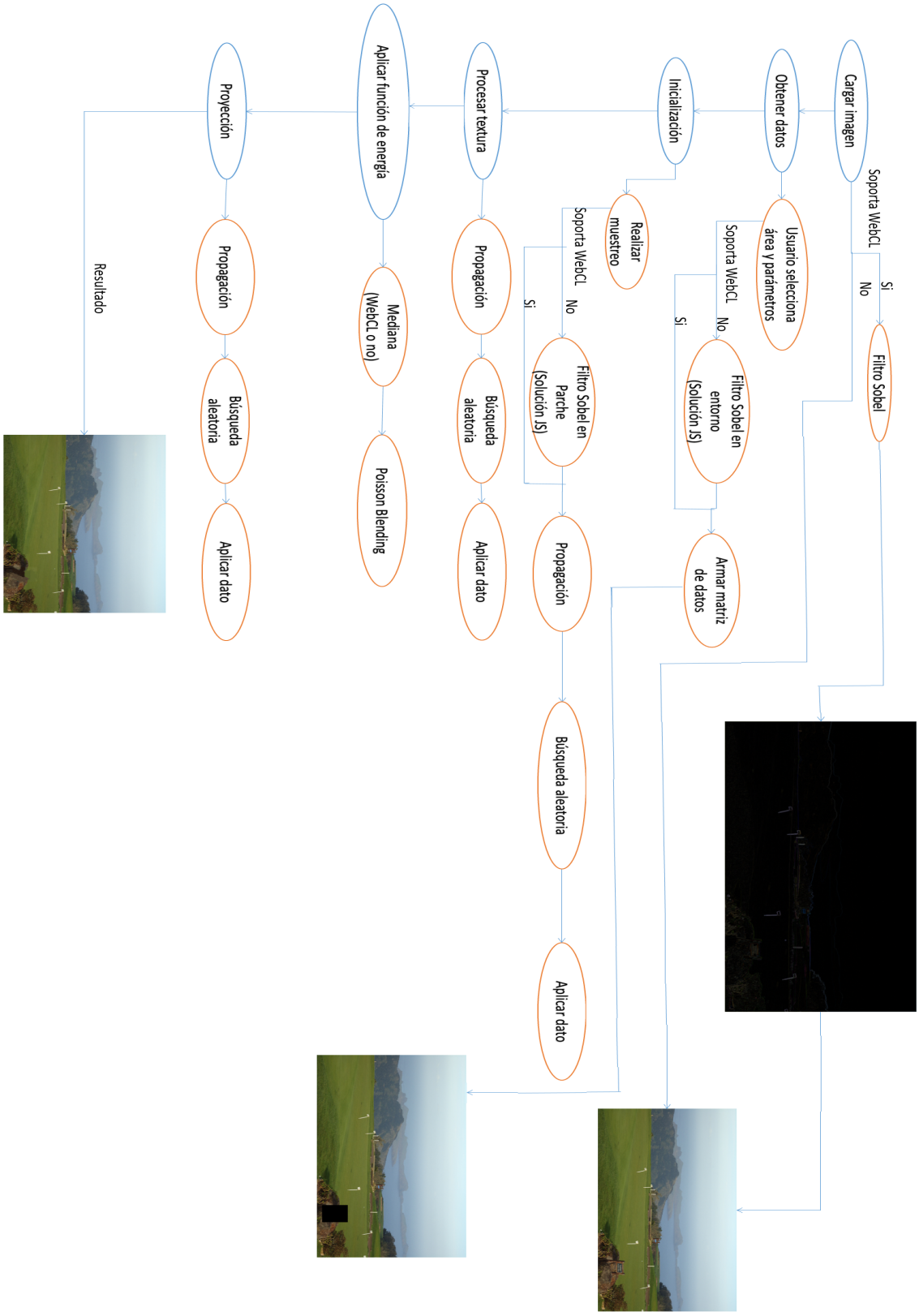


Figura 2.12: Estructura general del algoritmo propuesto

## 2.6. Obtener Datos

Después de haber cargado la imagen a ser manipulada se usa el plugin de Javascript *imgareaselect*, con el cual se podrá seleccionar un área rectangular de la imagen para luego aplicarle *Inpainting*.

Para empezar con este proceso el usuario selecciona la opción *Realizar Inpaiting*, la cual muestra un modal con la vista previa del área seleccionada junto a las distintas opciones para cada uno de los parámetros, como se puede ver en la Figura 2.13.

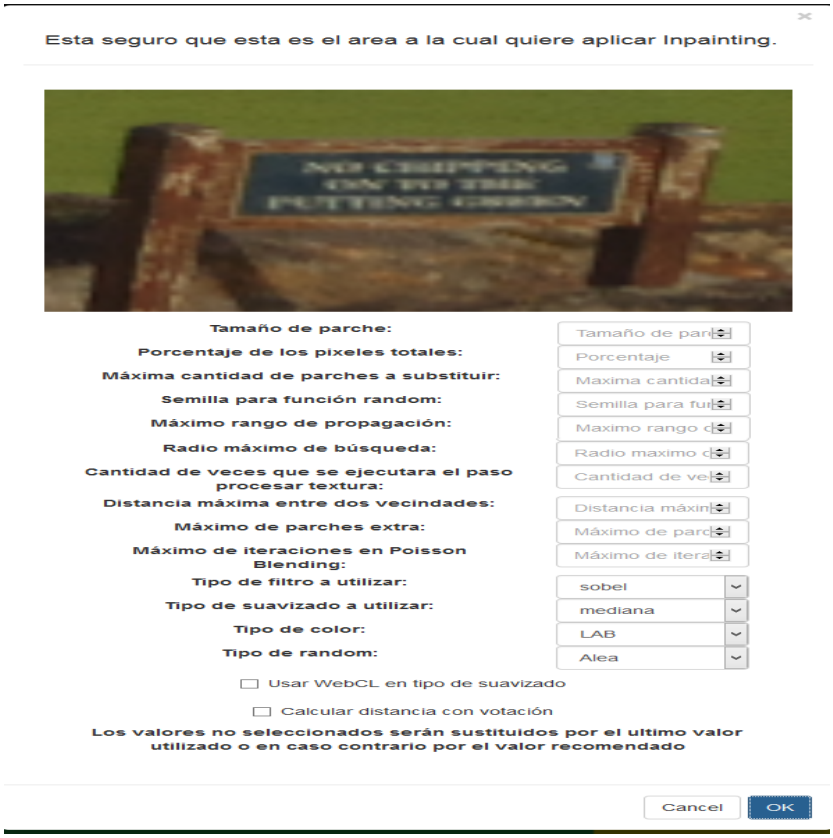


Figura 2.13: Vista previa del modal

Estos parámetros son:

- **Tamaño de parche**: Parámetro que define el tamaño de los parches, este es llamado *TamP*.
  Con este parámetro se sabe cual es la cantidad de píxeles que tiene esta ejecución, por ejemplo si es 2 el tamaño del parche, se tiene 4 píxeles en total.

- **Porcentaje de los píxeles totales**: Parámetro que define la cantidad de parches que utiliza el programa en base a la siguiente formula:

$$(width * height * porctN)/(100 * TamP * TamP) \tag{2.1}$$

  Este parámetro es llamado *porctN*. Por ejemplo si *width* es igual a 1250, *height* es igual a 525, *porctN* es igual a 0,06 y *TamP* es igual a 3. Aplicando la formula 2.1 tiene como resultado el valor: $43{,}75$, este resultado se le aplica la función de JavaScript Math.ceil(), para así obtener que la cantidad de parches que se genera en ésta iteración es de $44$.

- **Máxima cantidad de parches**: Parámetro que define la máxima cantidad de parches que se utiliza en el programa, este parámetro es llamado *maxN*. Con este parámetro se limita la formula anterior, ya que se puede tardar mucho tiempo en generar 44 parches por cada parche a substituir dentro del área $\Omega$.

- **Semilla para función random**: Parámetro que define la semilla que se utiliza en la función random, este parámetro es llamado *seedr*.

- **Máximo rango de propagación**: Parámetro que define la cantidad máxima que se propaga en un sentido x o y, este parámetro es llamado *cProg*. Este parámetro define la cantidad de parches que se toma en una dirección random, como se muestra en la Figura 2.15 y es un rango máximo porque puede ocurrir que la dirección random generada se sale del área de la imagen.

- **Radio máximo de búsqueda aleatoria**: Parámetro que define el rango donde se busca el nuevo parche, este parámetro es llamado *cBus*. Este parámetro define el rango máximo de búsqueda de un parche alrededor del parche a sustituir, como se muestra en la Figura 2.16.

- **Cantidad de veces que se ejecuta el paso procesar textura**: Parámetro que define la cantidad de veces que se ejecuta el paso Procesar Textura, este parámetro es llamado *totpb*.

- **Distancia máxima entre dos vecindades**: Parámetro que define la distancia máxima entre dos vecindades (esta distancia se usa al momento de verificar los parches que son reciclados), este parámetro es llamado *EnerM*. Este parámetro se usa para delimitar los parches que son tomados en cuenta durante todo el algoritmo, es decir que si la distancia total entre las dos vecindades exceden *EnerM* no es tomado en cuenta como parche a procesar y también es utilizado al momento de usar la técnica Poisson Blending.

- **Máximo de parches extra**: Parámetro que limita la cantidad de parches que se reciclan, este parámetro es llamado *MaxBest*. Todos los parches menores a *EnerM* se almacena en un objeto, para luego ser utilizados como nuevo posible parche, con este parámetro se delimita cual es la máxima cantidad de parches que se toma de ese objeto.

- **Máximo de iteraciones en Poisson Blending**: Parámetro que limita la cantidad de iteraciones que tiene Possion Blending en el programa, este parámetro es llamado *poissonmax*.
- **Tipo de filtro a utilizar**(ComboBox): Parámetro que define el tipo de filtro que se utiliza en nuestro programa **Sobel**, **Sobel en X**, **Sobel en Y** y **Laplaciano**.
- **Tipo de suavizado a utilizar**(ComboBox): Parámetro que define el tipo de suavizado que se utiliza en el programa **Mediana** y **Filtro Gaussiano**
- **Tipo de color**(ComboBox): Parámetro que define el tipo de color que se utiliza en el programa **LAB**, **LUV** y **LCH**.
- **Tipo de random**(ComboBox): Parámetro que define el tipo de random que se utiliza en el programa **Alea**, **LFIB4** y **KISS07**.
- **Usar WebCL en tipo de suavizado** (CheckBox): Parámetro que define si la solución que se aplica al suavizado es una con WebCL o no.
- **Calcular distancia con votación** (CheckBox): Parámetro que define si al momento de calcular la distancia entre dos colores se usa una solución por votación o por la sumatoria de todos los parches del entorno.

Después de que el usuario ha seleccionado los distintos parámetros, se calcula los datos alrededor del área $\Omega$ para ser almacenados en la matriz, por ello el área que se almacena en la matriz es desde $[x1 - tamP * 2, y1 - tamP * 2]$ hasta $[x2 + tamP * 2, y2 + tamP * 2]$ siendo $x1$, $y1$ el primer pixel (posición superior izquierda) y $x2$, $y2$ el ultimo píxel ( posición inferior derecha), si el navegador no soporta WebCL, se calcula el filtro Sobel del entorno conocido del área $\Omega$. Con ello se almacena en cada posición de la matriz un arreglo con los siguientes datos:

- **Posición 0**: En este punto del arreglo se almacena la coordenada *y* a la que hace referencia esta posición en la matriz.
- **Posición 1**: En este punto del arreglo se almacena la coordenada *x* a la que hace referencia esta posición en la matriz.
- **Posición 2**: En este punto del arreglo se almacena una bandera, para indicar si esta posición en la matriz es un área deseada a editar con el valor -1, y en caso contrario con 0, esta posición se usa para saber si ya se visito poniéndole el valor 0, al finalizar una iteración los valores correspondientes al área $\Omega$ son restablecidos a -1.
- **Posición 3**: En este punto del arreglo se almacena una bandera, para saber si esta posición en la matriz es un área deseada a editar con el valor -1, y en caso contrario con 0.

- **Posición 4**: Este punto del arreglo solo se almacena en los valores correspondientes al área $\Omega$, esta posición tiene un objeto con todas las coordenadas de los parches que ya fueron probados en este parche, con su respectiva distancia.
- **Posición 5**: Este punto del arreglo solo se almacena en los valores correspondientes al área $\Omega$, esta posición tiene un arreglo de 9 posiciones donde se almacena cada una de las distancias del ultimo parche que se comparo.
- **Posición 6**: Este punto del arreglo solo se almacena en los valores correspondientes al área $\Omega$, esta posición tiene la coordenada $y$ del mejor parche.
- **Posición 7**: Este punto del arreglo solo se almacena en los valores correspondientes al área $\Omega$, esta posición tiene la coordenada $x$ del mejor parche.
- **Posición 8**: Este punto del arreglo solo se almacena en los valores correspondientes al área $\Omega$, esta posición tiene la distancia del mejor parche.
- **Posición 9**: En este punto del arreglo se almacena una bandera, para saber si esta posición en la matriz es un área deseada a editar con el valor -1, y en caso contrario con 0.
- **Posición 10**: Este punto del arreglo solo se almacena en los valores correspondientes al área $\Omega$, esta posición tiene un arreglo de 9 posiciones donde se almacena cada una de las distancias del mejor parche.

Los píxeles dentro del área $\Omega$ serán rellenados como se muestra en la Figura 2.14.

Figura 2.14: Selección del área a reconstruir de la imagen

## 2.7. Inicialización

En la Sección 2.5 se explica a groso modo la Figura 2.12 que resume el algoritmo mediante una secuencia de etapas, en este apartado se explicara la etapa Inicialización de la Figura 2.12.

**Realizar Muestreo**

En este apartado se explica en detalle el desenvolvimiento de ejecutar el paso Realizar Muestreo uniforme sobre la imagen a procesar, este paso se realiza para escoger aleatoriamente los posibles candidatos de cada píxel $p \epsilon \Omega$. Para llevar a cabo tal muestreo se requiere de un generador de números pseudo-aleatorios con distribución uniforme, donde cada valor tenga la misma probabilidad.

Para realizar este muestreo principalmente se utilizaran 3 métodos descritos con anterioridad, y estos son **Alea**, **LFIB4** y **KISS07**.

Es importante realizar un muestreo sobre la parte conocida de la imagen, porque ejecutar este algoritmo a fuerza bruta tiene un alto costo computacional, es decir si se quiere realizar la comparación píxel por píxel (para ciertas imágenes podría tomar muchas horas de procesamiento), por ello este punto con lleva una mayor importancia en la aplicación.

Por ello se utiliza este tipo de solución y que se tiene una gran variedad de parámetros para que el algoritmo no valide o repita una gran cantidad de parches, con esto se garantiza que los parches candidatos sean los mejores.

**Propagación**

Antes de empezar a realizar la propagación primero se va a verificar los mejores parches que ya fueron calculados (exceptuando los parches que ya fueron verificados sobre este parche), estos cumplen con la condición de ser menores a *EnerM*, ademas que la máxima cantidad de parches que son seleccionados es *MaxBest*, esto se hace para acelerar el procesamiento del mejor parche a sustituir.

Las métricas que permiten conocer cual pixel es mejor candidato que otro, están basadas en la suma de las diferencias al cuadrado de las vecindades del píxel incógnita y el píxel seleccionado.

En esta solución, no solo se calcula la diferencias al cuadrado entre el parche original con algún otro del entorno, si no que también se le suma las diferencias al cuadrado entre sus correspondientes al aplicar el filtro de Sobel, y este ultimo es multiplicado por un valor $\lambda$, se hace de esta forma para omitir los parches que tengan muchas variaciones de color.

Para no generar una gran cantidad de parches extras, solo se realiza este paso sobre el mejor parche en cada instante de tiempo, para así empezar la técnica de propagación en el eje *x* y *y*; lo primero que se hace es calcular dos números random para definir la dirección donde se propagara *cProg* parches en el eje *x* y *y*.

Como se dijo anteriormente las direcciones son escogidas aleatoriamente siguiendo las ecuaciones 2.2 y 2.3, donde $0 \leq k < cProg$.

$$q_{kx} = (x \pm k, y) \tag{2.2}$$

$$q_{ky} = (x, y \pm k) \tag{2.3}$$

La aleatoriedad de las direcciones viene dada por la elección del signo. Debido a esto, en esta propuesta se generan dos números aleatorios, uno para el eje $x$ y otro para el eje $y$. En cada caso si el número seleccionado es par, el signo es positivo y en caso de que el numero sea impar se utiliza el signo negativo. En el caso de $y$ se aplica lo mismo que con $x$ al momento de seleccionar el número, pero con la diferencia que se aplica con respecto al eje $y$.

Se debe acotar que *cProg* se define como la máxima cantidad de parches en cada dirección seleccionada con el paso anterior, y se dice máxima porque si al momento de aplicar la propagación se sale del área de la imagen no se tomaran en cuenta esos parches, es decir que solo se toman los parches que estén dentro de nuestra imagen. Gráficamente se están examinando los puntos que se muestran en la Figura 2.15 para un píxel $q$ y un píxel $p$.

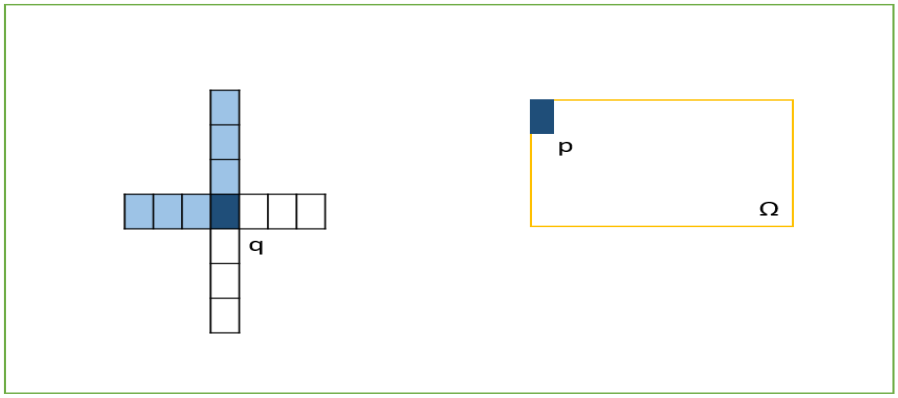


Figura 2.15: Se generan *cProg* píxeles hacia arriba y hacia la izquierda para este caso. Como la elección de la dirección es aleatoria se pueden generar otras 3 dimensiones.

### Búsqueda aleatoria

En el paso anterior se obtuvo una variedad de parches candidatos en base al entorno de tu mejor parche (Exactamente $(cProg - 1) * 2$). Para evitar un mínimo local, se aplica el algoritmo de *Búsqueda Aleatoria*, en donde para cada parche que se encuentre dentro de la imagen y fuera del área a editar se aplica lo siguiente:

Dado cada uno de los parches candidatos obtenidos en el paso anterior se desea mejorar ese valor intentando realizar desplazamientos dentro de un radio $cBus$ estos desplazamiento serán decididos aleatoriamente como se muestra en la Figura 2.16.

La cantidad de números aleatorios calculados son 4, 2 para decidir el sentido y 2 que deciden cuanto se desplaza en *x* e *y* siguiendo la ecuación 2.4, donde k o k1 definen el sentido.

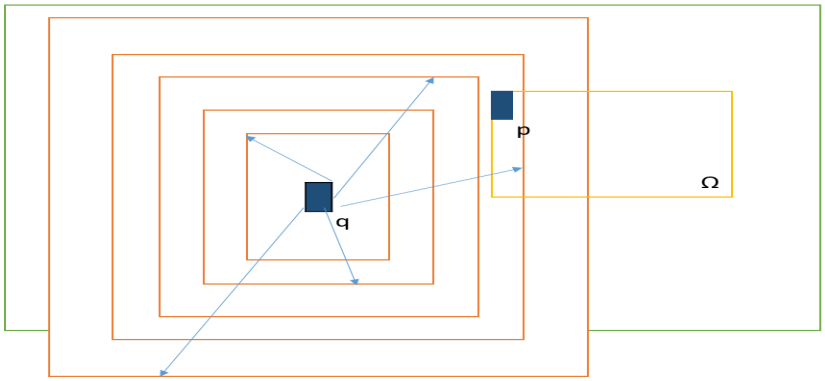


Figura 2.16: Búsqueda aleatoria de puntos a partir de una posición q

$$\begin{aligned} q_{kx} &= (x \pm k * h, y \pm k1 * h1) \\ 0 \leq x - cBus \leq h < x + cBus \\ 0 \leq y - cBus \leq h1 < y + cBus \end{aligned} \tag{2.4}$$

Sin embargo, cuando un nuevo punto es generado y esta fuera de la imagen, este es descartado y se pasa a la siguiente iteración, en vez de limitarlo al borde de la imagen en esa dirección, como se puede ver en la Figura 2.17.

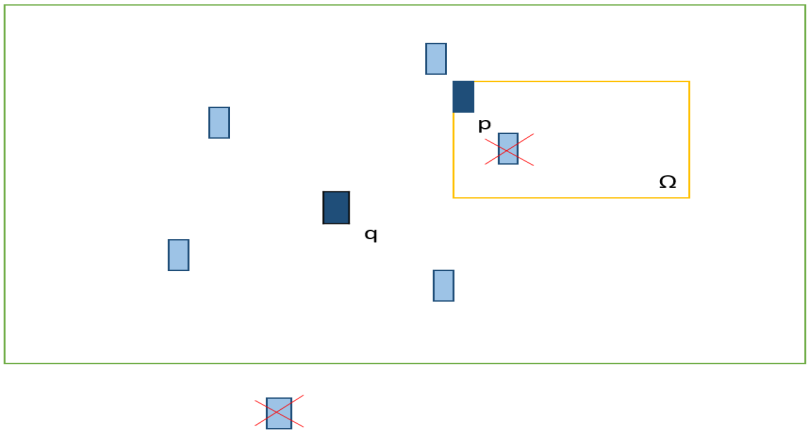


Figura 2.17: Píxeles fuera de la imagen o dentro de Ω son descartados.

**Aplicar dato**

En este paso se aplica uno de los siguientes dos métodos, en el primero se aplica la función de distancia sobre todos los píxeles del parche, esto se hace con el objetivo de obtener el mejor parche para rellenar el área desconocida $\Omega$ siguiendo el método de Darabi con la ecuación 1.22.

En la segunda forma, se aplica las formulas de distancia $D(Q, P) - D(\nabla Q, \nabla P)$, pero en este método se calcula la distancia por votación, en vez de realizar una sumatoria se va a comparar cada parche y con ello se selecciona como mejor candidato al que tenga la mayor cantidad de parches con menor distancia. Donde $D(Q, P)$, es aplicar la suma de distancias al cuadrado en LAB, LCH o LUV, sumado a $D(\nabla Q, \nabla P)$ que es lo mismo de aplicar la suma de distancias al cuadrado en LAB, LCH o LUV pero en la imagen afectada por el filtro de Sobel.

Por último $\lambda$, es una variable constante utilizada para suavizar la distancia entre dos colores de la imagen con filtro Sobel, esto se hace para que el mejor parche no tome en cuenta ruidos tan significativos al momento de sumarse con la distancia entre dos colores de la imagen original.

## 2.8. Procesar textura

Este paso tiene como entrada los mejores parches para todo el área $\Omega$, por lo cual ya se tendría el mejor candidato hasta el momento para el píxel *p* utilizando la ecuación 2.5, siendo estos los primeros valores que toma $\Omega$ en el paso de inicialización.

$$E(\delta) = \sum_{t \subset D0} \parallel M(p+t) - M(q+t) \parallel^2 \tag{2.5}$$

Donde $D0$ es el valor que se suma a $p$ o $q$ para colocarse en el entorno de los mismos, mientras $M(p+t)$ o $M(q+t)$ es el valor de color para la posición especificada en nuestra matriz.

En este paso se aplican los mismos procesos de Propagación, Búsqueda aleatoria y Aplicar dato; la cantidad de veces que el usuario selecciono al principio del algoritmo con el parámetro de nombre *totpb*.

## 2.9. Aplicar función de Energía

En este paso se aplica el filtro de la mediana sobre el área $\Omega$, esta sera nuestra imagen base al momento de aplicar Poisson Blending, para así obtener nuevos parches que se adapten al entorno.

**Filtro suavizado (Mediana o Gaussiano)**

En este paso se aplica sobre el área Ω el filtro de suavizado seleccionado por el usuario (la mediana o Gaussiano), este filtro se hace para eliminar el ruido o detalles que pueden reducir las variaciones de intensidad entre píxeles y la imagen resultante es usada en el próximo paso Poisson Blending como imagen base.

**Poisson Blending**

En este paso se aplica la técnica Poisson Blending, se tomara como imagen base la que se le aplico el filtro de suavizado (Mediana o Gaussiano) sobre el área Ω, y como imagen a sustituir la imagen sin aplicar el filtro de suavizado, para así evitar mantener el difulminado que se genera al utilizar el filtro de suavizado, así obteniendo el resultado de la Figura 2.18.

Figura 2.18: Resultado de aplicar todos los pasos hasta Poisson Blending

## 2.10. Proyección

En este paso se vuelve a realizar los 3 pasos anteriores Propagación, Búsqueda aleatoria y Aplicar dato, esto se hace porque al realizar Poisson Blending en los parches dentro del área Ω, se borra un poco el ruido o variaciones de intensidad, lo que hace surgir nuevos parches.

## 2.11. Consideraciones finales

Después de haber realizado todas las etapas descritas en este capítulo obtendremos nuestra imagen final después de aplicar la técnica *Inpainting*. Se puede notar que este algoritmo es complejo y puede tardarse mucho tiempo dependiendo de los parámetros especificados; WebCL aunque sea muy útil para realizar paralelismo en las distintas operaciones descritas en este capítulo, tiene unas pre condiciones que no son las esperadas para un navegador web, pero quitando esos inconvenientes se unifico las distintas etapa para obtener los resultados descritos en el Capítulo 3.

# Capítulo 3

# Pruebas y Resultados

En el capítulo anterior se detalla la solución del algoritmo, pero ahora es necesario evaluar su rendimiento y determinar los parámetros adecuados donde se obtiene el resultado esperado. En el transcurso de este capítulo están las diferentes pruebas realizadas para reconstruir el área desconocida de cada imagen, así como sus parámetros de ejecución y el análisis de los resultados obtenidos.

## 1. Pruebas cuantitativas

En este apartado se realiza un conjunto de pruebas con distintos parámetros, con el objeto de seleccionar los parámetros que den la mejor solución desde el punto de vista cuantitativo.

### 1.1. Ambiente de pruebas

En esta sección se expone el entorno de hardware y software donde se realizó las pruebas:

#### Requerimientos de hardware

Las distintas pruebas del algoritmo se realizaron en dos equipos con especificaciones de Hardware distintas, los cuales se denotan en la siguiente tabla:

| Sistema | Procesador | Memoria RAM | Sistema operativo |
|---|---|---|---|
| 1 | AMD Phenom(tm) II X6 1045T 2.70GHz | 6 GB | Windows 8 |
| 2 | AMD Phenom(tm) II X6 1045T 2.70GHz | 4 GB | Windows 7 |

Para usar WebCL lo primero que se necesita es tener instalado OpenCL en la computadora, por ello una de las características más importantes es la tarjeta gráfica, ya que gracias a dicho Hardware se explota el paralelismo en OpenCL, la siguiente tabla menciona las tarjetas gráficas de los sistemas de la tabla anterior:

| Sistema | Tarjeta | Memoria | Núcleos de CUDA |
|---|---|---|---|
| 1 | GeForce GTX 750 | 1024 MB | 512 |
| 2 | EVGA GeForce GTX 560 | 1024 MB | 336 |

**Requerimientos de software**

Los requerimientos de software son los siguientes:

- **Sistema Operativo XP en adelante**.
- **Versión de Firefox 33** (se puede descargar desde la aplicación).
- **Plugin WebCL** (se puede descargar desde la aplicación).
- **OpenCL** (para instalar OpenCL si la tarjeta de vídeo es NVIDIA use esta bibliografía para descargarlo [30] y si es AMD esta otra [31], esto permite poder utilizar en la aplicación WebCL).

## 1.2. Descripción de los escenarios

Se efectuaron experimentos en diferentes niveles. El primer nivel hace un estudio a fondo de los parámetros, realizando tantas ejecuciones como resulte pertinente, esta prueba se ejecuto en dos partes primero una cuantitativa y otra cualitativa. Para luego pasar a una prueba de segundo nivel donde se realizan pruebas con las mejores combinaciones cuantitativas y cualitativas obtenidas del paso anterior con respecto a 4 nuevas imágenes.

**Parámetros**

Los parámetros utilizados para las distintas pruebas están especificados a fondo en el Capítulo 2 Sección 2.6.

**Consideraciones previas**

En el Capítulo 3 Sección 1.1 se especifican los distintos sistemas donde se realizo las distintas pruebas. En la mayoría de los siguientes experimentos se realizo con el sistema 1. Esto se debe a su capacidad de procesamiento ( superior a los otros sistemas), lo cual permite ejecutar el algoritmo en un menor tiempo.

Al momento de implementar esta solución se notó que WebCL no soporta realizar mucho procesamiento por llamada, por eso se distribuyó el algoritmo por partes, dividiendo la carga computacional en fragmentos más pequeños, por ejemplo al calcular el Filtro Sobel no se puede hacer sobre la imagen completa se tiene que dividir la imagen en 4 y realizar 4 invocaciones a WebCL por cada una de esas áreas.

## 1.3. Experimentos

**Experimento (nivel 1) Parte 1**

El objetivo de este experimento es elegir las mejores combinaciones de los diferentes valores, como estas pruebas son desde el punto de vista cuantitativo se selecciona los mejores resultados con el menor tiempo posible según los 3 tipos de escalas de colores LUV, LAB y LCH.

Para luego en la siguiente sección que son las pruebas desde el punto de vista cualitativo seleccionar las soluciones que obtengan un resultado visual agradable con respecto a los mejores resultados cuantitativos.

Figura 3.1: Selección del área a reconstruir de la imagen para este conjunto de pruebas, con un tamaño de 126x84 píxeles

En la Figura 3.1, se puede ver el área $\Omega$ que se desea remplazar mediante la técnica *Inpainting*, a continuación se muestra las tablas que hacen referencia a 16 pruebas que se realizo con cada uno de los parámetros especificados, y el tiempo que se tarda en substituir el área $\Omega$ de la Figura 3.1:

- **TamP**: Es un entero que define el tamaño que tomara el píxel.
- **porctN**: Es un float que define la cantidad de parches de prueba, este porcentaje se realiza con respecto al tamaño de la imagen.
- **maxN**: Es un entero que define la máxima cantidad de parches de prueba.
- **seedr**: Es un float que define la semilla de la función random.
- **cProg**: Es un entero que define la cantidad de parches que se propagara.
- **cBus**: Es un entero que define el rango de búsqueda del nuevo parche.
- **totpb**: Es un entero que define el número de iteraciones que tendrá procesar textura.
- **EnerM**: Es un entero que define cual es la distancia máxima entre dos vecindades,
- **MaxBest**: Es un entero que define la cantidad de parches que se reciclan.
- **poissonmax**: Es un entero que define la cantidad de iteraciones que tiene Possion Blending.
- **Filtro**: Es un ComboBox donde se escoge el tipo de filtro.
- **Suavizado**: Es un ComboBox donde se escoge el tipo de filtro de suavizado.
- **color**: Es un ComboBox donde se escoge el tipo de escala de color.
- **Random**: Es un ComboBox donde se escoge el tipo de Random.
- **WebCL suavizado**: Es un CheckBox donde se decide si WebCL es utilizado en el tipo de suavizado.
- **distancia**: Es un CheckBox donde se decide que tipo de distancia se calculara.
- **Tiempo**: Es el tiempo resultante de realizar este caso de prueba.

| TamP | porctN | maxN | seedr | cProg | cBus | totpb | EnerM | MaxBest | poissonmax |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.05 | 7 | 0.42 | 3 | 5 | 1 | 50 | 100 | 150 |
| 2 | 0.05 | 20 | 0.42 | 3 | 5 | 1 | 50 | 100 | 150 |
| 2 | 0.1 | 20 | 0.42 | 3 | 5 | 1 | 50 | 100 | 150 |
| 2 | 0.1 | 20 | 0.42 | 5 | 6 | 1 | 30 | 100 | 150 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 30 | 100 | 150 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 2 | 50 | 75 | 100 |
| 3 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 9 | 1 | 50 | 75 | 100 |
| 4 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 4 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 4 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |

| Filtro | Suavizado | color | Random | WebCL suavizado | distancia | Tiempo |
|---|---|---|---|---|---|---|
| Sobel | mediana | LAB | Alea | false | false | 1 m 30 s |
| Sobel | mediana | LAB | Alea | false | false | 1 m 33 s |
| Sobel | mediana | LAB | Alea | false | false | 1 m 47 s |
| Sobel | mediana | LAB | Alea | false | false | 2 m 11 s |
| Sobel | mediana | LAB | Alea | false | true | 2 m 18 s |
| laplaciano | mediana | LAB | Alea | false | true | 1 m 27 s |
| laplaciano | mediana | LUV | Alea | false | true | 1 m 10 s |
| laplaciano | mediana | LAB | Alea | false | true | 1 m 29 s |
| laplaciano | mediana | LUV | Alea | false | true | 1 m 55 s |
| laplaciano | mediana | LCH | Alea | false | false | 2 m 19s |
| laplaciano | mediana | LUV | Alea | false | false | 1 m 12 s |
| laplaciano | gaussiano | LAB | Alea | false | false | 1 m 48 s |
| laplaciano | gaussiano | LAB | Alea | false | false | 1 m 47 s |
| laplaciano | gaussiano | LUV | Alea | false | false | 1 m 18 s |
| laplaciano | gaussiano | LCH | Alea | false | false | 2 m 10 s |
| laplaciano | gaussiano | LCH | Alea | false | true | 2 m 17 s |

Figura 3.2: Prueba número 7, este resultado fue obtenido utilizando la escala de color LUV en 1 min 10 seg

Desde el punto de vista cuantitativo el mejor tiempo lo tiene la prueba número 7 con 1 min 10 seg utilizando la escala de color LUV y obteniendo como resultado la Figura 3.2, luego para la escala de color LAB la prueba numero 6 con 1 min y 27 seg (este resultado se puede ver en la Figura 3.3 ), y por ultimo para la escala de color LCH la prueba número 15 con 2 min y 10 seg cuyo resultado se puede ver en la Figura 3.4.

Figura 3.3: Prueba número 6, este resultado fue obtenido utilizando la escala de color LAB en 1 min 27 seg

Figura 3.4: Prueba número 15, este resultado fue obtenido utilizando la escala de color LCH en 2 min 10 seg con TamP igual a 4

# 2. Pruebas cualitativas

En este apartado se analiza los resultados obtenidos en las pruebas cuantitativas y se decide cual es el mejor resultado cualitativo y cuantitativo.

## 2.1. Experimentos

En esta sección se muestra los resultados desde el punto de vista cualitativo y basado en los resultados cuantitativos obtenidos en la anterior sección; en el Experimento (nivel 2) se realiza una comparación basada en los parámetros del mejor resultado obtenido en Experimento (nivel 1) Parte 2, contra 4 nuevas imágenes.

### Experimento (nivel 1) Parte 2

Los resultados obtenidos en la sección anterior, desde el punto de vista cualitativo se puede concluir que a partir de TamP mayor o igual a 4 los parches tienen información no tan significativa y se obtiene resultados poco agradables como se puede ver en las Figuras 3.4, 3.5 y 3.6, viendo las imágenes desde lejos se nota una gran discrepancia entre los píxeles del entorno y al ver en detalle se puede nota muchas imperfecciones.

Figura 3.5: Prueba número 14, este resultado fue obtenido utilizando la escala de color LUV en 1 min 18 seg con TamP igual a 4

Figura 3.6: Prueba número 13, este resultado fue obtenido utilizando la escala de color LAB en 1 min 47 seg con TamP igual a 4

Luego al ver en detalle los resultados obtenidos con TamP igual a 3 (estos se muestran en las Figuras 3.8, 3.7 y 3.9) desde lejos se puede notar unos resultados que siguen una misma secuencia de píxeles, pero cuando empiezas a enfocarte en la imagen se nota los cambios, también se nota que aunque LUV sea el más rápido de las 3 escalas de colores, este selecciona algunos píxeles de color verde dentro del área que debe ser una roca, lo cual genera un poco de inconsistencia en la imagen resultante.

Figura 3.7: Prueba número 11, este resultado fue obtenido utilizando la escala de color LUV en 1 min 12 seg con TamP igual a 3

Viendo desde lejos el resultado de la Figura 3.8 que usa el espacio de color LAB, se nota que desde lejos se obtiene un resultado muy satisfactoria, pero si nos enfocamos se llega a notar que el mismo toma fragmentos azules que no deseados, pero la aparición de estos no es incorrecta, porque los parches de su entorno son sustituidos por las montañas de la izquierda cuyo entorno es cielo o agua. Lo cual en este caso permite su aparición en nuestro resultado final.

Figura 3.8: Prueba número 12, este resultado fue obtenido utilizando la escala de color LAB en 1 min 48 seg con TamP igual a 3

El resultado obtenido en la Figura 3.9 utilizando la escala de color LCH, ademas de ser el mas lento de los 3, con 2 min y 19 seg también desde lejos es el que obtiene el peor resultado de los 3, dado que el entorno no se logra acoplar de forma satisfactoria.

Después de analizar estos 3 resultados para TamP igual a 3 se puede concluir que aunque desde el punto de vista cuantitativo la Figura 3.7 es la mejor, desde el punto de vista cualitativo la mejor es la Figura 3.8 con un tiempo no muy lejano al mejor resultado.

Figura 3.9: Prueba número 10, este resultado fue obtenido utilizando la escala de color LCH en 2 min 19 seg con TamP igual a 3

A continuación se estudiaran los resultados obtenidos con TamP igual a 2 (estos se pueden ver en las Figuras 3.2, 3.3 y 3.10), entre los resultados obtenidos para TamP igual 2 se noto que las Figuras 3.2 y 3.3 son las que tienen uno de los mejores resultados cuantitativos.

La Figura 3.2 en 1 min 10 seg utilizando la escala de color LUV, pero este resultado desde el punto de vista cualitativo desde lejos se ve agradable pero con una gran discrepancia y es que selecciono muchos píxeles azules del cielo o agua, lo cual genera contraste en la imagen final.

Analizando la Figura 3.2 que se tarda 1 min 27 seg utilizando la escala de color LAB, podemos presenciar un resultado que desde lejos es satisfactorio y se acopla en gran medida con el entorno que le rodea, mientras que si lo vemos en detalle podemos notar unos píxeles de color azul en el centro de la imagen, omitiendo ese ultimo detalle este resultado es uno de los mejores que se obtuvo dentro de las 16 pruebas descritas.

Por ultimo en la Figura 3.10 que se tarda 2 min y 17 seg utilizando la escala de color LCH podemos presenciar un resultado poco satisfactorio desde lejos porque este no sigue la secuencia de píxeles de la roca y selecciona algunos fragmentos verde oscuro.

Figura 3.10: Prueba número 16, este resultado fue obtenido utilizando la escala de color LCH en 2 min 17 seg con TamP igual a 2

Después de realizar las distintas pruebas, junto a los distintos análisis cuantitativos y cualitativos se llega a la conclusión que los 4 mejores resultados son las Figuras 3.3, 3.2, 3.7 y 3.8.

La información de estas son: Figura 3.3 con 1 min 27 seg utilizando la escala de color LAB (TamP igual 2), Figura 3.2 con 1 min 10 seg utilizando la escala de color LUV (TamP igual 2), Figura 3.7 con 1 min 12 seg utilizando la escala de color LUV (TamP igual 3) y Figura 3.8 con 1 min 48 seg utilizando la escala de color LAB (TamP igual 3).

Con este conjunto de pruebas, se pudo notar que al aplicar LCH no se obtiene resultados satisfactorios cualitativos o cuantitativos.

**Experimento (nivel 2)**

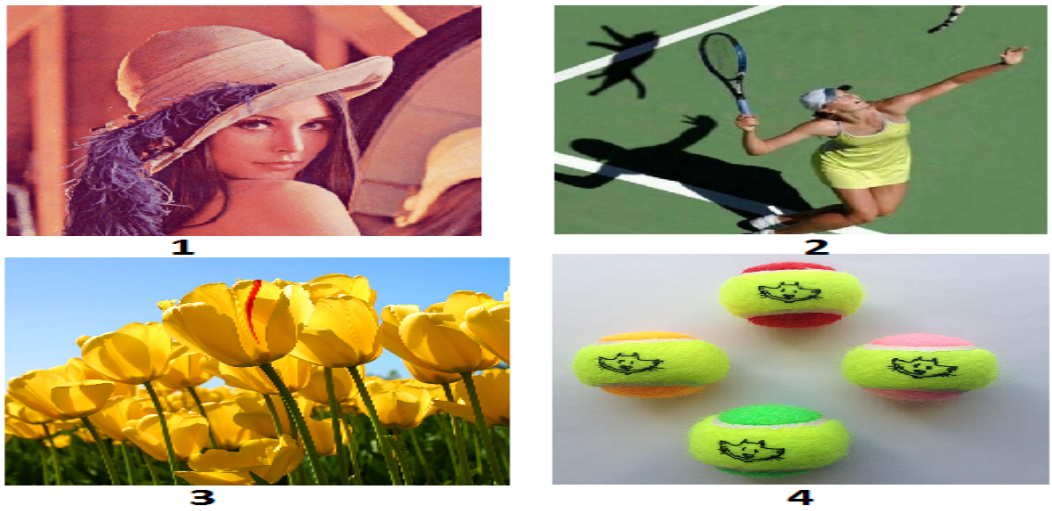


Figura 3.11: Imagen para pruebas de nivel 2

En la Figura 3.11 se muestra las 4 imágenes de prueba para el siguiente conjunto de pruebas sobre los mejores parámetros seleccionados en el nivel anterior. Ahora se muestra el resultado cuantitativo de aplicar sobre la primera imagen de la Figura 3.11 en el área Ω de la Figura 3.12:

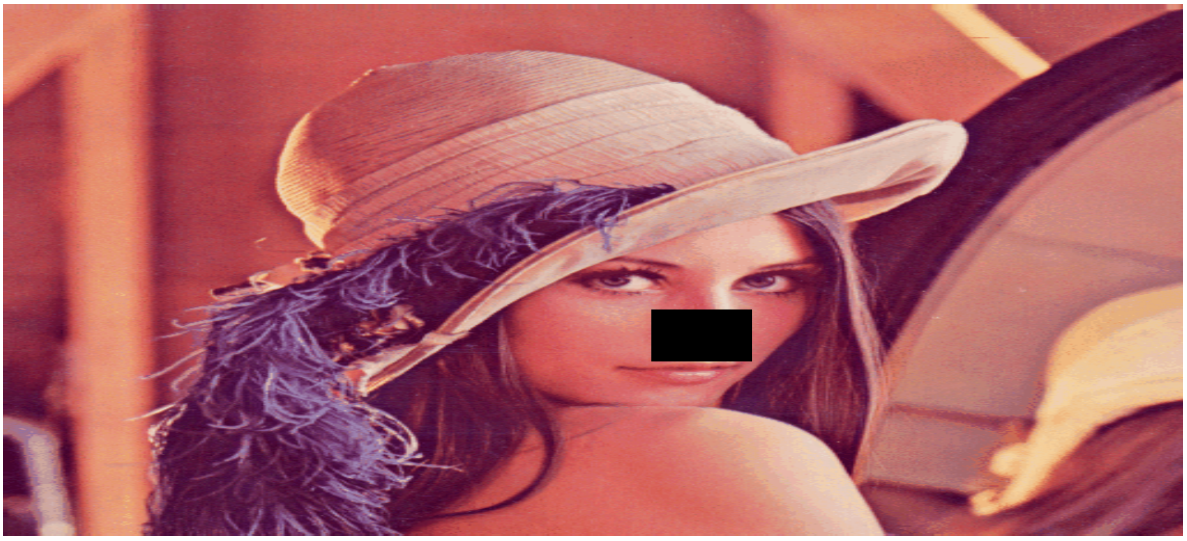

Figura 3.12: Área Ω a sustituir de la primera imagen de prueba 80x130 píxeles

En la siguiente tabla se muestras los resultados de aplicar *Inpainting* sobre el área $\Omega$ que se puede ver en la Figura 3.12:

| TamP | porctN | maxN | seedr | cProg | cBus | totpb | EnerM | MaxBest | poissonmax |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 9 | 1 | 50 | 75 | 100 |

| Filtro | Suavizado | color | Random | WebCL suavizado | distancia | Tiempo |
|---|---|---|---|---|---|---|
| laplaciano | mediana | LAB | Alea | false | true | 1 m 1 s |
| laplaciano | mediana | LUV | Alea | false | true | 45 s |
| laplaciano | mediana | LUV | Alea | false | false | 47 s |
| laplaciano | gaussiano | LAB | Alea | false | false | 1 m 37 s |

Figura 3.13: Resultado obtenido para la escala de color LAB en 1 min 1 seg con TamP igual a 2

Desde el punto de visto cuantitativo en este primer conjunto de pruebas el mejor resultado es la segunda prueba cuyo resultado cualitativo se puede ver en la Figura 3.14 con 45 seg, pero este resultado como en de la Figura 3.13 con 1 min 1 seg no es muy agradable viendo de lejos o de cerca, es decir se nota que fue sustituidos los píxeles, es decir que en estos casos no se obtuvieron buenos resultado con TamP igual 2.

Figura 3.14: Resultado obtenido para la escala de color LUV en 45 seg con TamP igual a 2

Figura 3.15: Resultado obtenido para la escala de color LUV en 47 seg con TamP igual a 3

Figura 3.16: Resultado obtenido para la escala de color LAB en 1 min 37 seg con TamP igual a 3

En los resultados obtenidos con TamP igual a 3 se nota un resultado ameno, es decir un resultado agradable para la persona que lo vea de lejos como se puede ver en la Figura 3.15 y 3.16; entre los dos resultados el que tiene un mejor resultado de cerca es la Figura 3.16. Ahora se muestra el resultado cuantitativo de aplicar sobre la segunda imagen de la Figura 3.11 en el área $\Omega$ que se puede ver en la Figura 3.17:

Figura 3.17: En esta imagen se muestra el área $\Omega$ a sustituir de la segunda imagen de prueba 224x232 píxeles

En la siguiente tabla se muestra los resultados de aplicar *Inpainting* sobre el área $\Omega$ que se puede ver en la Figura 3.17:

| TamP | porctN | maxN | seedr | cProg | cBus | totpb | EnerM | MaxBest | poissonmax |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 9 | 1 | 50 | 75 | 100 |

| Filtro | Suavizado | color | Random | WebCL suavizado | distancia | Tiempo |
|---|---|---|---|---|---|---|
| laplaciano | mediana | LAB | Alea | false | true | 11 m 26 s |
| laplaciano | mediana | LUV | Alea | false | true | 9 m |
| laplaciano | mediana | LUV | Alea | false | false | 4 m 39 s |
| laplaciano | gaussiano | LAB | Alea | false | false | 5 m 37 s |

Figura 3.18: Resultado obtenido para la escala de color LAB en 11 min 26 seg con TamP igual a 2

Desde el punto de visto cuantitativo en este segundo conjunto de pruebas, el mejor resultado lo tiene la tercera prueba, cuyo resultado cualitativo se puede ver en la Figura 3.20 con 4 min 39 seg pero desde el punto de vista cualitativo los 4 resultados no son satisfactorios ya que no se propaga la línea de la cancha de tenis como se puede ver en la Figuras 3.18, 3.19, y 3.21; la única solución que propaga la línea es la Figura 3.20, pero no se obtiene el resultado esperado.

Figura 3.19: Resultado obtenido para la escala de color LUV en 9 min con TamP igual a 2

Figura 3.20: Resultado obtenido para la escala de color LUV en 4 min 39 seg con TamP igual a 3

Figura 3.21: Resultado obtenido para la escala de color LAB en 5 min 37 seg con TamP igual a 3

Ahora se muestra el resultado cuantitativo de aplicar sobre la tercera imagen de la Figura 3.11 en el área $\Omega$ que se puede ver en la Figura 3.22:

Figura 3.22: Área $\Omega$ a sustituir de la tercera imagen de prueba 160x48 píxeles

En la siguiente tabla se muestra los resultados de aplicar *Inpainting* sobre el área $\Omega$ que se puede ver en la Figura 3.22:

| TamP | porctN | maxN | seedr | cProg | cBus | totpb | EnerM | MaxBest | poissonmax |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 9 | 1 | 50 | 75 | 100 |

| Filtro | Suavizado | color | Random | WebCL suavizado | distancia | Tiempo |
|---|---|---|---|---|---|---|
| laplaciano | mediana | LAB | Alea | false | true | 1 m 31 s |
| laplaciano | mediana | LUV | Alea | false | true | 1 m 14 s |
| laplaciano | mediana | LUV | Alea | false | false | 1 m 4 s |
| laplaciano | gaussiano | LAB | Alea | false | false | 1 m 16 s |

Figura 3.23: Resultado obtenido para la escala de color LAB en 1 min 31 seg con TamP igual a 2

Desde el punto de vista cuantitativo en este tercer conjunto de pruebas, el mejor resultado lo tiene la tercera prueba cuyo resultado cualitativo se puede ver en la Figura 3.25 con 1 min 14 seg, siendo este un resultado agradable con un pequeño fallo en la sombra, ya que esta tiene una forma rara, pero es mucho mejor si se compara a las Figuras 3.23 y 3.24 ambas con TamP igual 2.

Figura 3.24: Resultado obtenido para la escala de color LUV en 1 min 14 seg con TamP igual a 2

Figura 3.25: Resultado obtenido para la escala de color LUV en 1 min 4 seg con TamP igual a 3

Figura 3.26: Resultado obtenido para la escala de color LAB en 1 min 16 seg con TamP igual a 3

En este tercer conjunto de prueba se puede notar que las pruebas con el valor TamP igual a 3 son los mejores resultados, un resultado agradable para una persona que lo vea de lejos como se puede ver en la Figura 3.26 y 3.25; entre los dos la que tiene un mejor resultado de cerca es la Figura 3.26, ya que en este resultado la sombra no tiene ese pequeña discrepancia. En la tabla de a seguir se muestra el resultado cuantitativo de aplicar sobre la cuarta y ultima imagen de la Figura 3.11 en el área $\Omega$ que se puede ver en la Figura 3.27:

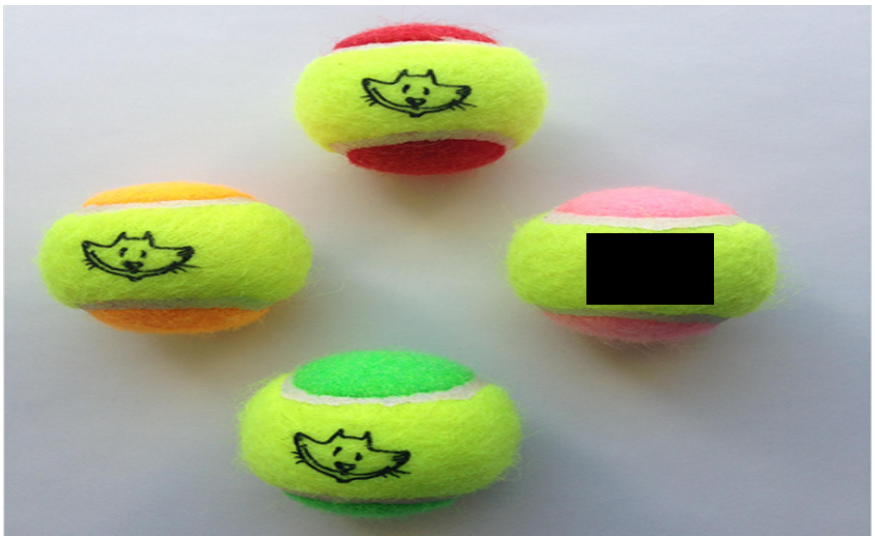

Figura 3.27: Figura que muestra el área $\Omega$ que sera sustituida en esta prueba 76x128 píxeles

En la siguiente tabla se muestra los resultados de aplicar *Inpainting* sobre el área Ω que se puede ver en la Figura 3.22:

| TamP | porctN | maxN | seedr | cProg | cBus | totpb | EnerM | MaxBest | poissonmax |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 2 | 0.1 | 20 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 6 | 1 | 50 | 75 | 100 |
| 3 | 0.05 | 10 | 0.26 | 5 | 9 | 1 | 50 | 75 | 100 |

| Filtro | Suavizado | color | Random | WebCL suavizado | distancia | Tiempo |
|---|---|---|---|---|---|---|
| laplaciano | mediana | LAB | Alea | false | true | 1 m 42 s |
| laplaciano | mediana | LUV | Alea | false | true | 1 m 19 s |
| laplaciano | mediana | LUV | Alea | false | false | 1 m 11 s |
| laplaciano | gaussiano | LAB | Alea | false | false | 1 m 17 s |

Figura 3.28: Resultado obtenido para la escala de color LAB en 1 min 42 seg con TamP igual a 2

Desde el punto de visto cuantitativo en este cuarto conjunto de pruebas, el mejor resultado lo tiene la tercera prueba cuyo resultado cualitativo se puede ver en la Figura 3.30 con 1 min 11 seg, siendo este un mejor resultado si es comparado a las Figuras 3.28 y 3.29 ambas con TamP igual 2. Pero tampoco es un resultado satisfactorio ya que en la zona inferior derecha del área Ω es substituido por unos parches oscuros.

Figura 3.29: Resultado obtenido para la escala de color LUV en 1 min 19 seg con TamP igual a 2

Figura 3.30: Resultado obtenido para la escala de color LUV en 1 min 11 seg con TamP igual a 3

Figura 3.31: Resultado obtenido para la escala de color LAB en 1 min 17 seg con TamP igual a 3

Como se menciono anteriormente los mejores resultado en este cuarta prueba son las que tiene el valor TamP igual a 3, en el cual se obtiene un resultado agradable para una persona que lo vea de lejos como se puede ver en la Figura 3.31 y 3.30; entre los dos últimos, la que tiene un mejor resultado de cerca es la Figura 3.31, ya que en este no se genera parches oscuros en la zona inferior izquierda y el valor de los parches es mejor distribuido alrededor de la imagen.

# 3. Consideraciones finales

Al realizar los distintos experimentos cuantitativos y cualitativos se puede llegar a la conclusión de que el mejor de los resultados, el cual presenta menos fallas a nivel cuantitativo y cualitativo son con los parámetros expuestos en la siguiente tabla:

| TamP | porctN | maxN | seedr | cProg | cBus | totpb | EnerM | MaxBest | poissonmax |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 0.05 | 10 | 0.26 | 5 | 9 | 1 | 50 | 75 | 100 |

| Filtro | Suavizado | color | Random | WebCL suavizado | distancia |
|---|---|---|---|---|---|
| laplaciano | gaussiano | LAB | Alea | false | false |

# Capítulo 4

# Conclusiones y Trabajos futuros

## 1. Conclusiones

El algoritmo de *Inpainting* es empleado para reconstruir partes dañadas o indeseadas de una imagen. Es un algoritmo complejo, especialmente cuando es automático y requiere una intervención mínima del usuario, como al momento de seleccionar el área indeseada, la selección de los parámetros al que mejor se adapte la imagen, la sección donde el usuario sacará los posibles parches a substituir o el uso de programas donde se indique que parches utilizar.

Con esto, se quiere decir que existen una gran variedad de soluciones o enfoques para el algoritmo *Inpainting*, que dependiendo de los datos que le suministre el usuario, puede llegar a ser más complejo.

En este trabajo, se presentó una solución para el navegador Firefox utilizando nada más el lado cliente y la tecnología WebCL. Todo esto, con el fin de acelerar el procesamiento de los cálculos, dado que WebCL permite utilizar OpenCL desde JavaScript. Sin embargo, WebCL lo que trajo fue un gran número de inconvenientes al momento de realizar esta solución. Inicialmente sucedió que la página principal que proveía el plugin dejó de funcionar, trayendo como consecuencia el tener que buscar otra forma para utilizar WebCL en nuestro navegador con total normalidad. Por ello, se colocó la condición que para utilizar la herramienta WebCL se tiene que instalar Firefox 33.

Luego de haber delimitado las condiciones necesarias para usar WebCL en nuestro navegador, nos encontramos con la presencia de otros problemas. La idea que se tenía inicialmente de ejecutar una solución única, que solo utilice WebCL. No se pudo realizar, debido al manejo de los hilos por parte de WebCL, puesto que al momento de utilizar numerosas operaciones, este inhibe al navegador mostrando así un mensaje de que OpenCL dejó de funcionar, generando así que el navegador se quede inhibido indefinidamente esperando por la solución proporcionada por OpenCL, por ello se cambió a una solución donde sólo se realiza las iteraciones puntuales o necesarias a WebCL.

Como se pude ver en las pruebas realizadas anteriormente, el resultado final, y la rapidez del mismo depende fundamentalmente de los parámetros que son suministrados por el usuario. También se puede observar, que no existe para éste algoritmo una combinación de estos parámetros que funcione para todas las imágenes.

## 2. Trabajos futuros

Con el objeto de mejorar la solución de la técnica *Inpainting* en una aplicación web, se proponen los siguientes cambios. Por los momentos, se tiene una aplicación que realiza la técnica *Inpainting* utilizando WebCL o solo Javascript, además de varios efectos extras. Donde se puede inhibir el navegador Firefox por la gran carga computacional que tiene la solución de la técnica *Inpainting*, por ello se propone un conjunto de mejoras importantes:

- **Realizar una solución que sirva para todos los navegadores.**
- **Realizar una solución donde el servidor se encargue del procesamiento.**
- **Utilizar OpenCL en el servidor para acelerar los distintos procesamientos.**
- **Permitir reconstruir varias áreas a la vez.**
- **Conseguir una mejor función de energía para la solución final.**

Esta solución, depende totalmente de los parámetros suministrados como se notó en las pruebas realizadas, donde la solución suele acoplarse en gran medida con el entorno que le rodea. Sin embargo, aun le falta utilizar una mejor función de energía con el fin de obtener un resultado que sea menos perceptible el cambio.

# Bibliografía


[1] G. Rafael y R. Woods, "Digital image processing," *Prentice Hall*, vol. 3ra edición, 2012.

[2] G. Prieto, "Interpolación (clase 4)," 2011.

[3] digitalfotored, "PÃxeles: Los puntos de una imagen." [En línea]. Disponible en: http://www.digitalfotored.com/imagendigital/pixelesimagen.htm

[4] fotonostra, "Modos de color: Rgb, cmyk y srgb." [En línea]. Disponible en: http://www.fotonostra.com/grafico/rgb.htm

[5] BabenbergerstraÃe, "Advancing knowledge and providing standardization to improve the lighted environment." [En línea]. Disponible en: http://www.cie.co.at/index.php/LEFTMENUE/About+us

[6] nquinlan, "Better random numbers for javascript." [En línea]. Disponible en: https://github.com/nquinlan/better-random-numbers-for-javascript-mirror

[7] Adobe, "Color models cieluv." [En línea]. Disponible en: http://dba.med.sc.edu/price/irf/Adobe_tg/models/cieluv.html

[8] konicaminolta, "Entendiendo el espacio de color cie l*c*h*." [En línea]. Disponible en: http://sensing.konicaminolta.com.mx/2015/08/entiendiendo-el-espacio-de-color-cie-lch/

[9] R. Medina y J. Bellera, "Bases del procesamiento de imágenes médicas." 1999.

[10] Gimp, "Sobel." [En línea]. Disponible en: https://docs.gimp.org/es/plug-in-sobel.html

[11] UAM, "Procesamiento de imagenes." [En línea]. Disponible en: http://arantxa.ii.uam.es/~tacc1/mm_05/Slides/2per_page_pdf/J/07.pdf

[12] P. Pietro y J. Malik, "Scale-space and edge detection using anisotropic diffusion," 1990.

[13] P. Perez, "Poisson image editing." [En línea]. Disponible en: https://www.cs.jhu.edu/~misha/Fall07/Papers/Perez03.pdf

[14] B. Marcelo y G. Sapiro, "Image inpainting." *In Proceedings SIGGRAPH 2000, Computer Graphics Proceedings,Annual Conference Series.*, 2000.

[15] M. Olveira y Y. Chang, “Fast digital image inpainting,” 2001.

[16] T. Chan y J. Shen, “Non-texture inpainting by curvature-driven diffusions (cdd),” 2001.

[17] A. Efros y K. Leung, “Texture synthesis by nonparametric sampling,” 1999.

[18] Y. Xu y H. Shum, “Chaos mosaic: Fast and memory efficient texture synthesis,” 2000.

[19] Q. Wu y Y. Yu, “Feature matching and deformation for texture synthesis,” 2004.

[20] O. Luis, “Construcción de kernels y funciones de densidad de probabilidad,” 2013.

[21] A. Criminisi y K. Toyama, “Region filling and object removal by exemplar based image inpainting,” 2004.

[22] Y. Wexler y M. Irani, “Space-time video completion,” 2007.

[23] C. Barnes y D. Goldman, “Patchmatch: A randomized correspondence algorithm for structural image editing,” 2009.

[24] M. Hadhoud y S. Shenoda, “Digital images inpainting using modified convolution based method,” 2009.

[25] S. Darabi y E. Shechtman, “Image melding: Combining inconsistent images using patch-based synthesis,” 2012.

[26] K. He y J. Sun, “Statistical image completion,” 2012.

[27] J.-B. Huang y J. Kopf, “Image completion using planar structure guidance,” *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)*, vol. 33, núm. 4, p. to appear, 2014.

[28] Khronos, “Web computing language,” 2015. [En línea]. Disponible en: https://www.khronos.org/webcl/

[29] MD, “Gpu apps for your browser: first webcl demos,” 2011. [En línea]. Disponible en: http://www.mdproductions.ca/news/gpu-apps-for-your-browser-first-webcl-demos-are-here

[30] NVIDIA, “Nvidia driver downloads.” [En línea]. Disponible en: http://www.nvidia.com/Download/index.aspx?lang=en-us

[31] AMD, “Amd opencl 2.0 driver.” [En línea]. Disponible en: http://support.amd.com/en-us/kb-articles/Pages/OpenCL2-Driver.aspx

[32] Khronos, “Webcl specification.” [En línea]. Disponible en: https://www.khronos.org/registry/webcl/specs/latest/1.0/

[33] Nokia, “Webcl nokia.” [En línea]. Disponible en: webcl.nokiaresearch.com

[34] toaarnio, "Webcl toaarnio." [En línea]. Disponible en: https://github.com/toaarnio/webcl-firefox

[35] amd, "Webcl implementation for chromium." [En línea]. Disponible en: https://github.com/amd/Chromium-WebCL

[36] Khronos, "Opengl shading language," 2015. [En línea]. Disponible en: https://www.khronos.org/opengl/

[37] A. Chavez, "Si eres desarrollador web, debes utilizar bootstrap y punto." 2013. [En línea]. Disponible en: https://alanchavez.com/si-eres-desarrollador-web-debes-utilizar-bootstrap-y-punto/

[38] Khronos, "Open computing language," 2015. [En línea]. Disponible en: https://www.khronos.org/opencl/

[39] M. Alvarez, "Llego el momento de hablar sobre html." 2001. [En línea]. Disponible en: http://www.desarrolloweb.com/articulos/que-es-html.html

[40] F. Scholz, "Conceptos fundamentales de javascript," 2015. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Introducci%C3%B3n

[41] jQuery, "What is jquery," 2015. [En línea]. Disponible en: https://jquery.com/

[42] nquinlan, "Better random numbers for javascript." [En línea]. Disponible en: https://github.com/nquinlan/better-random-numbers-for-javascript-mirror

[43] Inpaint, "Inpaint photo restoration software," 2015. [En línea]. Disponible en: https://www.theinpaint.com/

[44] R. Martin, "Gimp inpainting plug-in," 2015. [En línea]. Disponible en: http://inpaintgimpplugin.github.io/

[45] EcuRed, "Modelo de prototipos," 2015. [En línea]. Disponible en: http://www.ecured.cu/index.php/Modelo_de_Prototipos