



Universitat de les
Illes Balears



Treball Final de Grau

Edición semiautomática online de imágenes con la ecuación de Poisson: aplicación a la eliminación de sombras

ÁNGEL TORRES RODRÍGUEZ

Tutor

José Luis Lisani Roca

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 31 de mayo de 2017

ÍNDICE GENERAL

Índice general	i
Acrónimos	iii
1 Introducción	1
2 Propósito y justificación	3
2.1 Propósito	3
2.2 Justificación	3
3 Objetivo	5
4 Conceptos Básicos	7
4.1 Conceptos Tecnológicos	7
4.1.1 <i>Domain Name System</i> (DNS)	7
4.1.2 <i>Uniform Resource Locator</i> (URL)	8
4.1.3 <i>Hypertext Transfer Protocol</i> (HTTP)	9
4.1.4 <i>Cookie</i>	12
4.1.5 <i>Hypertext Markup Language</i> (HTML)	13
4.1.6 Lenguajes de programación	14
4.1.7 <i>Common Gateway Interface</i> (CGI)	15
4.2 Conceptos matemáticos	16
4.2.1 Gradiente	16
4.2.2 Edición de imágenes basada en la ecuación de Poisson	20
4.2.3 Explicación	23
5 Desarrollo	29
5.1 Interesados y requisitos del sistema.	30
5.2 Comunicación cliente-servidor	32
5.2.1 <i>Asynchronous JavaScript And XML</i> (AJAX)	32
5.2.2 CGI	33
5.3 Almacenaje de los datos.	36
5.4 Funcionalidades de la aplicación.	36
5.4.1 Interfaz de sesión.	37
5.4.2 Interfaz de menú de usuario	38
5.4.3 Interfaz de funcionalidades para imágenes basadas en Poisson	39
6 Pruebas	55

6.1	Ejemplo de seamless stitching.	55
6.2	Ejemplos de eliminación de sombras.	56
6.3	Creación de gradientes	58
7	Conclusiones	59
A	Manual de Instalación	61
B	Manual de Usuario	65
	Bibliografía	71

ACRÓNIMOS

DNS *Domain Name System*

IP *Internet Protocol*

URL *Uniform Resource Locator*

HTTP *Hypertext Transfer Protocol*

WWW *World Wide Web*

HTML *Hypertext Markup Language*

AJAX *Asynchronous JavaScript And XML*

CGI *Common Gateway Interface*

MIME *Multipurpose Internet Mail Extensions*

API *Application Programming Interface*

PHP *Hypertext Preprocessor*

JSP *Java Server Pages*

ASP *Active Server Pages*

JSON *JavaScript Object Notation*

PNG *Portable Network Graphics*

INTRODUCCIÓN

En los últimos años, se han desarrollado muchas investigaciones y aplicaciones para realizar tareas de edición de imágenes debido al aumento del número de personas interesadas en modificarlas.

Las tareas de edición de imágenes engloban tanto los cambios globales (correcciones de color intensidad, aplicación de filtros, deformaciones...) como los cambios locales que se encuentran confinados a una selección.

Una de las tareas de edición de imágenes de mayor interés es la de composición de imágenes, que no es más que el proceso de componer dos imágenes para crear una nueva imagen. En este proceso de composición, el usuario dibuja un límite alrededor del objeto deseado en la imagen fuente para posteriormente clonarlo en la imagen destino. Actualmente, las técnicas más populares de composición de imágenes se basan en la modificación de los gradientes de la imagen, y posterior reconstrucción mediante la resolución de una ecuación en derivadas parciales llamada ecuación de Poisson. Esta técnica se utiliza en muchas aplicaciones como *seamless cloning* (insertar un pedazo de una imagen en otra imagen), *seamless stitching* (unir dos o más imágenes para crear un panorama, e.g. [1]), *inpainting* o eliminación de sombras.

El presente proyecto se enmarca en la aplicación de la ecuación de Poisson a la eliminación de sombras. En este sentido, un anterior proyecto dirigido por el mismo tutor [2], tenía por objeto la eliminación de sombras basada en la modificación de los valores de intensidad de la imagen. En el actual proyecto exploraremos cómo aplicar las técnicas de modificación de gradiente para lograr el mismo objetivo.

PROPÓSITO Y JUSTIFICACIÓN

2.1 Propósito

Obtener una aplicación web interactiva, de fácil uso y acceso, dotada de una serie de herramientas que permitan y faciliten el uso del método de edición de imágenes basada en la ecuación de Poisson y su aplicación para la eliminación de sombras.

2.2 Justificación

Se ha decidido desarrollar una aplicación web porque son aplicaciones que :

- Son independientes del sistema operativo del cliente.
- No requieren de ningún tipo de instalación por parte de los usuarios que harán uso de ella.
- No requieren de ningún tipo de distribución a los usuarios potenciales.
- Son fáciles de actualizar y mantener.

CAPÍTULO



3

OBJETIVO

Desarrollar una aplicación web interactiva cuya funcionalidad permita hacer edición de imágenes mediante la modificación selectiva de valores de gradiente y la posterior reconstrucción con la ecuación de Poisson.

CONCEPTOS BÁSICOS

En este capítulo se expondrán y se explicarán los conceptos básicos necesarios para el desarrollo del proyecto.

4.1 Conceptos Tecnológicos

4.1.1 DNS

Es un sistema de nivel de aplicación que surgió para permitir a las personas recordar con más facilidad los nombres de los servidores que se encuentran conectados a Internet.

Este sistema actúa a modo de soporte realizando la traducción del nombre de dominio a la dirección *Internet Protocol* (IP) destino, que es utilizada por la capa de red, para permitir la comunicación entre el origen y el destino. Por ejemplo, gracias a este sistema no es necesario recordar la dirección IP del servidor que nos proporciona el sitio web, únicamente sería necesario recordar el nombre de dominio, lo cual es más sencillo de cara al usuario que una dirección IP.

Obviamente, para hacer uso de este sistema de traducción se tuvo que definir el conjunto de nombres de dominio disponibles para los servidores y se tienen que almacenar el par nombre de dominio-dirección IP de cada una de las máquinas. El almacenaje que en un principio se produjo de manera centralizada en el archivo 'Hosts' de un mismo servidor, fue más adelante descentralizado debido al crecimiento masivo de la red.

El conjunto de nombres de dominio es conocido como el espacio de nombres de dominio y está diseñado de forma que los nombres se definen en una estructura de árbol invertida (la raíz se encuentra en la parte superior), tal y como muestra la figura 4.1, formada por 128 niveles.

Los nombres de dominio se definen como una secuencia de etiquetas (cada una de ellas identifica un nodo del árbol) separadas por puntos (.) los cuales se leen desde el

nodo hacia arriba hasta la raíz dentro del espacio de nombres de dominio. Se puede observar en la figura 4.1 un ejemplo de nombre de dominio (es.wikipedia.org.)

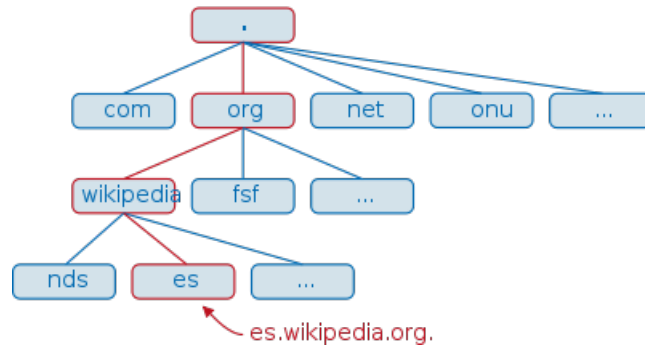


Figura 4.1: Espacio de nombres de dominio.

4.1.2 URL

El concepto de **URL** surgió para permitir el acceso de manera fácil y sencilla a cualquier tipo de información que se encuentra en Internet.

El **URL** define cuatro cosas:

1. El protocolo. Programa cliente/servidor utilizado para recuperar el documento. **HTTP** sería un ejemplo de programa utilizado para recuperar documentos.
2. La estación . Computadora en la que se encuentra la información a la que se desea acceder.
3. El puerto. Este campo es opcional y se utiliza para especificar el puerto del servidor.
4. El camino. Nombre del archivo donde se encuentra la información. Puede contener separadores (/) que separan los directorios de los subdirectorios y archivos.

El formato es el siguiente:

Protocolo://Estación:Puerto/Camino

Un ejemplo sería:

http://www.marca.com

En este ejemplo se pueden apreciar varias cosas:

1. El protocolo utilizado ha sido **HTTP**.
2. La estación se ha identificado con el nombre de dominio marca.com

3. No se ha incluido el puerto. Al hacer uso del protocolo **HTTP** el navegador ya interpreta que el puerto destino es el puerto 80, ya que es el especificado en el estándar para el uso de este protocolo.
4. No se ha introducido el camino. Los servidores suelen tener configurado un documento por defecto para entregar si no se les especifica uno en concreto en la petición.

La figura 4.2 muestra la página web obtenida al realizar la petición **HTTP** haciendo uso de la **URL** indicada en el ejemplo.

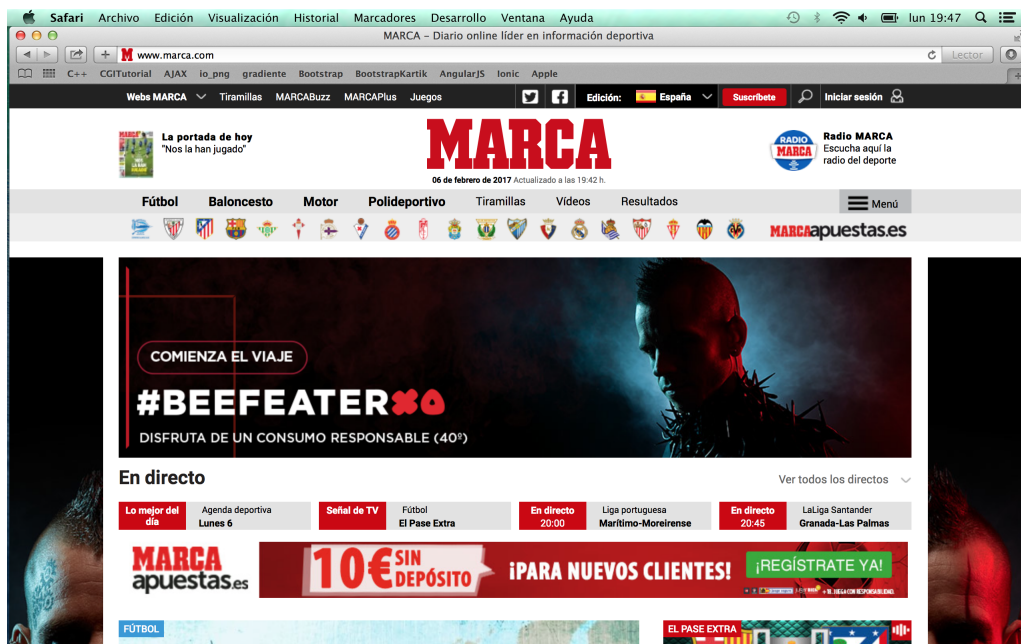


Figura 4.2: Ejemplo URL

4.1.3 **HTTP**

Es un protocolo de comunicación que permite la transferencia de información en la *World Wide Web* (**WWW**).

El funcionamiento de **HTTP** es simple, el cliente realiza una petición al puerto 80 de un servidor y este le responde tal y como muestra la figura 4.3. Para ello **HTTP** define el formato de los mensajes de petición y respuesta.

Los mensajes de petición y respuesta que utiliza este protocolo, los cuales se muestran en la figura 4.4, están formados por los siguientes bloques:

1. Línea de petición (figura 4.5). Primera línea que se encuentra únicamente en todo mensaje de petición y que está formada por los siguientes campos:

- Tipo de petición. Define el método utilizado en la petición.

Los tipos de petición **HTTP** utilizados más frecuentemente son los siguientes:

- GET. Solicita un documento al servidor.
- HEAD. Solicita información sobre un documento pero no el documento en si.
- POST. Envía información del cliente al servidor.
- PUT. Envía un documento del servidor al cliente.

- **URL**. Se ha definido el concepto previamente en este documento.
- Versión. Indica la versión, valga la redundancia, del protocolo **HTTP** utilizado.

2. Línea de estado (figura 4.5). Primera línea que se encuentra únicamente en todo mensaje de respuesta y que esta formada por los siguientes campos:

- Versión. Indica la versión, valga la redundancia, del protocolo **HTTP** utilizado.
- Código de estado. Código que consta de tres dígitos.

Los tipos de código de estado son los siguientes:

- 1XX. Códigos con mensajes informativos.
- 2XX. Códigos de petición exitosa.
- 3XX. Códigos de redireccionamiento.
- 4XX. Códigos de error del cliente.
- 5XX. Códigos de error del servidor.

- Frase de estado. Este mensaje se utiliza para indicar el estado en formato de texto.

3. Cabecera. Consta de una o más líneas de cabecera cada una de las cuales siguen la siguiente estructura:

Nombre de cabecera: valor

Las líneas de cabecera permiten intercambiar información adicional entre el cliente y el servidor.

4. Cuerpo. Contiene el documento a enviar o recibir.

Cabe decir que **HTTP** es un protocolo sin estado, no guarda ningún tipo de información sobre conexiones anteriores a los servidores. Debido a que el desarrollo de aplicaciones web necesita frecuentemente mantener el estado, surgieron las *cookies*.

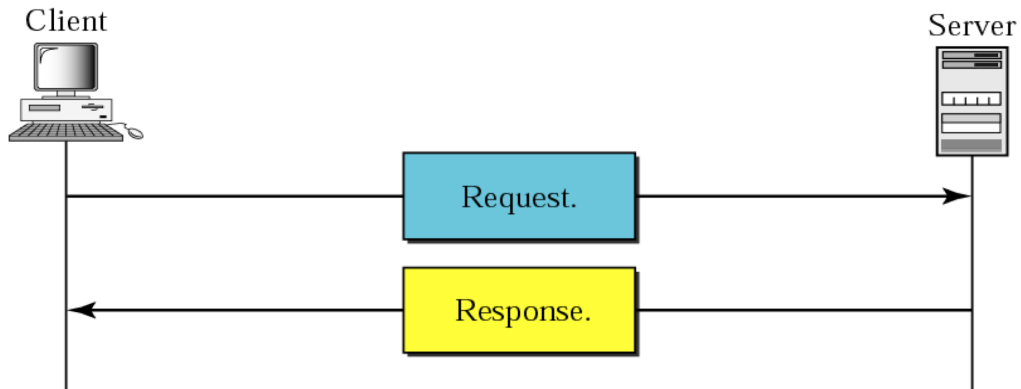


Figura 4.3: Comunicación HTTP

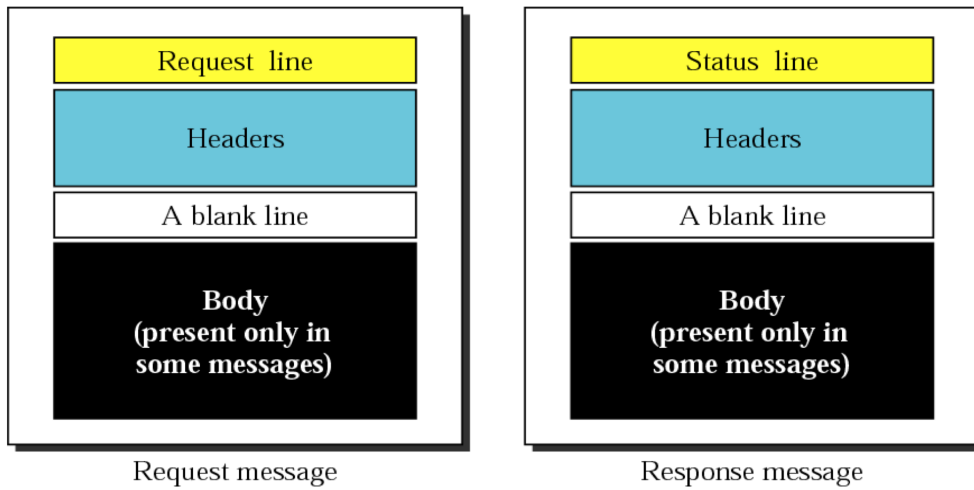


Figura 4.4: Formatos de los mensajes

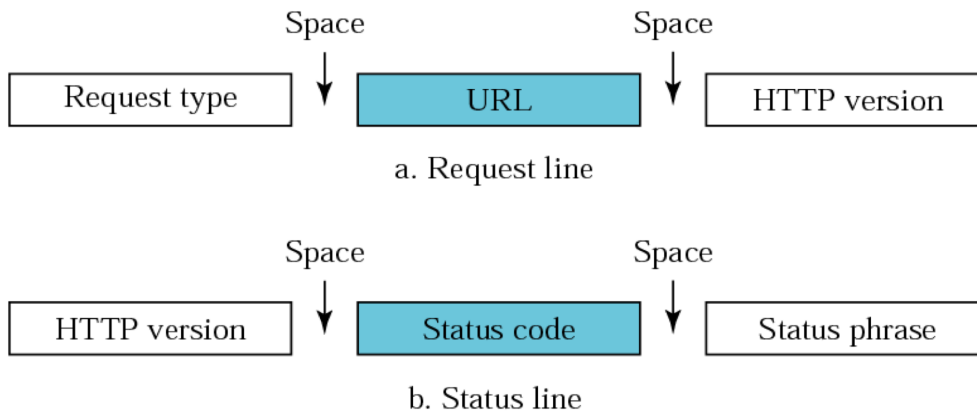


Figura 4.5: Líneas de petición/respuesta

4.1.4 Cookie

En una arquitectura cliente-servidor existen dos partes involucradas: el cliente y el servidor. El servidor es el encargado de procesar y responder a las peticiones que le llegan de los clientes (navegadores web).

Tradicionalmente los servidores no tenían ninguna manera de reconocer a los clientes que les realizaban las peticiones. No podían saber al recibir una petición de un cliente si este ya le había realizado una petición para solicitarle un recurso o servicio con anterioridad. De ahí que surgiera más adelante el concepto de *cookie*.

Una *cookie* es un trozo de información creada por un servidor y que contiene información del cliente. Por ejemplo: un nombre, un identificador de usuario, el contenido del carrito de compra, información personal, etc.

El proceso que se sigue para su utilización es el siguiente:

1. El servidor genera la *cookie* con la información que el considere necesaria y se la transmite al cliente.
2. El cliente recibe la *cookie* y la almacena. Este no puede modificar el contenido de la *cookie* almacenada pero si eliminarla.
3. El cliente incluye la *cookie* en todas las peticiones que realice al servidor que la generó.

Aunque el objetivo principal de las *cookies* es proporcionar información de estado entre peticiones **HTTP**, también permiten proporcionar ciertas funciones adicionales. Algunas de las funciones adicionales proporcionadas son las siguientes:

- Carrito de compras virtual. La utilización de *cookies* para compras virtuales permite al cliente seleccionar los productos que serán añadidos o eliminados del carrito para posteriormente realizar la compra de los elementos que se encuentren en el mismo. Sin el uso de *cookies* el cliente tendría que comprar cada producto individualmente.
- Identificación y autenticación del cliente. Las *cookies* pueden ser utilizadas como mecanismo de reconocimiento de los usuarios. De esta forma se evita que el usuario deba introducir las credenciales (usuario y contraseña) en cada petición que realice al sitio web.
- Personalización de las preferencias del usuario. Las *cookies* pueden ser usadas para la personalización de sitios web según las preferencias del usuario. De esta forma se mantendrán las preferencias del usuario en las posteriores peticiones al servidor.
- Seguimiento de clientes. Las *cookies* permiten registrar todas las peticiones realizadas por un cliente. Con este registro de peticiones el propietario del sitio web puede averiguar los intereses del cliente y actuar en consecuencia.
- Perfiles de usuario. Se pueden obtener perfiles de usuario muy amplios haciendo uso de las llamadas *cookies* de terceros mediante el rastreo de los usuarios a través de varios sitios web.

- **Análisis de un sitio web.** El propietario de un sitio web puede realizar análisis de las actividades de los usuarios en el mismo y modificarlo en función del análisis realizado.

Existen diversos tipos de *cookies* que pueden ser clasificadas siguiendo diferentes criterios. El criterio más habitual es clasificar en base al tiempo que el navegador almacena la *cookie*. En base a este criterio las *cookies* se pueden clasificar en los siguientes tipos:

- *Cookies* de sesión. Son aquellas utilizadas durante una sesión del usuario. El navegador las elimina una vez se ha cerrado la sesión.
- *Cookies* persistentes. Son *cookies* que permanecen almacenadas en la memoria del ordenador una vez se ha cerrado el navegador. El navegador elimina estas *cookies* cuando expira la fecha de validez que introdujo el servidor en las mismas.

4.1.5 HTML

Es un estándar a cargo del **WWW** que hace referencia al lenguaje de marcado para la elaboración de páginas web.

Las etiquetas que definen toda página **HTML** son las siguientes:

- Las etiquetas `<html> </html>` indican que el contenido se debe interpretar con el lenguaje **HTML**.
- La cabecera viene definida por las etiquetas `<head> </head>`, dentro de las cuales se encuentra el lenguaje marcador que describe la página.
- El cuerpo viene definido por las etiquetas `<body> </body>`, dentro de las cuales se encuentra el lenguaje marcador que define el contenido de la página.

En la figura 4.6 se muestra un ejemplo sencillo de página **HTML** en el cual se puede apreciar el lenguaje marcador que define toda web así como dos tipos más de etiquetas como son las de título (`<title> </title>`) y encabezado (`<h1> </h1>`).

```
<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Hello world!!!!</h1>
  </body>
</html>
```

Figura 4.6: HTML Hello world

Este tipo de páginas son interpretadas por los navegadores y posteriormente visualizadas por los mismos.

En la figura 4.7 se puede observar la interpretación del ejemplo anterior realizada por el navegador google chrome.

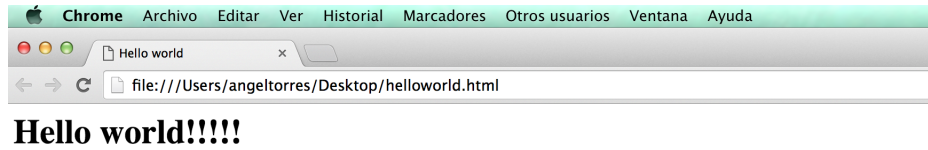


Figura 4.7: Hello world

El conjunto de etiquetas **HTML** se puede encontrar en la siguiente **URL**:

<https://www.w3schools.com/TAGs/>

4.1.6 Lenguajes de programación

Lenguaje informático diseñado para expresar órdenes y instrucciones las cuales serán llevadas a cabo por una computadora. Esta compuesto por símbolos, reglas semánticas y sintácticas que definen la estructura del lenguaje.

Los lenguajes de programación surgieron debido a la dificultad de uso del lenguaje específico conocido como código máquina, el cual es necesario para que una computadora entienda las instrucciones que se le transmiten. Estos lenguajes actúan a modo de traductores entre un lenguaje de palabras o abstracción de palabras y el lenguaje máquina (0 y 1) .

Pueden ser clasificados en dos tipos: lenguajes de bajo y alto nivel.

- Lenguajes de bajo nivel. Lenguajes de programación que ejercen un control directo sobre el *hardware* y están condicionados por la estructura física que soportan.
- Lenguajes de alto nivel. Lenguajes de programación independientes de la arquitectura del *hardware* y cuya característica principal consiste en una estructura sintáctica y semántica legible.

Dejando a un lado los lenguajes de programación de bajo nivel, los lenguajes de programación de alto nivel pueden ser clasificados en dos tipos: lenguajes compilados y lenguajes interpretados.

1. Lenguajes de programación interpretados.

Son aquellos que no requieren de un compilador para ser ejecutados sino de un intérprete. El intérprete actúa de forma similar a un compilador a excepción de que no genera un ejecutable previo a la ejecución del código, lo ejecuta directamente.

Estos lenguajes suelen ser más lentos que los lenguajes compilados debido a esta sobrecarga de traducción en tiempo de ejecución. Algunos de los lenguajes interpretados más populares son JavaScript y Python.

JavaScript

JavaScript es un lenguaje de programación interpretado orientado a objetos que se ejecuta en el lado del cliente permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

Tradicionalmente se utilizaba en páginas web **HTML** para realizar operaciones sencillas en el lado del cliente, sin acceso a funciones del servidor. Actualmente también se suele utilizar para enviar y recibir información junto con la ayuda de otras tecnologías como **AJAX**.

Python

Python es un lenguaje de programación interpretado, multiplataforma, de tipado dinámico y multiparadigma.

- Multiplataforma. Puede ser interpretado por diversos sistemas operativos tales como Linux, Windows, Mac OS, entre otros.
- Multiparadigma. Acepta diferentes paradigmas de programación, tales como orientación a objetos, programación imperativa y funcional.
- Tipado dinámico. No requiere que las variables del lenguaje sean definidas según el tipo de datos asignado, se auto-asignan en tiempo de ejecución, según el valor declarado.

2. Lenguajes de programación compilados.

Son aquellos que requieren de un compilador para ser ejecutados. El compilador genera un ejecutable, que contiene el código máquina fruto de la traducción del código fuente, previo a la ejecución del código. C++ sería un ejemplo claro de lenguaje de programación compilado.

C++

C++ es un lenguaje de programación compilado que fue creado a modo de extensión del lenguaje popular C para poder permitir la manipulación de objetos. Además de la programación orientada a objetos, C++ permite otros métodos programación como son la programación genérica y estructurada.

4.1.7 CGI

Tecnología de la **WWW** que permite a un cliente solicitar datos a un programa que se ejecuta en un servidor web.

Características de los **CGIs**:

4. CONCEPTOS BÁSICOS

- Definen la forma en la que se realiza la interacción entre el servidor y programas externos.
- Se ejecuta como un hijo del proceso servidor Web.
- Generan contenido web, el cual será identificado con un tipo *Multipurpose Internet Mail Extensions* (MIME), que será enviado al cliente.

Ventajas y desventajas del uso de CGI:

- Ventajas
 1. Independiente del lenguaje.
 2. Interfaz sencilla. No es necesario hacer uso de una librería o *Application Programming Interface* (API) específicos.
- Desventajas
 1. Cada petición HTTP crea un nuevo proceso en el servidor y eso provoca un exceso de uso en los recursos del sistema.
 2. Sobrecarga en la interpretación del script debido a que el código interpretado consume más tiempo que el compilado.

Los CGI permiten crear de manera sencilla contenido dinámico en un sitio web. Existen alternativas a los CGI para poder obtener contenido dinámico en un sitio web como son los Java Servlets y los códigos de scripts embebidos en páginas HTML como *Hypertext Preprocessor* (PHP), *Java Server Pages* (JSP) y *Active Server Pages* (ASP).

4.2 Conceptos matemáticos

4.2.1 Gradiente

Se define el gradiente G de una imagen monocroma $f(x, y)$ en el punto (x, y) como el vector de dos dimensiones:

$$G(f(x, y)) = (\Delta_x, \Delta_y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \quad (4.1)$$

$$\Delta_x = \frac{f(x + dx, y) - f(x, y)}{dx} \quad (4.2)$$

$$\Delta_y = \frac{f(x, dy + y) - f(x, y)}{dy} \quad (4.3)$$

El vector gradiente G apunta en la dirección de máximo cambio de f en el punto (x, y) . Un valor alto del gradiente en un punto de la imagen denota una variación local de la intensidad. Dado que las variaciones locales en la intensidad de la imagen (el paso de nivel oscuro a uno brillante, o viceversa) definen los contornos, o bordes, de los objetos presentes en la escena, la mayoría de técnicas para la detección de estos contornos utilizan métodos basados en el gradiente.

Para la detección de bordes se puede hacer uso de la magnitud del vector gradiente el cual denotaremos como $Gmag$:

$$Gmag = \sqrt{\Delta_x^2 + \Delta_y^2} \quad (4.4)$$

Se puede considerar dx y dy en términos del número de píxeles entre dos puntos al tratarse de imágenes digitales (discretas). Si el píxel donde vamos a calcular el gradiente tiene coordenadas (i, j) y $dx = dy = 1$ tenemos que:

$$\Delta_x = f(i + 1, j) - f(i, j) \quad (4.5)$$

$$\Delta_y = f(i, j + 1) - f(i, j) \quad (4.6)$$

Con lo cual se puede apreciar que para calcular el vector gradiente de un determinado píxel el problema se simplifica al tratarse de una imagen digital. Para calcular el vector gradiente de la forma más simple bastaría con:

- Realizar una diferencia entre el valor del posterior píxel adyacente horizontalmente y el valor del píxel del cual se quiere calcular el gradiente.
- Realizar una diferencia entre el valor del posterior píxel adyacente verticalmente y el valor del píxel del cual se quiere calcular el gradiente.

Habitualmente se suelen utilizar máscaras para el cálculo del gradiente ya que presentan beneficios a la hora de calcularlo. El gradiente de la fila G_f y de columna G_c en cada punto se obtienen mediante la convolución de la imagen con las máscaras H_f y H_c , esto es:

$$G_f(i, j) = F(i, j) \otimes H_f(i, j) \quad (4.7)$$

$$G_c(i, j) = F(i, j) \otimes H_c(i, j) \quad (4.8)$$

Algunos de los operadores más populares son el de Roberts, Prewitt y Sobel.

- Operador de Roberts. Este operador responde bien a los bordes diagonales pero al ser extremadamente sensible al ruido proporciona una calidad pobre en la detección.

Dirección horizontal.

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad (4.9)$$

Dirección vertical.

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (4.10)$$

- Operador de Prewitt. Involucra a los vecinos de filas/columnas adyacentes para proporcionar mayor inmunidad al ruido

Dirección horizontal.

$$\frac{1}{3} \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad (4.11)$$

Dirección vertical.

$$\frac{1}{3} \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad (4.12)$$

- Operador de Sobel. Este operador es más sensible a los bordes diagonales que el de Prewitt

Dirección horizontal.

$$\frac{1}{4} \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad (4.13)$$

Dirección vertical.

$$\frac{1}{4} \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad (4.14)$$

Una manera alternativa de calcular el gradiente es utilizando las siguientes fórmulas, que resultan de calcular el gradiente de la interpolación bilineal de la image f (**gradiente bilineal**):

$$G_f(i+0,5, j+0,5) = \frac{(f(i+1, j) + f(i+1, j+1)) - (f(i, j) + f(i, j+1))}{2} \quad (4.15)$$

$$G_f(i+0,5, j+0,5) = \frac{(f(i, j+1) + f(i+1, j+1)) - (f(i, j) + f(i+1, j))}{2}$$

Esta última opción de cálculo proporciona resultados más estables (respecto a variaciones espaciales y a ruido) que los operadores anteriores y por eso es la opción utilizada en nuestros códigos.

Las siguientes imágenes (figura 4.8) ilustran los contornos resultantes de aplicar los diferentes métodos de cálculo de gradiente a la misma imagen. Las imágenes mostradas son binarias (blanco/negro indica presencia/ausencia de contorno) y utilizan un umbral (*threshold*) prefijado para decidir la existencia del contorno: si $G_{mag}(i, j) > \text{threshold}$ entonces el pixel (i, j) pertenece a un contorno. Los resultados de Roberts, Prewitt y Sobel se obtuvieron con la función *edge* de Matlab, que muestra los puntos donde hay un máximo del gradiente. El resultado del gradiente bilineal se obtuvo con nuestro código y muestra los píxeles cuya magnitud de gradiente es superior a 20.



Figura 4.8: De izquierda a derecha y de arriba a abajo: imagen original y resultados del cálculo de gradiente con los operadores de Sobel, Prewitt, Roberts y gradiente bilineal.

4.2.2 Edición de imágenes basada en la ecuación de Poisson

La edición de imágenes con la ecuación de Poisson es una técnica que permite combinar dos imágenes de forma automática.

Antes de entrar en la idea a la que se reduce este algoritmo, vamos a definir lo que nos gustaría lograr cuando combinamos dos imágenes juntas. Llamaremos a la imagen que estamos cambiando imagen A y a la imagen que estamos cortando y pegando en A, imagen B tal y como muestra la figura 4.9.

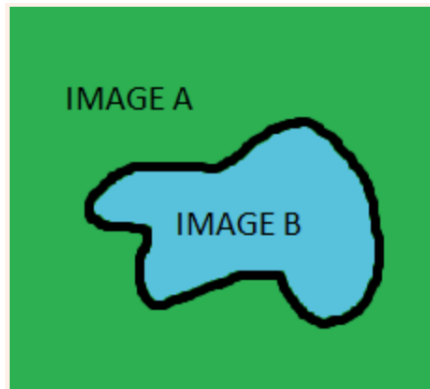


Figura 4.9: Imágenes

Con el objetivo de realizar una explicación más intuitiva se seleccionaron, a modo de ejemplo, las imágenes que se muestran en la figura 4.10 y 4.11. La imagen de la figura 4.10 se corresponderá con la imagen A, mientras que un trozo de la imagen 4.11 se corresponderá con la imagen B.



Figura 4.10: Imagen A

En la figura 4.12 se muestra el pegado de la imagen B en la imagen A. Lo ideal sería que se modificaran los colores de la imagen B, al pegarla en la imagen A, manteniendo algunos detalles de la misma como son todos los bordes, esquinas, transiciones... en la imagen B.



Figura 4.11: Imagen B



Figura 4.12: Unión imagen A y B

Estos detalles son obtenidos mediante métodos basados en el cálculo del gradiente, el cual permite describir como cambian los pixeles de la imagen con respecto a los pixeles alrededor de ellos.

El objetivo de la edición de imágenes usando la ecuación de Poisson es permitir realizar un pequeña modificación de la información absoluta de la imagen B (colores) pero conservar la información relativa (imagen de gradiente) de la misma después de pegarla.

4. CONCEPTOS BÁSICOS

Para llevar a cabo la modificación lo que se hace es fijar el valor de los píxeles en el límite de la imagen B con un valor igual al valor de los píxeles de la imagen A, donde reside B, y luego resolver para el resto de píxeles en el interior de la imagen B con el objetivo de que estos preserven el gradiente original de la imagen.

En lo que concierne al ejemplo que se está explicando, se puede observar en la figura 4.13 el límite de la imagen B y en la figura 4.14 el resultado obtenido al aplicarle el método de reconstrucción basado en la ecuación de Poisson a la imagen.

Al comparar las imágenes 4.12 y 4.14 se puede apreciar de manera muy clara como el fondo que envuelve a la leona (imagen B) ha adoptado un color idéntico al del entorno de la sabana (imagen A). De esta forma se ha conseguido combinar las dos imágenes y dar una cierta impresión de que se trata de una sola única imagen.

Cuanto más uniforme sea la zona que rodea al objeto B y la zona de pegado en A, la combinación de las dos imágenes haciendo uso de la ecuación de Poisson será más efectiva.

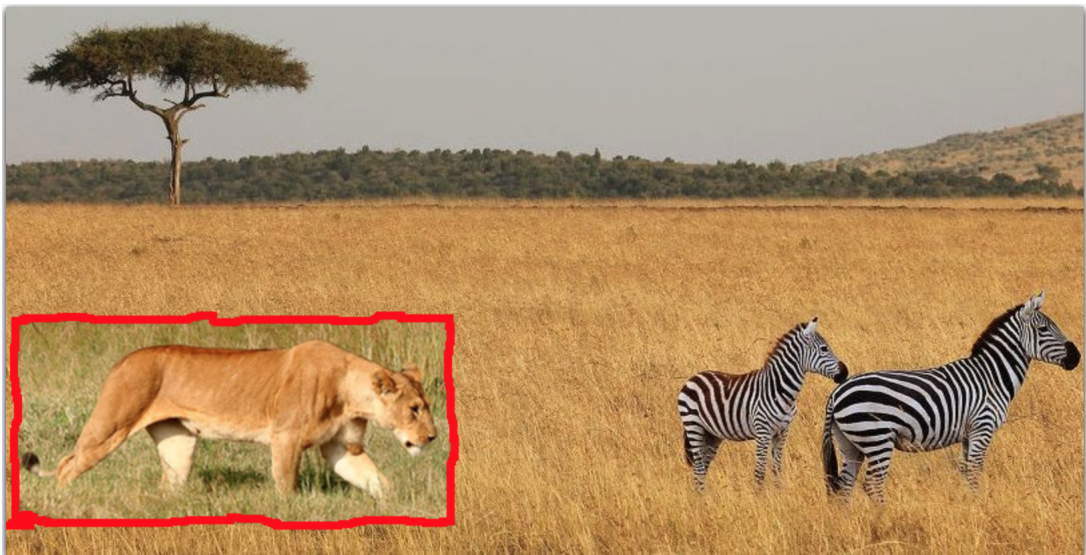


Figura 4.13: Limite seleccionado en la imagen



Figura 4.14: Imagen resultado

4.2.3 Explicación

En esta sección se pretende detallar como se lleva a cabo la interpolación de imágenes haciendo uso de un campo vectorial de orientación. Como es suficiente resolver el problema de interpolación para cada componente de color por separado, consideramos solamente funciones de imagen escalar. La sección resume el contenido del artículo de Pérez et al. (*Poisson Image Editing*, 2003 [6]) que sentó las bases de las actuales técnicas de edición basadas en la ecuación de Poisson.

La figura 4.15 ilustra las notaciones empleadas:

- Sea S , un subconjunto de \mathbb{R}^2 , el dominio de definición de imagen.
- Sea Ω un subconjunto cerrado de S con límite $\partial\Omega$.
- Sea f^* una función escalar conocida definida sobre S menos el interior de Ω .
- Sea f una función escalar definida sobre el interior de Ω .
- Sea \mathbf{v} un campo vectorial definido sobre Ω (que puede ser, o no, el gradiente de una cierta función escalar g).

La interpolación mas sencilla f de f^* sobre Ω es la solución obtenida del siguiente problema de minimización:

$$\min_f \int_{\Omega} |\nabla f|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (4.16)$$

donde $\nabla = [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}]$ es el operador de gradiente. El minimizador debe satisfacer la ecuación de Euler-Lagrange:

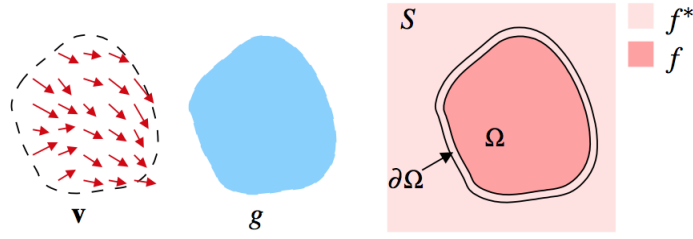


Figura 4.15: Notaciones de interpolación guiada.

$$\Delta f = 0 \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (4.17)$$

donde $\Delta. = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ es el operador Laplaciano. Esta ecuación es la ecuación de Laplace con las condiciones en el límite de Dirichlet. Para las aplicaciones de edición de imágenes, este método simple produce un interpolante borroso, insatisfactorio y esto puede superarse de varias maneras. Aquí se propone modificar el problema introduciendo restricciones adicionales como es la utilización de un campo de orientación tal y como se explica a continuación.

Un campo de orientación es un campo vectorial \mathbf{v} utilizado en una versión extendida del problema de minimización anterior

$$\min_f \int_{\Omega} |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (4.18)$$

cuya solución es la solución única de la ecuación de Poisson con las condiciones en el límite de Dirichlet.

$$\Delta f = \text{div} \mathbf{v} \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (4.19)$$

donde $\text{div} \mathbf{v}$ es la divergencia de $\mathbf{v}=(u,v)$. Este es el mecanismo de la edición de imágenes por Poisson: tres ecuaciones de Poisson de la forma anterior son resueltas independientemente para los tres canales del espacio de colores elegido (RGB).

Cuando el campo de orientación es conservativo (es decir, \mathbf{v} es el gradiente de una cierta función g), una forma de entender lo que hace la interpolación por Poisson es definir la función de corrección f' en Ω tal que $f = g + f'$. La ecuación de Poisson (4.19) se convierte en la siguiente ecuación de Laplace:

$$\Delta f' = 0 \text{ over } \Omega \text{ with } f'|_{\partial\Omega} = (f^* - g)|_{\partial\Omega} \quad (4.20)$$

Para el caso de imágenes digitales, el problema variacional (4.18) y la ecuación de Poisson asociada con las condiciones de frontera de Dirichlet (4.19) pueden discretizarse y resolverse de varias maneras. Para no perder la generalidad, haremos uso de las mismas notaciones para los objetos continuos y sus contrapartes discretas:

- S, Ω ahora se convierten en conjuntos de puntos en una rejilla discreta infinita donde S puede incluir todos los píxeles de una imagen o solo un subconjunto de ellos.
- Para cada píxel p en S , N_p representará el conjunto de sus 4 vecinos conectados que están en S , y $\langle p, q \rangle$ denotará un par de píxeles tal que $q \in N_p$.
- El límite de Ω es ahora $\partial\Omega = \{p \in S \setminus \Omega : N_p \cap \Omega \neq \emptyset\}$.
- Sea f_p el valor de f en p .

La tarea consiste en calcular el conjunto de intensidades $f|_{\Omega} = \{f_p, p \in \Omega\}$.

Para las condiciones de frontera de Dirichlet definidas en un límite de forma arbitraria, lo mejor es discretizar el problema variacional (4.18) directamente, en lugar de la ecuación de Poisson (4.19). La discretización de diferencias finitas de (4.18) produce el siguiente problema de optimización:

$$\min_{f|_{\Omega}} \sum_{\langle p, q \rangle \cap \Omega \neq \emptyset} (f_p - f_q - v_{pq})^2 \text{ with } f_p = f_p^* \text{ for all } p \in \partial\Omega \quad (4.21)$$

donde v_{pq} es la proyección de $\mathbf{v}(\frac{p+q}{2})$ orientada al borde $[p, q]$. Esta solución satisface la siguiente ecuación lineal:

$$\text{for all } p \in \Omega, |N_p|f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq} \quad (4.22)$$

La elección básica para el campo de orientación \mathbf{v} es un campo de gradiente tomado directamente de una imagen fuente. Denotada por g esta imagen fuente, la interpolación se realiza bajo la guía de

$$\mathbf{v} = \nabla g \quad (4.23)$$

y la ecuación de Poisson (4.19) ahora se lee

$$\Delta f = \Delta g \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (4.24)$$

En cuanto a la implementación numérica, la especificación continua (4.23) se discretiza en

$$\text{for all } \langle p, q \rangle, v_{pq} = g_p - g_q \quad (4.25)$$

Screened Poisson Equation. La implementación discreta de la ecuación de Poisson descrita en los párrafos anteriores es lenta, pues implica la resolución del sistema de ecuaciones (4.22). Una alternativa mucho más rápida, propuesta por Petro y Sbert en [7] es definir el vector \mathbf{v} no únicamente sobre el dominio Ω sino globalmente, sobre todo el dominio S de la imagen f^* , de la siguiente manera

$$\mathbf{v} = \begin{cases} \nabla g & \text{in } \Omega \\ \nabla f^* & \text{in } S - \Omega \end{cases} \quad (4.26)$$

Y el nuevo problema de minimización, en lugar de (4.18), es

$$\min_f \int \int_S |\nabla f - \mathbf{v}|^2 \quad (4.27)$$

que tiene una solución global f definida sobre S que satisface

$$\begin{cases} \Delta f = \operatorname{div} \mathbf{v} & \text{in } S \\ \frac{\partial f}{\partial \mathbf{n}} = 0 & \text{in } \partial S \end{cases} \quad (4.28)$$

Esta ecuación se conoce como **Screened Poisson Equation** y admite una solución en el dominio de la transformada de Fourier, que en su versión discreta se escribe como:

$$\hat{f}_{mn} = \frac{\frac{2\pi im}{J} \hat{v}_{xmn} + \frac{2\pi in}{L} \hat{v}_{ymn}}{\left(\frac{2\pi m}{J}\right)^2 + \left(\frac{2\pi n}{L}\right)^2} \quad (4.29)$$

donde el símbolo $\hat{\cdot}$ denota la transformada de Fourier discreta, $J \times L$ es el tamaño de la imagen f^* , (m, n) son los índices de la transformada de Fourier discreta y (v_x, v_y) son las componentes horizontal y vertical de \mathbf{v} . Esta solución está definida para $(m, n) \neq (0, 0)$. Para el caso $(m, n) = (0, 0)$ se impone la solución $\hat{f}_{00} = 0$.

El algoritmo final para la modificación de los gradientes de una imagen es:

1. Dada una imagen de partida f^* , seleccionar un conjunto de píxeles (veremos en las siguientes secciones como la herramienta web diseñada permite hacer esta selección de manera sencilla).
2. Imponer un valor al gradiente de estos píxeles.
3. Resolver (4.29).
4. La solución final f se obtiene haciendo la transformada de Fourier inversa y asignando linealmente los valores obtenidos en el rango $[0, 255]$.

Tratamiento de las texturas. Como veremos en la sección experimental, una aplicación de la edición de imágenes basada en la ecuación de Poisson es la eliminación de gradientes de una imagen. Esto se consigue aplicando el algoritmo anterior a un conjunto de píxeles de la imagen e imponiendo que el gradiente en estos puntos valga cero. Un ejemplo del tipo de resultado obtenido se muestra en la figura 4.16-izquierda. El resultado muestra como el algoritmo es capaz de eliminar la sombra de la imagen, pero observamos que las texturas de la zona en que el gradiente se ha puesto a cero han quedado totalmente difuminadas. Hemos diseñado la siguiente estrategia (post-processing) para prevenir en lo posible la eliminación de estas texturas.

Para cada canal de la imagen y para cada píxel cuyo gradiente ha sido puesto a cero:

1. Se toma una subimagen de tamaño 20×20 píxeles (patch) centrada en el píxel.
2. Se estima el valor medio del patch en la imagen original (antes de la reconstrucción de Poisson).
3. Se calcula la diferencia entre el valor medio del patch y el valor original del píxel (antes de la reconstrucción de Poisson).
4. Si esta diferencia es inferior a un parámetro T (e.g. 20), se añade la diferencia al valor actual del píxel (después de la reconstrucción de Poisson).

El resultado de aplicar este post-processing se muestra en las figura 4.16-centro y derecha, para diferentes valores del parámetro. Observamos como los resultados mejoran el obtenido sin post-procesamiento. Más resultados de este tipo se muestran en la sección de resultados.

Creación de gradientes. Otra posibilidad del algoritmo para la modificación de gradientes es la de imponer un valor de gradiente distinto de cero a un grupo de píxeles de la imagen. Esto obliga también a definir una dirección para el vector gradiente. Se ha desarrollado el siguiente algoritmo para la definición de la orientación del gradiente dada una selección de píxeles:

1. Si el conjunto de píxeles seleccionados se halla, aproximadamente, sobre una segmento, entonces el vector gradiente se orienta en la dirección perpendicular al segmento.
2. En caso contrario, se calcula el baricentro (centro de masas) del grupo de píxeles y el gradiente de cada píxel se orienta en la dirección del baricentro.

La herramienta web desarrollada permite elegir la magnitud del gradiente y la orientación (positiva o negativa) del vector en la dirección estimada. La figura 4.17 muestra algunos ejemplos de las direcciones de gradiente estimadas mediante este algoritmo.

4. CONCEPTOS BÁSICOS



Figura 4.16: De izquierda a derecha y de arriba a abajo: imagen original, selección de píxeles (en negro) cuyos gradientes serán puestos a cero y resultados de la reconstrucción de Poisson para diferentes valores del parámetro T (0, 20 y 40). Para $T = 0$ el resultado obtenido es equivalente a no hacer ningún post-procesamiento.



Figura 4.17: Direcciones de gradiente estimadas.

DESARROLLO

En este capítulo se detallará paso a paso el proceso que se ha seguido para desarrollar la aplicación web del proyecto.

El primer punto a destacar es que la aplicación web seguirá el paradigma cliente-servidor. Con lo cual se dispondrá de un servidor web encargado de proporcionar unos servicios y recursos a los clientes que le soliciten los mismos tal y como se puede apreciar en la figura 5.1 .

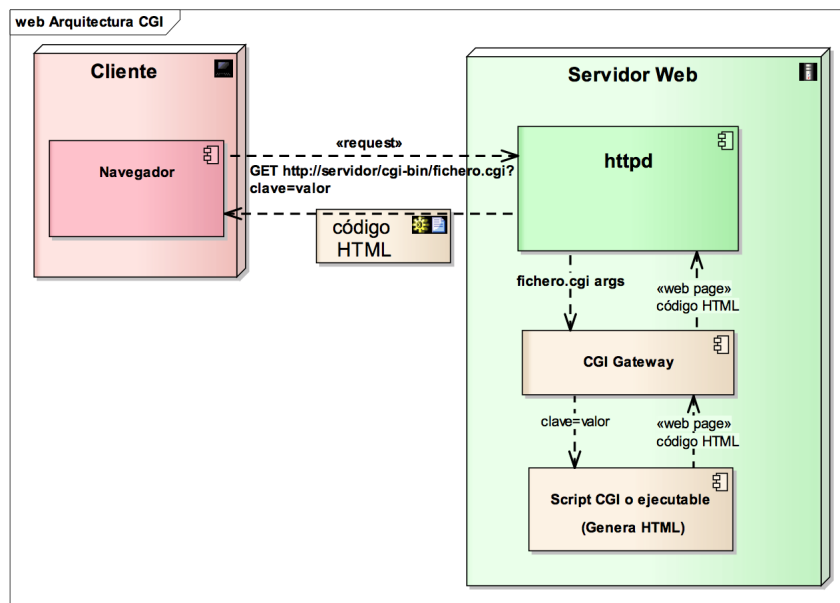


Figura 5.1: Arquitectura cliente-servidor

5.1 Interesados y requisitos del sistema.

El primer paso, en todo desarrollo, de cualquier tipo de aplicación es definir los interesados del sistema, que no son más que las partes involucradas que tienen una influencia directa en los requisitos del sistema a desarrollar. El papel de los mismos es importante ya que sus requisitos son los que definirán, en mayor medida o en su totalidad, la funcionalidad del sistema.

Los interesados del proyecto son los siguientes:

- Usuarios finales. Usuarios que harán uso de la aplicación.

Para el caso de esta aplicación en concreto solo tendríamos un interesado, que es el usuario final que hará uso de la misma.

Una vez definidos los interesados del sistema, es importante definir los requisitos funcionales y no funcionales del sistema. Los requisitos funcionales son los que vienen determinados por los interesados del sistema, mientras que los requisitos no funcionales son requisitos que vienen determinados por las características de funcionamiento del sistema.

En el caso de esta aplicación tenemos como requisitos funcionales (RF)

- Requisitos de los Usuarios Finales (UF).
 1. RF_UF01. Un usuario tiene que poder registrarse en la aplicación.
 - La información necesaria para registrarse será: nombre de usuario y contraseña
 2. RF_UF02. Un usuario tiene que disponer de un menú que le permita acceder a la aplicación de Poisson así como a otras aplicaciones de procesamiento de imágenes.
 3. RF_UF03. Un usuario tiene que poder subir imágenes a la aplicación desde su equipo.
 4. RF_UF04. Un usuario tiene que poder modificar las imágenes.
 - Tiene que poder dibujar con colores sobre ella.
 - Tiene que poder dibujar con varios grosores de línea.
 5. RF_UF05. Un usuario tiene que poder aplicar técnicas de procesamiento de imágenes sobre las mismas.
 - Tiene que poder obtener la imagen de gradiente.
 - Tiene que poder obtener la imagen resultado de la aplicación del algoritmo de Poisson.
 6. RF_UF06. Un usuario tiene que poder descargar las imágenes a su equipo.

y como requisitos no funcionales(RNF)

1. RNF01 Disponibilidad. La aplicación tiene que estar disponible las 24 horas del día.

5.1. Interesados y requisitos del sistema.

2. RNFx02 Accesibilidad. La aplicación tiene que ser accesible a través de cualquier navegador web.
3. RNFx03 Escalabilidad. La aplicación debe ser ampliable y configurable para adaptarse a cambios futuros.
4. RNFx04 Usabilidad. La aplicación tiene que ser fácil de utilizar.
5. RNFx05 Mantenibilidad. La aplicación tiene que ser fácil de mantener.
6. RNFx06 Costo. La aplicación debe ser barata de mantener.

Una primera aproximación , muy genérica, a la aplicación que se desea realizar sería el diagrama de casos de uso que se muestra en la figura 5.2 donde se puede apreciar la interacción del usuario final con el sistema que se pretende desarrollar.

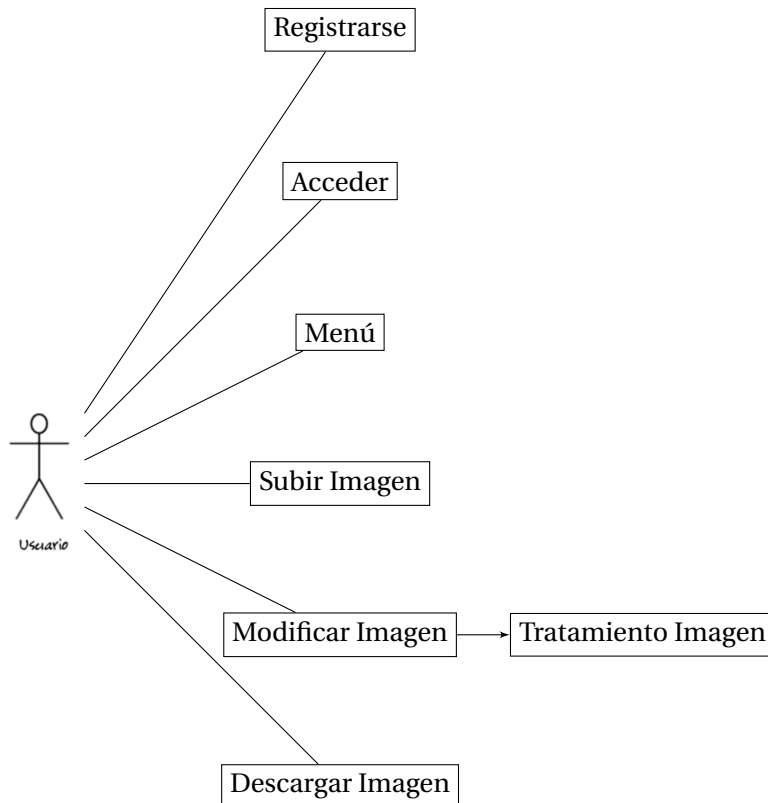


Figura 5.2: Diagrama de casos de uso.

5.2 Comunicación cliente-servidor

Al tratarse de una aplicación que sigue el paradigma cliente-servidor, la comunicación entre las dos entidades resulta esencial para el desarrollo de las funcionalidades de la misma.

Para llevar a cabo esta comunicación se ha utilizado la tecnología **AJAX** en la parte del cliente, la cual iniciará todas las comunicaciones con el servidor, y un **CGI** en la parte del servidor, el cual será el encargado de ejecutar las funcionalidades solicitadas por los clientes y responder a las peticiones de los mismos.

5.2.1 **AJAX**

AJAX es la tecnología utilizada para establecer comunicaciones entre el cliente y el servidor. Haciendo uso de esta tecnología podemos definir desde la parte del cliente la petición que se desea enviar a un servidor y el correspondiente procesamiento de la respuesta obtenida por parte del mismo.

Para el caso de esta aplicación se ha definido una función de propósito general que permitirá realizar cualquier tipo de petición al servidor llamada `ajaxRequest` y una función de propósito general de procesamiento de respuestas llamada `ajaxResponse`.

La función `ajaxRequest` se muestra en la figura 5.3 y los aspectos a considerar de la misma son los siguientes:

- Variable `peticion`. Hace referencia al tipo de petición **HTTP** (GET,POST...)
- Variable `filename`. Hace referencia al fichero que se encuentra y es ejecutado en el servidor al cual se envían los datos.
- Variable `datosEnviados`. Hace referencia a los datos que se desean enviar al servidor.
- Variable `ajaxreq`. Objeto `XMLHttpRequest` de javascript que permite establecer y controlar la comunicación con el servidor.
 - Función `open`. Abre una conexión con el servidor.
 - Atributo `onreadystatechange`. Ejecuta la función que se le indique cuando la conexión cambie de estado.
 - Función `send`. Envía datos al servidor.

En lo referente a los datos enviados a través de la función `ajaxRequest` al servidor de esta aplicación, todos serán del tipo `FormData` de javascript ya que este permite codificar los datos enviados en formato clave-valor. Esto significa que cada dato o conjunto de datos enviado/s al servidor irá asociado a una clave, que no es más que una cadena de texto.

La variable del tipo `FormData` que se le introduzca como argumento a la función `ajaxRequest` siempre contendrá lo siguiente.

1. Una clave llamada `método` cuyo valor es una cadena de texto que identifica la funcionalidad a realizar por parte del servidor.

```

//Define una petición
function ajaxRequest(peticion, filename, datosEnviados) {

    try {

        ajaxreq = new XMLHttpRequest();

    } catch (error) {

        try {
            ajaxreq = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (error) {
            return false;
        }
    }

    ajaxreq.open(peticion, filename);
    ajaxreq.onreadystatechange = ajaxResponse;
    ajaxreq.send(datosEnviados);
}

```

Figura 5.3: ajaxRequest

2. Siempre contiene los datos necesarios para que el servidor realice la funcionalidad mencionada anteriormente.

Una vez obtenida la respuesta de la petición realizada, el método `ajaxRequest` ejecutará el método `ajaxResponse` para procesar la respuesta obtenida.

La funcionalidad del método `ajaxResponse` es la siguiente:

1. Comprueba que la respuesta a la petición ha sido un éxito verificando que ha recibido un código 200 en la misma. En caso de no recibir un código 200 no realizará ningún tipo de funcionalidad.
2. En caso de recibir un código 200 comprueba el contenido de la respuesta obtenida. La respuesta obtenida por parte del servidor será del tipo *JavaScript Object Notation (JSON)* ya que este permite codificar los datos enviados en formato clave-valor.
3. En caso de obtener una respuesta positiva de la funcionalidad realizada se comprobará el método utilizado en la petición, el cual es reenviado en la respuesta. En base al método recibido se realizará una funcionalidad u otra.

Otra solución podría haber sido pasar por parámetro a la función `ajaxRequest`, una función (*callback*) que se le asignaría al atributo `onreadystatechange` para procesar en una función distinta cada respuesta.

5.2.2 CGI

El **CGI** vendría a ser el núcleo de nuestra aplicación debido a que todas las funcionalidades que se requieran realizar en el servidor serán ejecutadas por el mismo. Además, es el encargado de gestionar todas las peticiones realizadas al servidor por los distintos

usuarios.

Tanto la página **HTML** de sesión como la página **HTML** principal de la aplicación desarrolladas se comunican con él, por lo tanto se debe controlar y identificar en todo momento la funcionalidad que se desea realizar.

Con el objetivo de poder controlar que funcionalidad se debe realizar en cada comunicación con el **CGI**, se ha identificado con una etiqueta distinta cada una. Esta etiqueta es enviada por el cliente, en la petición realizada al servidor, para indicarle la funcionalidad a realizar.

Los posibles etiquetas obtenidas son las siguientes:

- registrarse. Indicará al servidor que se debe realizar la función de registrarse en la aplicación
- acceder. Indicará al servidor que el usuario desea establecer una sesión.
- log_out. Indicará al servidor que el usuario quiere finalizar la sesión.
- subirImagen. Indicará al servidor que se debe guardar la imagen enviada por el cliente.
- pixeles. Indicará al servidor que se desea ejecutar el algoritmo de Poisson sobre una imagen determinada.
- gradient. Indicará al servidor que se desea obtener la imagen de gradiente de una imagen determinada.

El funcionamiento lógico del **CGI**, el cual se muestra en la figura 5.4, es el siguiente:

1. Recibe los datos enviados en la petición del cliente.
2. Comprueba si el usuario tiene creada o no una sesión puesto que las funcionalidades habilitadas serán diferentes para cada caso . La comprobación la realiza verificando que ha recibido en la petición del cliente una *cookie* de sesión.
3. Si el usuario no tiene creada una sesión se comprueba la etiqueta recibida en la petición para averiguar cual de las dos funcionalidades disponibles se debe ejecutar: registrarse o acceder.
4. Si el usuario tiene creada la sesión se comprueba la etiqueta recibida en la petición para averiguar cual de las siguientes funcionalidades se debe ejecutar: log_out, subirImagen, pixeles o gradient.
5. Una vez realizada la funcionalidad, el **CGI** envía una respuesta **HTTP**, cuyo contenido vendrá codificado en formato **JSON** (datos en formato clave-valor), al cliente que le realizó la petición.

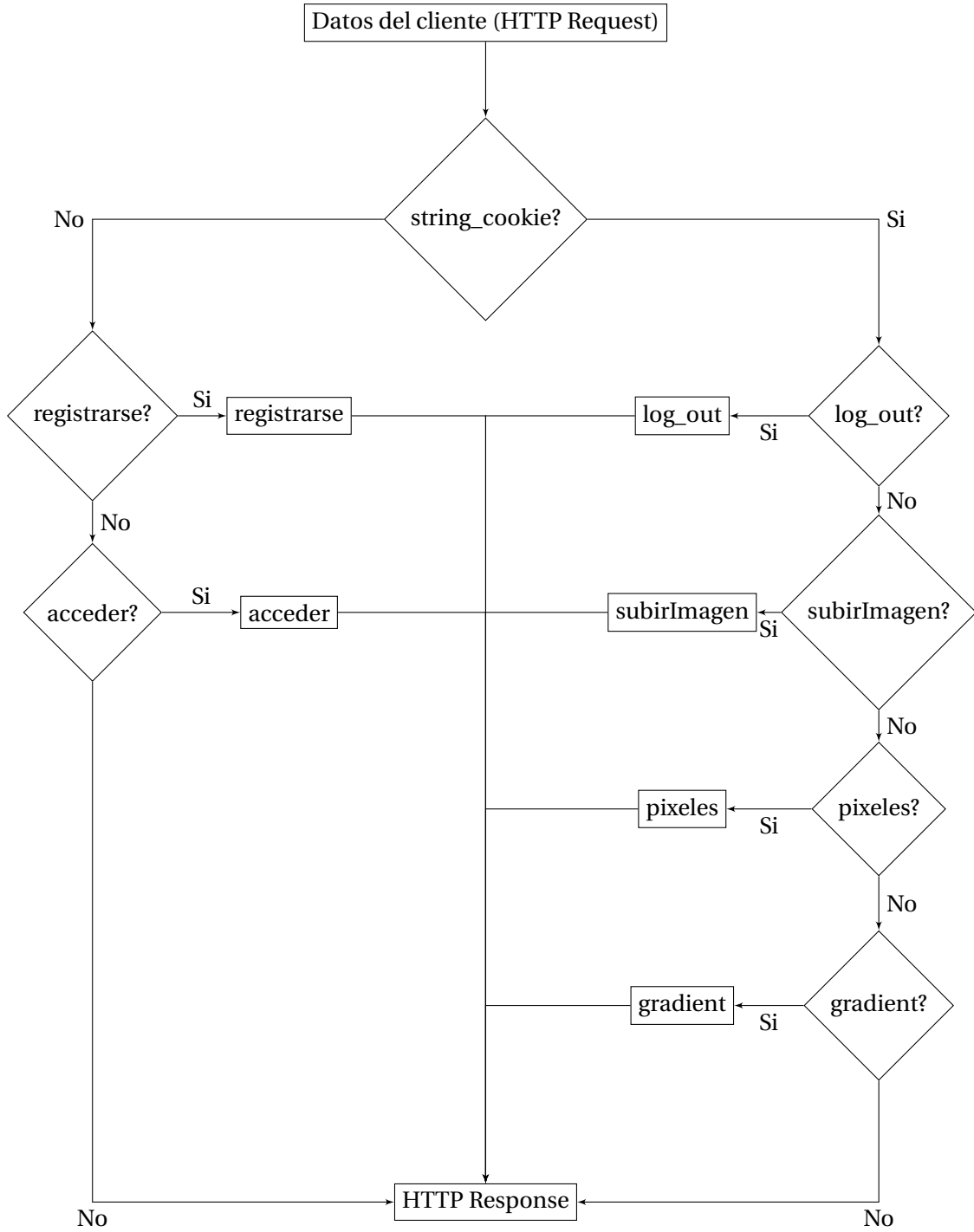


Figura 5.4: CGI

5.3 Almacenaje de los datos.

Otro aspecto importante a tener en cuenta es almacenaje de los datos en el sistema. Tener claro el tipo sistema utilizado para almacenarlos y definir los datos necesarios a almacenar.

Para almacenar los datos se ha hecho uso del módulo *shelve* de python, ya que al tratarse de una aplicación sencilla hacer uso de un motor de base de datos resultaría ineficiente por el exceso de recursos que serían utilizados por el sistema.

Una *shelf* no es más que un diccionario persistente que permite guardar cualquier objeto de python en formato clave-valor.

Para el desarrollo de la aplicación es necesario almacenar los siguientes datos:

- Los datos del usuario registrado en la aplicación los cuales son los siguientes:
 1. Nombre de usuario y la contraseña del usuario. Necesarios para realizar sesiones de usuarios.
 2. Un integer que representará el número de imágenes generadas por el usuario. Utilizado para generar imágenes para el usuario y no producir una sobrescritura con las imágenes que tienen el mismo nombre.

Estos datos forman parte del objeto usuario que será guardado en la *shelf*, utilizando como clave el nombre de usuario.

- Las imágenes de las que hará uso el usuario de la aplicación. Las imágenes se guardarán en un directorio creado exclusivamente para cada usuario registrado.

5.4 Funcionalidades de la aplicación.

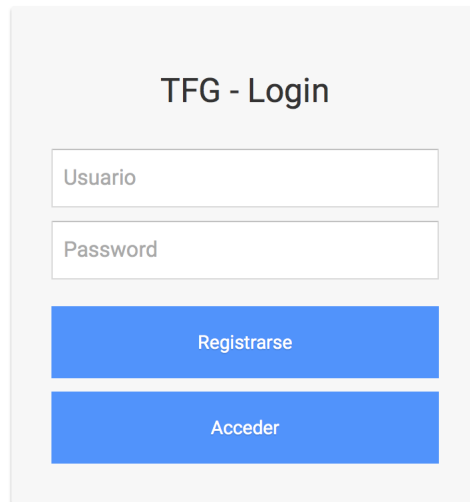
Para explicar las funcionalidades de la aplicación de una manera más concisa, sencilla y entendible, se explicarán de forma secuencial al uso que haríamos de ellas en la aplicación. De esta forma se pretende instruir como se utiliza la aplicación y las funcionalidades que lleva acabo.

Dividiremos la explicación en tres secciones que corresponden a las tres pantallas **HTML** creadas que utilizará el usuario:

- Interfaz de sesión. Pantalla que permitirá al usuario registrarse y realizar sesiones en la aplicación.
- Interfaz de menú de usuario. Pantalla que contendrá un menú para que el usuario pueda acceder a la aplicación que desee utilizar.
- Interfaz de funcionalidades para imágenes basadas en Poisson. Pantalla que permitirá al usuario realizar tratamientos de imágenes haciendo uso del método de Poisson.

5.4.1 Interfaz de sesión.

La primera página que nos encontraremos al hacer uso de la aplicación web, es una página que contiene un formulario como el que se muestra en la figura 5.5.



The image shows a login form titled "TFG - Login". It consists of two text input fields, one labeled "Usuario" and one labeled "Password". Below these fields are two blue buttons: "Registrarse" (top) and "Acceder" (bottom). The entire form is enclosed in a light gray border.

Figura 5.5: Sesión

Esta primera página ha sido desarrollada para poder realizar sesiones de usuarios, las cuales son necesarias para poder garantizar que las acciones llevadas a cabo por un usuario de la aplicación no afecten al resto de usuarios. Por ello, el usuario que quiera hacer uso de la aplicación debe tener una cuenta registrada en el servidor que proporciona la misma.

El uso de este formulario es muy simple:

- **Registrarse.** El usuario debe introducir un nombre de usuario y una contraseña para poder registrarse. La aplicación creará el usuario, introduciendo sus datos en el *shelve* y creando un directorio para el mismo, si no existe ningún usuario que se haya registrado anteriormente con los mismos datos.
- **Acceder.** El usuario debe introducir el nombre de usuario y contraseña, los cuales utilizó para registrarse con anterioridad. Si el usuario es validado por el servidor de forma satisfactoria, este enviará una *cookie* de sesión para el mismo y se accederá a la aplicación. En caso de no ser validado satisfactoriamente, se permanecerá en la página actual.

5.4.2 Interfaz de menú de usuario

La segunda pantalla que nos encontraremos en la aplicación es una pantalla de menú como el que muestra la figura 5.6.

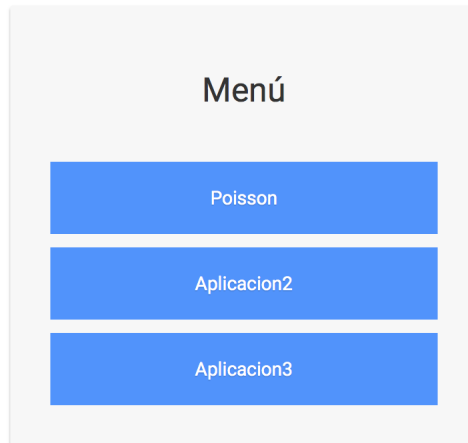


Figura 5.6: Menú

Este menú ha sido creado de cara al desarrollo de futuras aplicaciones con el objetivo de poder tener acceso a cada una de las mismas.

El funcionamiento del menú es muy simple:

1. Se selecciona la aplicación a la que se desea acceder.
2. El navegador envía una petición del tipo GET al servidor. Para ello utilizaremos la función creada con javascript llamada `accederAplicacion` (Figura 5.7).
3. El navegador recibe la página de la aplicación seleccionada y la muestra.

La función `accederAplicacion` (figura 5.7) permite acceder al documento, que contendrá la aplicación, que se le introduzca por parámetro.

```
function accederAplicacion(documento){  
    window.location='/' + documento;  
}
```

Figura 5.7: `accederAplicacion`

5.4.3 Interfaz de funcionalidades para imágenes basadas en Poisson

La tercera pantalla desarrollada es la que permite hacer uso de la técnica de Poisson en imágenes (5.8).

Para hacer la explicación más sencilla y intuitiva, la dividiremos en tres secciones:

- Acceso a las imágenes.
- Modificación y tratamiento de la imagen.
- Obtención de las imágenes.

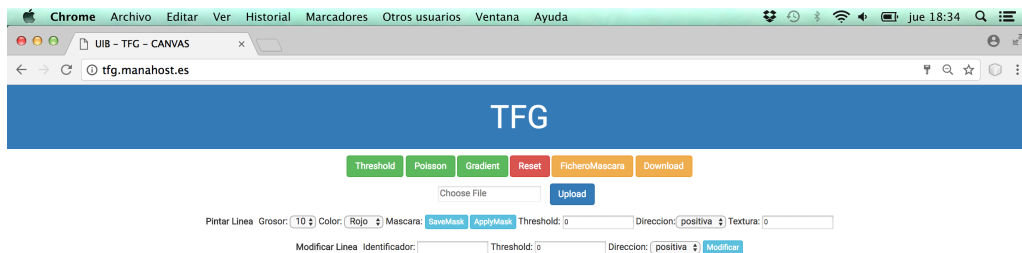


Figura 5.8: Interfaz Poisson

Acceso a las imágenes.

El primer paso en esta interfaz es desarrollar la funcionalidad que permita tener acceso a la imagen que se desea tratar con la aplicación.

Para este propósito se ha creado un botón llamado *upload* (figura 5.8) del tipo *file* (figura 5.9) el cual permite al usuario acceder de manera automática a los archivos de su equipo.

Una vez que el usuario seleccione una imagen, el cliente o navegador hará lo siguiente:

- Enviará la imagen seleccionada al servidor a través de la función `ajaxRequest`.
 - El servidor convertirá la imagen a una imagen en formato *Portable Network Graphics* (PNG).
 - El servidor guardará la imagen en la carpeta del usuario de la sesión

5. DESARROLLO

```
<div class="fileUpload btn btn-primary">
  <span>Upload</span>
  <input id="imagen" name="file" type="file" class="upload" />
</div>
```

Figura 5.9: Boton upload

- Recibirá la imagen seleccionada del servidor para mostrarla tal y como muestra la figura 5.10 a través de:
 - Un objeto *canvas* de HTML que nos permitirá modificar la imagen.
 - Un objeto *image* de HTML que formará parte de un carrusel interactivo de imágenes.

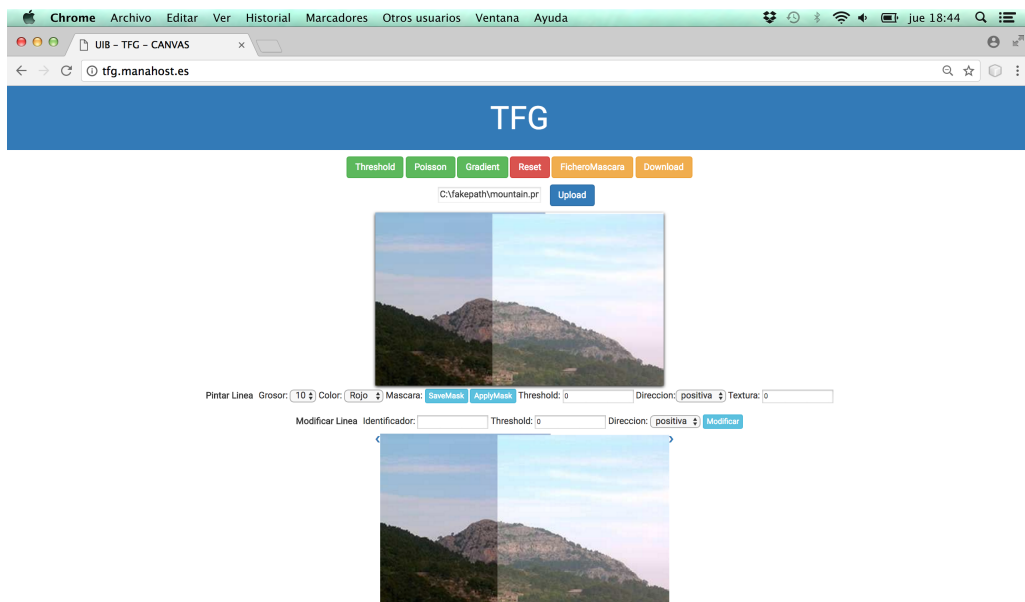


Figura 5.10: Resultado del upload

De esta forma conseguimos tener una imagen almacenada en el servidor y otra imagen idéntica en la parte del cliente con el propósito de:

1. Modificar la imagen cliente sin afectar la imagen que se encuentra en el servidor. Se pretende usar la imagen en la parte del cliente con el objetivo de obtener la máscara de píxeles que luego se enviará al servidor para que sea posteriormente utilizada en la aplicación de la técnica de Poisson.
2. Evitar estar continuamente enviando la imagen en las peticiones al servidor lo cual implicaría enviar más información de la necesaria. En lugar de enviar la imagen, se enviará un identificador (cadena de texto) para que el servidor sepa la imagen que debe tratar.

Modificación y tratamiento de la imagen

Una vez se tenga acceso a la imagen seleccionada, se comenzará a realizar el tratamiento de la misma.

El proceso que se sigue en la aplicación, que se muestra en la figura 5.11, para aplicar el método de edición de imágenes por Poisson es el siguiente:

- 1. Seleccionar una imagen en la que dibujar la máscara.
- 2. Dibujar sobre la imagen y actualizar la máscara.
- 3. Enviar datos al servidor.
- 4. Obtener en el servidor los datos enviados por el cliente.
- 5. Aplicar el método de Poisson.
- 6. Obtener en el cliente el resultado de Poisson.

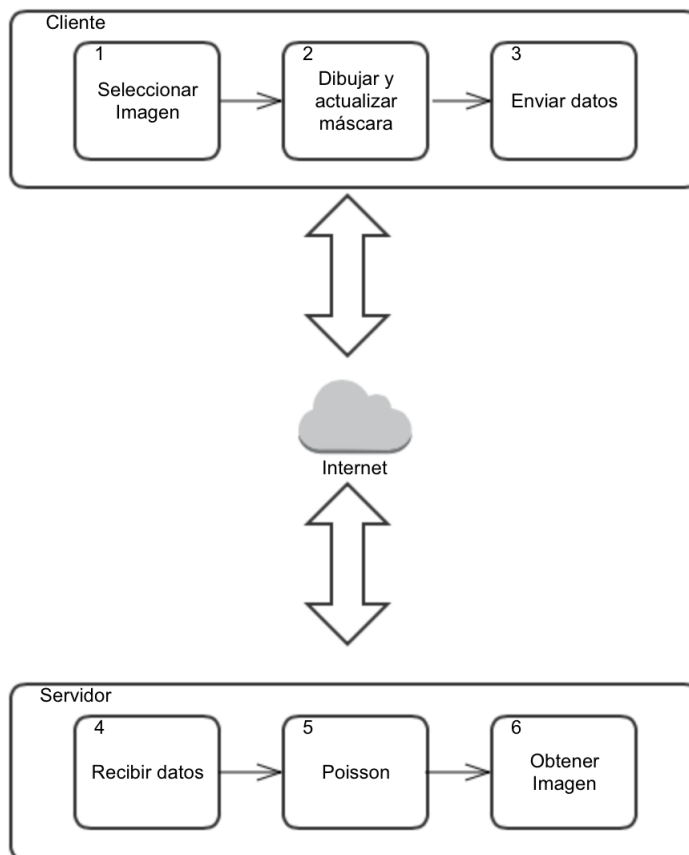


Figura 5.11: Proceso de aplicación de la funcionalidad de Poisson.

1. Seleccionar una imagen en la que dibujar la máscara.

El primer paso para hacer uso de la técnica de Poisson es seleccionar una imagen en la que dibujar la máscara.

Se disponen de tres tipos de imagen para dibujar la máscara y posteriormente aplicar la técnica de Poisson.

1. Imagen original. Se puede dibujar la máscara en la imagen subida a través del botón *upload* de la interfaz para posteriormente aplicar la técnica de Poisson en una réplica de la misma que se encuentra en el servidor.

Las imagen original se encuentra disponible en el objeto *canvas* de javascript y en el carrusel de imágenes en el momento en el que se hace el *upload*.

2. Imagen obtenida por el procesado de Poisson. Se puede dibujar la máscara en una imagen obtenida del proceso de Poisson con anterioridad para volver aplicarlo posteriormente en la réplica que se encuentra en el servidor.

Cada vez que se realice el procesado de Poisson se guardará la imagen resultado en el carrusel de imágenes para que el usuario pueda de manera interactiva seleccionar la imagen que quiere modificar y visualizar los diferentes resultados obtenidos.

3. Imagen de gradiente. Se puede dibujar la máscara en una imagen de gradiente obtenida de la imagen original o imagen obtenida por el procesado de Poisson para posteriormente aplicar la técnica de Poisson en una réplica de la de la imagen, que se encuentra en el servidor, de la cual se obtuvo la imagen de gradiente.

Datos a tener en cuenta para este tipo de imagen:

- Esta imagen sólo es de interés, en el caso de esta aplicación, para proporcionar al usuario una mejor visualización de la imagen a la hora de dibujar la máscara no para aplicar el proceso de Poisson en la misma.
- Esta imagen no es más que una representación de una imagen en la cual se muestran en color blanco los píxeles cuyo gradiente supere el umbral determinado por el usuario en el campo *threshold* (figura 5.8) y en negro los píxeles que no superen el umbral mencionado.
- Se obtiene de la imagen que se encuentra en el *canvas* una vez se pulse el botón de *gradient* mostrado en la interfaz (figura 5.8).

En la figura 5.12 se puede apreciar la imagen de gradiente obtenida de una imagen haciendo uso de un umbral 10.

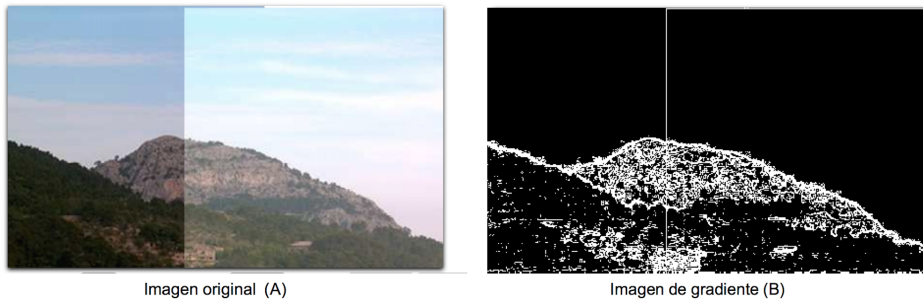


Figura 5.12: Imagen original y de Gradiente.

2. Dibujar sobre la imagen y actualizar el valor de máscara.

Una vez se tenga seleccionada una imagen a la cual dibujar la máscara se comenzará con el proceso de dibujo.

Para realizar esta funcionalidad se han definido tres eventos con javascript:

- Evento de pulsar con el ratón (*mousedown*). Invocará una función de javascript llamada *pulsaRaton* cada vez que se haga *click* derecho con el ratón.
- Evento de desplazar el ratón (*mousemove*). Invocará una función de javascript llamada *mueveRaton* repetidamente mientras el ratón se desplace.
- Evento de dejar de pulsar el ratón (*mouseup*). Invocará una función de javascript llamada *levantaRaton* al dejar de pulsar el ratón.

La logística de las tres funciones es la siguiente:

- *pulsaRaton* (figura 5.13). Esta función inicializa el proceso de pintado.
 - Establece un booleano *estoyDibujando* a *true* para indicar que se va a pintar.
 - Instancia el grosor y el color de la línea con la que se va a dibujar.
 - Crea un *path* para controlar el pintado en el *canvas* a través de la función *beginPath* y se sitúa en la posición inicial (coordenadas donde se pulsa en el *canvas*) con la función *moveTo*
- *mueveRaton* (figura 5.14). Esta función es la encargada de realizar el proceso de pintado.
 - Comprueba que se esta dibujando a través del booleano *estoyDibujando*.
 - * Si el valor de *estoyDibujando* es *true* significará que alguien esta pulsando con el ratón y por lo tanto se debe dibujar.
 - * Si el valor de *estoyDibujando* es *false* significará que alguien mueve el ratón sin pulsar el botón y por lo tanto no se debe dibujar.
 - Se va calculando la posición del ratón y se dibuja la línea a través de la función *lineTo*.

```
function pulsaRaton(capturo) {  
  
    estoyDibujando = true;  
  
    ctx.lineWidth = grosorLinea;  
    //indicamos el color de la línea.  
    ctx.strokeStyle = colorLinea;  
    //Indico que vamos a dibujar  
    ctx.beginPath();  
    //Averiguo las coordenadas X e Y por dónde va pasando el ratón.  
    var position = $("#myCanvas").position();  
    var x = position.left + parseInt($("#myCanvas").css('marginLeft'), 10);  
    var y = position.top + parseInt($("#myCanvas").css('marginTop'), 10);  
  
    ctx.moveTo(capturo.clientX - x, capturo.clientY - y);  
  
}
```

Figura 5.13: pulsaRaton.

```
function mueveRaton(capturo) {  
    if (estoyDibujando) {  
        ///Por dónde vamos dibujando  
        var position = $("#myCanvas").position();  
        var x = position.left + parseInt($("#myCanvas").css('marginLeft'), 10);  
        var y = position.top + parseInt($("#myCanvas").css('marginTop'), 10);  
  
        ctx.lineTo(capturo.clientX - x, capturo.clientY - y);  
        ctx.stroke();  
    }  
}
```

Figura 5.14: mueveRaton.

- levantaRaton (figura 5.15). Esta función es la encargada de finalizar el proceso de pintado.
 - Cierra el *path* utilizado para establecer el camino de dibujo.
 - Pasa el booleano *estoyDibujando* a *false* para indicar que se ha acabado de dibujar.
 - Invoca una función llamada *setDatos* para actualizar la información de la máscara. Esta función hace lo siguiente:
 1. Obtiene el valor del campo *threshold* (figura 5.8).
 2. Obtiene el valor del campo *dirección* (figura 5.8).
 3. Obtiene los píxeles seleccionados en el proceso de dibujo a través del método *GetMask*. Este método recorre un array que contiene todos los píxeles de la imagen con el objetivo de almacenar los números de los píxeles que se encuentran dibujados y modificar el pintado de los mismos (pintándolos de negro) para no obtenerlos en el próximo proceso de dibujo.
 4. Crea un objeto **JSON** con los valores obtenidos anteriormente y lo añade al array de **JSONs**. De esta forma asociamos cada línea dibujada a un **JSON** que contiene los valores obtenidos anteriormente.

5.4. Funcionalidades de la aplicación.

```
function levantaRaton(capturo) {  
  
    //Indico que termino el dibujo  
    ctx.closePath();  
    estoyDibujando = false;  
    setDatos();  
  
}
```

Figura 5.15: levantaRaton.

A modo de ejemplo se ha dibujado en la imagen que aparece en la figura 5.16 una línea roja de grosor 10 encima de la línea blanca de la imagen original. Una vez terminada de dibujar la misma se actualizará la máscara creando un **JSON** que contendrá el threshold (en este ejemplo 0), la dirección (en este ejemplo positiva) y los números de los píxeles seleccionados a modificar por parte del servidor. Además se puede apreciar también como al actualizar la máscara se dibuja esta automáticamente en color negro para no contemplar esta línea en las próximas actualizaciones de la misma.

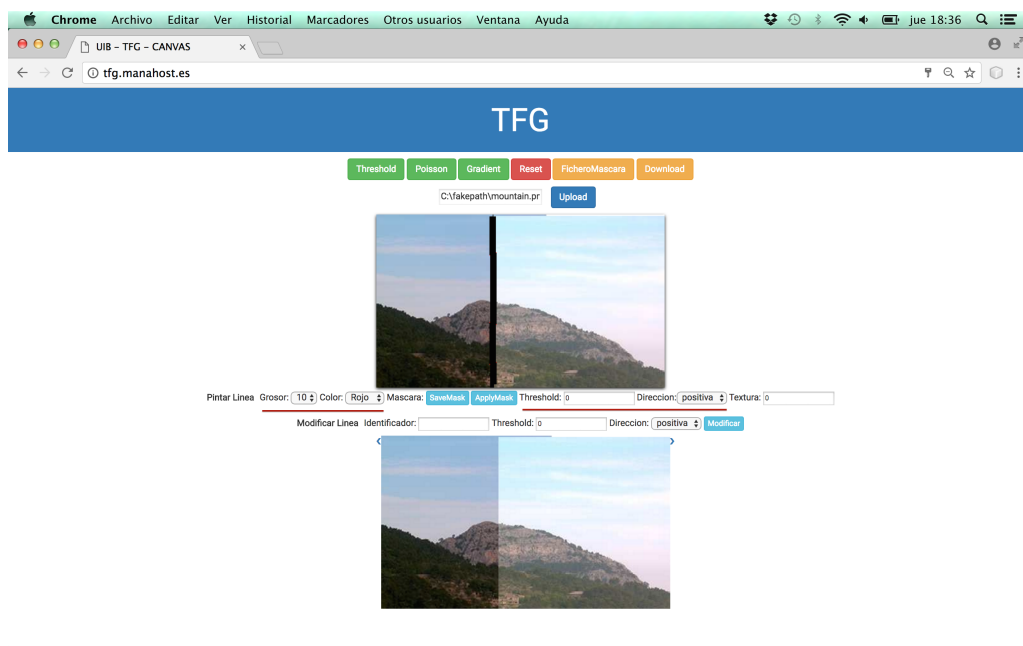


Figura 5.16: Foto dibujada de ejemplo.

3. Enviar datos al servidor.

Para hacer uso del método de Poisson el usuario de la aplicación dispone de un botón en la interfaz (figura 5.8) llamado Poisson.

Cuando el usuario pulse el botón mencionado, la aplicación en la parte del cliente ejecutará una función llamada MetodoPíxeles (figura 5.18) que hará lo siguiente:

- Obtener el nombre de la imagen, el cual se encuentra guardado en un objeto input de **HTML** con identificador imagenActual (figura 5.17), para poderla identificar posteriormente en el servidor.

```
<input id="imagenActual" type="text" style="display: none" >
```

Figura 5.17: imagenActual.

- Obtener la máscara dibujada. Este valor se encuentra guardado en la variable informacionJson y se va actualizando en:
 - Cada proceso de dibujo tal y como se mencionó anteriormente.
 - Cuando se aplica una máscara guardada con anterioridad a través de la funciones SaveMask y ApplyMask de la interfaz (figura 5.8).
- Obtener el valor del campo textura (figura 5.8).
- Introducir en un objeto FormData los siguientes valores asociados a las siguientes claves.

Clave	Valor
metodo	pixeles
namefile	nombre de la imagen (valor imagenActual)
pixeles	máscara
textura	textura

- Enviar el objeto FormData al servidor haciendo uso de la función genérica ajax-Request (figura 5.3) tal y como se mencionó en el apartado de comunicación cliente-servidor [5.2].

```
function MetodoPixeles(){
    var txtImagenActual = document.getElementById("imagenActual").value;
    var textura=document.getElementById("textura").value;

    if (txtImagenActual) {
        var formData = new FormData();
        formData.append('namefile', txtImagenActual);
        formData.append('metodo', 'pixeles');

        var pxs=inicioJson+informacionJson+finJson;
        formData.append('pixeles', pxs);
        formData.append('textura',textura);

        console.log(txtImagenActual);
        console.log(pxs);

        ajaxRequest("POST", pathServidor, formData);
    } else {
        alert('No image!!!');
    }
}
```

Figura 5.18: MetodoPixeles.

4. Recibir en el servidor los datos enviados por el cliente.

El **CGI**, ejecutado por el servidor al recibir la petición **HTTP**, recibirá los datos enviados por el cliente y los recuperará a través de las claves mediante la función de python `getvalue`.

El conjunto de datos a recuperar dependerá de la funcionalidad a realizar tal y como se ha mencionado anteriormente en el apartado de comunicación cliente-servidor [5.2].

Dejando de lado la parte del **CGI** para gestionar las sesiones de usuario, los datos a recuperar de la interfaz de tratamiento de imágenes por Poisson para cada funcionalidad son los siguientes:

- La funcionalidad de subir imagen recupera los siguientes datos:
 1. La cadena de texto utilizada para identificar esta función.
 2. La imagen que se guardará posteriormente en la carpeta del usuario de la sesión.
- La funcionalidad que permite obtener la imagen resultado del algoritmo de Poisson recupera los siguientes datos:
 1. La cadena de texto utilizada para identificar esta función.
 2. La cadena de texto que identificará la imagen a tratar.
 3. La cadena de texto que contendrá la máscara.
 4. La cadena de texto que indicará el parámetro de textura utilizado.
- La funcionalidad que permite obtener la imagen de gradiente recupera los siguientes datos:
 1. La cadena de texto utilizada para identificar esta función.
 2. La cadena de texto utilizada para identificará la imagen a tratar.
 3. El valor del límite (*threshold*) necesario para aplicar la funcionalidad.

Una vez que el **CGI** verifique que se desea ejecutar el algoritmo de Poisson, este ejecutará una función externa llamada `pixel` (figura 5.19) que realiza lo siguiente:

- Crea un fichero que contiene la máscara recibida siguiendo una serie de pautas.
 - Cada línea del fichero se corresponderá con una línea (objeto **JSON**) de la máscara. Por lo tanto se recorrerá el array de **JSONs** (máscara) y a cada objeto del array que se guarde (línea) en el fichero se le añadirá un salto del línea.
 - Cada línea de la máscara (objeto **JSON**) se guardará en el fichero introduciendo un espacio en blanco al principio de cada valor guardado eliminando las comas que separaban los píxeles.
- Ejecuta un programa de `c++` recibe como parámetros de entrada la siguiente información:

- El *path* donde se encuentra la imagen de entrada (infilename).
- El *path* donde se guardará la imagen del salida (outfilename).
- Un string '1' que le indicará al código de c++ que la funcionalidad a realizar es la de Poisson.
- El *path* al fichero que contiene la información necesaria para realizar la funcionalidad.
- Un valor que será utilizado por la funcionalidad de Poisson para recuperar la textura en la imagen.

```
#Ejecuta la funcionalidad de poisson (1).
def pixel(infilename, outfilename, pixelesFileName, pixeles, textura):
    savePixeles(pixelesFileName, pixeles)
    subprocess.call([pathCodigo, infilename, outfilename, "1", pixelesFileName, textura])
```

Figura 5.19: Función Pixel.

5. Aplicar el método de Poisson.

Tal y como se ha mencionado en el apartado anterior, para aplicar el método de Poisson utilizaremos un programa desarrollado con el lenguaje de programación c++.

El proceso, representado de forma genérica, que sigue este programa se muestra en la figura 5.22 y es el siguiente:

1. Recibe como argumentos de entrada los siguientes valores:
 - a) namein. *Path* donde se encuentra la imagen de entrada.
 - b) nameout. *Path* donde se guardará la imagen de salida.
 - c) operation. Indicará la funcionalidad a realizar por el programa.
 - d) cadena. Datos necesarios para realizar la funcionalidad.
 - e) textura. Dato adicional que será utilizado en caso de hacer uso del algoritmo de Poisson.
2. Carga en memoria la imagen que se desea tratar (namein) a través de la función `io_png_read_u8_rgb` que nos proporciona el módulo `png` de c++.
3. Realiza una copia de los canales R, G y B de la imagen de entrada (namein) a través de la función `input2RGB` (figura 5.20) . Para ello esta función hace lo siguiente:
 - Crea tres punteros, uno para cada canal (R, G y B).
 - Recorre la imagen leída anteriormente y guarda el valor R, G y B de cada píxel en la misma posición para las tres variables creadas.
 - Devuelve las tres variables (R, G y B).

```

17 void input2RGB(unsigned char *input,
               unsigned char **RR, unsigned char **GG, unsigned char **BB,
               int size)
{
    unsigned char *R, *G, *B;
    int n;

    R=new unsigned char[size];
    G=new unsigned char[size];
    B=new unsigned char[size];
    for (n=0; n < size; n++) {
        R[n]=input[n];
        G[n]=input[size+n];
        B[n]=input[2*size+n];
    }

    *RR=R;
    *GG=G;
    *BB=B;
}

```

Figura 5.20: input2RGB.

4. Comprueba el tipo de operación a realizar a través del valor recibido como argumento (operation) .
 - Si recibe el valor '1' realizará la función poisson_pixel.
 - Si recibe el valor '2' realizará la función gradient_image.
5. Realiza una de las dos operaciones (poisson_pixel o gradient_image).
 - poisson_pixel. Esta función permite realizar la reconstrucción de Poisson sobre la imagen de entrada (namein) y obtener la imagen resultado de la misma.
 - Recibe como argumentos los canales R, G y B de la imagen y el *path* (variable cadena) a un fichero que contiene la información para realizar la funcionalidad.
 - Calcula el gradiente para los tres canales a través de la función compute_gradient.
 - Recorre el fichero obtenido a partir de su *path* línea por línea realizando las siguientes funciones para cada una.
 - * Almacena en un array la información de la línea leída del fichero. La primera posición del array contiene el valor del *threshold*, la segunda contiene el valor de la dirección y el resto de posiciones contienen los píxeles a modificar.
 - * Si el valor del *threshold* es cero, todos los gradientes de los píxeles leídos de la línea se vuelven cero. La dirección no se tiene en cuenta puesto que no existirá ningún vector.
 - * Si el valor del *threshold* no es cero, se modificarán los vectores gradiente de los píxeles obtenidos asignándoles un nuevo vector gradiente que tendrá como módulo el *threshold* obtenido.
 - Realiza un reconstrucción de la textura de los píxeles en base al parámetro textura recibido como argumento.
 - Realiza la reconstrucción de Poisson de cada canal con los gradientes de los mismos a través de la función Poisson_reconstruction.
 - gradient_image. Esta función permite obtener la imagen de gradiente de la imagen de entrada (namein).

- Recibe como argumentos los canales R, G y B de la imagen y una variable (th) que indicará el umbral del algoritmo.
- Calcula el gradiente para los tres canales a través de la función `compute_gradient`.
- Calcula la magnitud de los gradientes a través de la función `compute_gradient_magnitude`.
- Vuelve cero los tres canales a través de la función de c++ `memset`.
- Vuelve blancos los píxeles cuya magnitud de gradiente sea superior al umbral (th).

Para la aplicación de estas dos funcionalidades se requiere siempre el cálculo del gradiente de los píxeles el cual es realizado por una función llamada `compute_gradient` y que implementa las fórmulas de la ecuación 4.15 (gradiente bilineal).

6. Realiza una sobrescritura de los canales R, G y B de la imagen de entrada (namein) a través de la función `RGB2output` (figura 5.21). Para ello esta función hace lo siguiente:
 - Recibe los canales R, G y B como argumento.
 - Modifica los canales R, G y B de la imagen original.

```
*/
void RGB2output(unsigned char *R, unsigned char *G, unsigned char *B,
               unsigned char *output, int size)
{
    int n;

    for (n=0; n < size; n++) {
        output[n]=R[n];
        output[size+n]=G[n];
        output[2*size+n]=B[n];
    }
}
```

Figura 5.21: RGB2output.

7. Escribe la imagen en el *path* que se le introdujo por parámetro (nameout) haciendo uso de la función `io_png_write_u8` que se encuentra en el módulo `png` de C++.

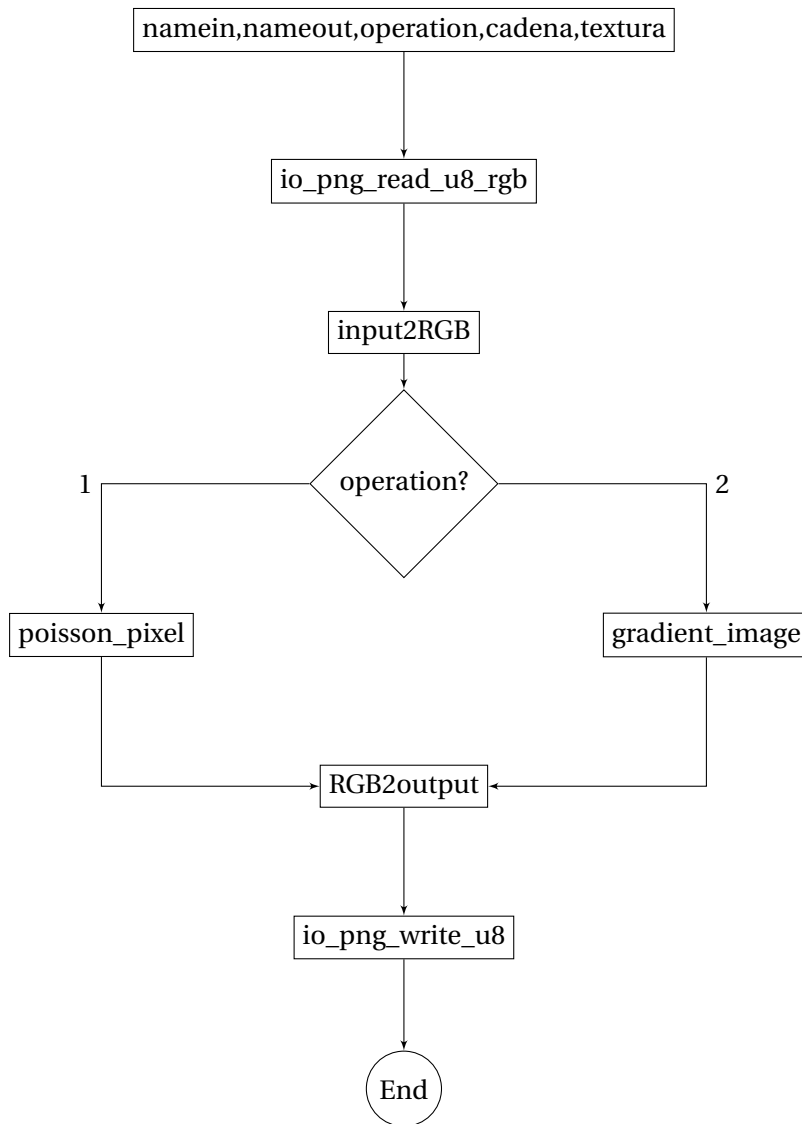


Figura 5.22: Testpoisson (c++)

6. Obtener en el cliente el resultado de Poisson.

Una vez se haya realizado la funcionalidad de Poisson mediante el uso del programa de c++ descrito anteriormente, se deberá obtener en la parte del cliente (navegador web) la imagen resultado generado por el mismo.

Para obtener esta imagen se sigue el siguiente proceso:

- Se envía una respuesta desde el CGI al cliente, que contiene la información codificada como JSON, en la cual se le indica: la funcionalidad realizada (metodo), la ruta (URL) de la imagen (urlout) y el nombre de la imagen (descripcion) tal y como se puede apreciar en la figura 5.23.
- El cliente al recibir la respuesta por parte del servidor ejecutará la función ajax-

```
print json.dumps({'metodo':metodo,'resp':'ok','urlout':'http://'+dominio+'/cgi-bin/files/'+sid+'/'+ outfile,'descripcion':outfile})
```

Figura 5.23: Contenido respuesta.

Response la cual hará principalmente lo siguiente (figura 5.24):

- Creará una imagen a través de javascript y descargará la imagen del servidor haciendo uso del atributo src y de la URL que le envió el servidor al cliente en la respuesta (urlout).
- Añadirá la imagen creada al *canvas* a través de la función `addImageToCanvas`.
- En caso de que el método ejecutado por el servidor no sea el de obtener la imagen de gradiente se añadirá la imagen obtenida al carrusel de imágenes a través de la función `addImageToSlide`.

```
image = new Image();
image.src = response['urlout'];

if (image){
    image.onload=function(){
        image.id=response['descripcion'];
        addImageToCanvas(image);
    };
}

if(metodoS!='gradient'){
    var txtImagenActual = document.getElementById("imagenActual");
    txtImagenActual.value = response['descripcion'];

    addImageToSlide(response['descripcion'], response['urlout'], CurrentSlidImgToCanv);
}
```

Figura 5.24: Procesar respuesta.

En la figura 5.25 se puede observar la imagen resultado de la aplicación del algoritmo de Poisson eliminando los píxeles que forman parte de la máscara creada en la figura 5.16.

Se puede apreciar en el resultado obtenido como al eliminar el gradiente de los píxeles que formaban una línea de separación entre dos zonas de diferente tonalidad en los píxeles y aplicar posteriormente el algoritmo de Poisson, se ha obtenido una imagen con una tonalidad uniforme y diferente a las dos tonalidades que se aprecian en la imagen original.

5.4. Funcionalidades de la aplicación.

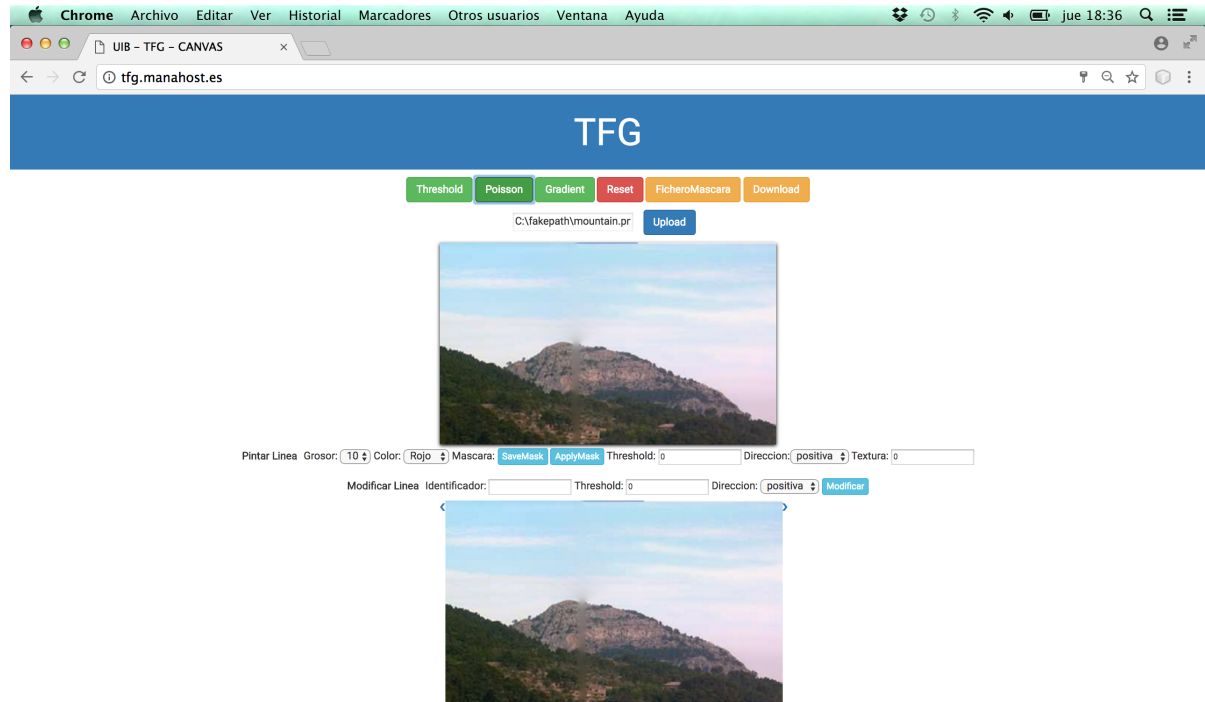


Figura 5.25: Ejemplo de resultado de la aplicación.

Obtención de las imágenes

Otra funcionalidad de interés que contiene la aplicación es la de descargar las imágenes generadas por la misma. Esta funcionalidad es importante ya que permite al usuario final almacenarlas en su propio equipo.

Para realizar esta tarea se ha desarrollado una función llamada `DownloadImg` (figura 5.26) la cual permite descargar la imagen mostrada en el *canvas*.

```
//Permite descargar en local la imagen del canvas.  
function DownloadImg() {  
  
    //Convertimos el canvas a imagen.  
    var descarga = c.toDataURL("image/png");  
    //manipulamos el mimetype de la imagen y la pasamos a image/octet-stream para que el navegador descargue la imagen.  
    descarga = descarga.replace("image/png", "image/octet-stream");  
    document.location.href = descarga;  
}
```

Figura 5.26: `DownloadImg`.

PRUEBAS

En este capítulo se mostrará un subconjunto del total de pruebas realizadas con la aplicación desarrollada.

6.1 Ejemplo de seamless stitching.

En esta primera sección del capítulo se mostrará un breve ejemplo en el que se muestran los resultados obtenidos para un caso de *seamless stitching* como es el de la figura 6.1.



Figura 6.1: mountain

Si aplicamos el algoritmo de Poisson haciendo uso de la máscara mostrada en la figura 6.1 sobre esta imagen obtendríamos el resultado que se muestra en la imagen de la izquierda de la figura 6.2.

Pese a que el resultado obtenido (imagen de la izquierda de la figura 6.2) es bastante bueno, se pueden observar unas costuras producidas al eliminar el gradiente de la

parte de la montaña. Este resultado puede ser mejorado aplicando el algoritmo para reconstruir la textura descrito en el capítulo de Conceptos Básicos. El resultado se puede observar en la imagen de la derecha de la figura 6.2.



Figura 6.2: Resultados mountain.

6.2 Ejemplos de eliminación de sombras.

Esta segunda sección se centrará en el objetivo del proyecto: la eliminación de sombras en imágenes.

Todos los resultados obtenidos se han logrado haciendo uso de la aplicación de Poisson con corrección de textura ya que al no hacer uso de este tratamiento se obtenían resultados en los que se podía apreciar costuras más o menos visibles (dependiendo de la imagen procesada).

La primera imagen a tratar es la que aparece a la izquierda en la figura 6.3. En la imagen de la derecha de la misma se puede apreciar como se ha eliminado de una forma bastante aceptable la sombra frontal, la sombra del árbol del fondo y incluso la banderilla amarilla.



Figura 6.3: Campo de golf

Otro buen resultado de eliminación de sombras obtenido de la aplicación de Poisson es el obtenido de la imagen izquierda de la figura 6.4 tal y como se puede apreciar en la imagen de la derecha de la misma en la que se eliminó la sombra frontal.

6.2. Ejemplos de eliminación de sombras.



Figura 6.4: Caballos

En las figuras 6.5 y 6.6 se puede observar dos ejemplos de eliminación de sombras más. En estos casos no se han obtenido resultados tan espectaculares como los dos primeros pero sí bastante aceptables.



Figura 6.5: Parque

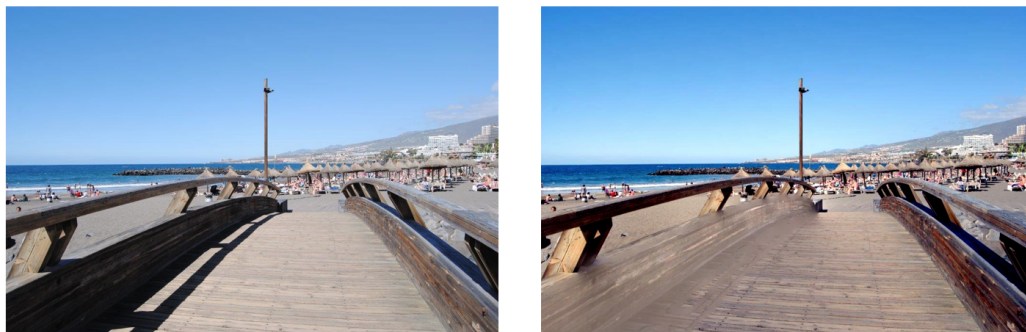


Figura 6.6: Paseo.

6.3 Creación de gradientes

Otra de las aplicaciones proporcionadas por la interfaz es la de modificar el vector gradiente de la máscara seleccionada.

Por ejemplo se puede observar en la figura 6.7 como se ha conseguido iluminar una zona de la imagen aplicando el algoritmo de estimación de gradientes descrito en el capítulo de conceptos básicos con una magnitud de gradiente de 20 y una dirección positiva o como se ha conseguido crear una sombra en una imagen (figura 6.8) aplicando una magnitud de gradiente de 10 y dirección negativa en el algoritmo mencionado.

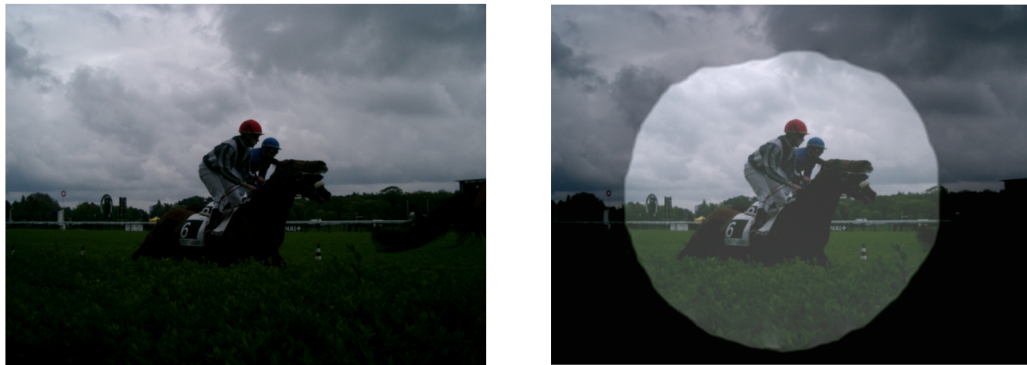


Figura 6.7: Imagen jinetes.



Figura 6.8: Traffic.

CONCLUSIONES

Como se ha podido apreciar en el capítulo de resultados, se ha conseguido cumplir el objetivo de eliminar sombras en imágenes mediante el uso de la técnica de Poisson a través de la aplicación desarrollada. Sin embargo, se tuvo que añadir un tratamiento adicional de textura para evitar producir, en la medida de lo posible, los artefactos que se generaban al eliminar sombras en las imágenes. Al hacer uso de este tratamiento se obtuvieron mejores resultados.

De cara a futuro, sería interesante intentar de mejorar el tratamiento de la imagen con el objetivo de obtener resultados más aceptables en imágenes con una textura menos uniforme ya que son las que producen los peores resultados.



MANUAL DE INSTALACIÓN

Este manual detallará los pasos a realizar para la instalación de la aplicación en una máquina Linux:

1. Actualizar las librerías:

```
sudo apt-get update
```

2. Instalar build-essential para poder hacer uso del lenguaje de programación C++:

```
sudo apt-get install build-essential
```

3. Instalar python junto con la librería Pillow del mismo:

```
sudo apt-get install python2.7  
pip install Pillow
```

4. Instalar el servidor apache. Para ello se utilizará el siguiente comando.

```
sudo apt-get install apache2
```

5. Guardar el directorio html que contiene la aplicación dentro del directorio */var/www*

6. Acceder al site configurado por defecto en apache. El fichero de configuración se encuentra en el directorio */etc/apache2/sites-enabled*

Una vez se acceda al fichero default se debe hacer lo siguiente:

- En la directiva `ServerName`. Se debe indicar el nombre de dominio o la dirección ip del servidor.

- En la directiva DocumentRoot. Se debe establecer como ruta `/var/www/html`
- Se debe comentar la configuración por defecto del CGI (ScriptAlias y Directory) y añadir una nueva configuración tal y como muestra la figura A.1

```
ServerName tfg.manahost.es
ServerAdmin lere@lere.com
DocumentRoot /var/www/html

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

<Directory /var/www/html>
    Options FollowSymLinks MultiViews
    AllowOverride None
    AuthType Basic
    AuthName tfg.manahost.es
    AuthUserFile /etc/apache2/users/users
    Require valid-user
</Directory>

#ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
#<Directory "/usr/lib/cgi-bin">
#    AllowOverride None
#    Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
#    Require all granted
#</Directory>

ScriptAlias /cgi-bin/ /var/www/html/cgi-bin/
<Directory "/var/www/html/cgi-bin">
    AddHandler cgi-script .cgi .py
    AddHandler default-handler .jpg .png .gif .js .txt .bat .css
    AllowOverride All
    Options +Indexes +FollowSymLinks +ExecCGI
    Require all granted
</Directory>
```

Figura A.1: Fichero Site

7. Reiniciar el servidor. Cada vez que se modifica el fichero de configuración del site se debe reiniciar o recargar el servidor.

La siguiente directiva permite reiniciar el servidor.

```
sudo service apache2 restart
```

Una vez realizados estos pasos el servidor apache quedaría configurado de manera correcta.

8. Acceder al directorio `/var/www/html/cgi-bin` y aplicar los siguientes comandos para añadir permisos a los directorios:

- `files`. Es necesario darle permisos para poder guardar en su interior las imágenes generadas por la aplicación.

```
sudo chmod -R 777 files
```

- `Session`. Es necesario darle permisos para poder guardar la información de los usuarios que se registren en la aplicación.

```
sudo chmod -R 777 Session
```

- `codigoBueno`. Es necesario darle permisos a este directorio para poder ejecutar desde python el código c++ que se encuentra dentro del mismo.

```
sudo chmod -R 777 PruebaTFG
```

9. Acceder al directorio `codigoBueno` (`/var/www/html/cgi-bin/codigoBueno`).

a) Instala las siguientes librerías:

```
sudo apt-get install libfftw3-dev
sudo apt-get install libpng12-dev
```

b) Realizar una limpieza:

```
make clean
```

c) Compilar el código C++:

```
make
```

10. Acceder al fichero `prueba1.py` que se encuentra en el directorio `/var/www/html/cgi-bin` y modificar el valor de la variable dominio asignándole como valor el dominio o la dirección ip de la máquina

En resumen para tener operativa la aplicación se debe realizar lo siguiente.

- Tener instalado C junto con sus librerías `libfftw3-dev` y `libpng12-dev`.
- Tener instalado Python junto con su librería Pillow.
- Tener configurado el site del servidor correctamente.
- Tener agregados los permisos al directorio que contiene el código C y a los dos directorios que utiliza la aplicación para guardar datos ya que en caso contrario dará un error en la aplicación debido a permisos de superusuario.

MANUAL DE USUARIO

Este manual se ha redactado con el objetivo de explicar la forma de uso de las distintas herramientas ofrecidas por la aplicación (figura B.1).

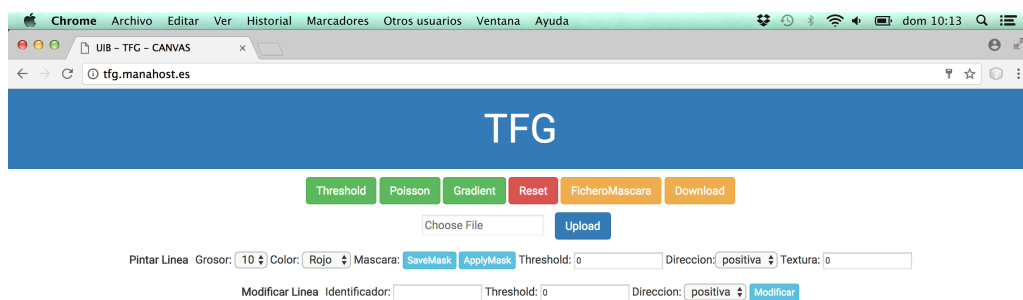


Figura B.1: Aplicación de Poisson

Botón *Upload*. Permite seleccionar al usuario una imagen de su equipo. Esta será la primera acción a llevar a cabo por parte del usuario y la primera vez que se aplique será cuando aparezca el *canvas* utilizado para tratar la imagen y el carrusel de imágenes utilizado para ir seleccionando de manera dinámica una imagen a tratar de entre todas las que se han obtenido de los resultados de procesamientos anteriores (figura B.2).

B. MANUAL DE USUARIO

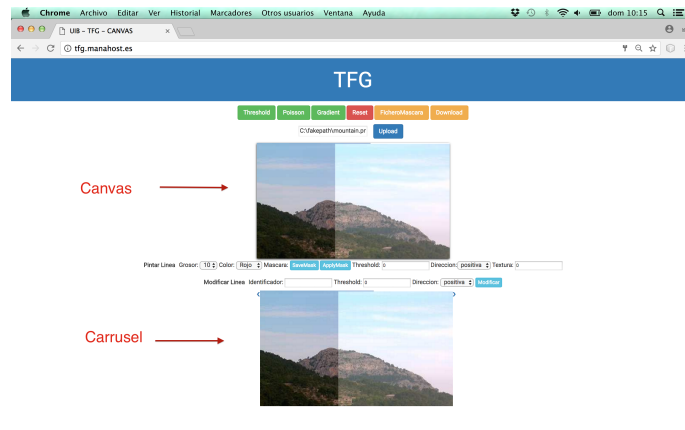


Figura B.2: Ejemplo upload.

Botón *Gradient*. Permite obtener la imagen de gradiente de la imagen mostrada en el *canvas*. La imagen de gradiente obtenida variará dependiendo del valor introducido en el campo *threshold* por parte del usuario. Las figuras B.3 y B.4 muestran los resultados obtenidos para un *threshold* 10 y 50 respectivamente.

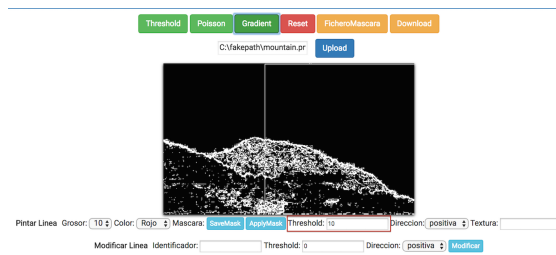


Figura B.3: Imagen de gradiente threshold 10

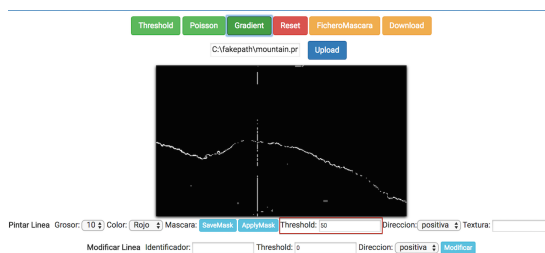


Figura B.4: Imagen de gradiente threshold 50

Botón *Poisson*. Permite obtener los resultados de la aplicación del algoritmo de Poisson en base a la edición selectiva de valores de gradiente. Por lo tanto antes de hacer uso de esta funcionalidad se debe realizar una selección de los píxeles que serán tratados posteriormente por el citado algoritmo (selección de una máscara).

Para realizar la selección de los píxeles lo que se hace es dibujar sobre el *canvas* los píxeles a modificar.

- Campo *Grosor*. Se selecciona un grosor de línea (1-20).
- Campo *Color*. Se selecciona un color de dibujo (Rojo, Verde o Azul).

Cada línea dibujada (la cual identificará un conjunto de píxeles) irá asociada al valor de los campos *threshold* y *dirección*. De esta forma podemos tener varias líneas dibujadas asociadas cada una a distintos valores de los parámetros *threshold* y *dirección* que serán utilizados para la edición de los gradientes de los píxeles seleccionados en las mismas.

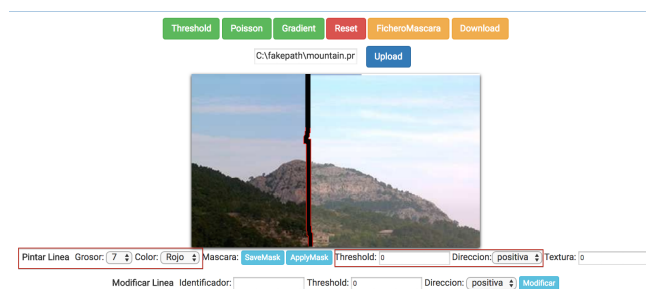


Figura B.5: Ejemplo:Primera línea dibujada.



Figura B.6: Ejemplo:Segunda línea dibujada.

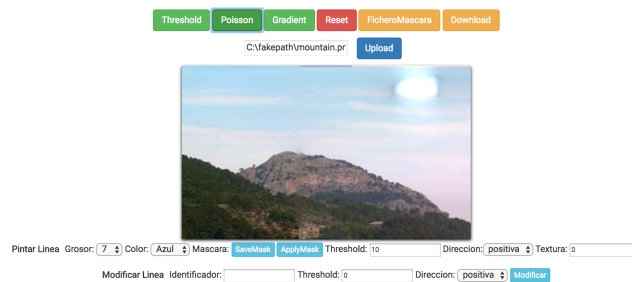


Figura B.7: Ejemplo:Resultado obtenido.

A modo de ejemplo se ha dibujado una primera línea con un *threshold* 0 y *direccion* positiva (figura B.5) y una segunda línea con un *threshold* 10 y *direccion* positiva (B.6). Una vez dibujada la máscara se aplicará el algoritmo de Poisson (figura B.7). Al aplicarlo se enviará además de esta máscara que hemos dibujado el valor del campo *textura* que indicará al algoritmo de Poisson el parámetro que ha de utilizar para reconstruir la textura de los gradientes eliminados (líneas con *threshold* 0).

Para facilitar la selección la máscara a utilizar se han creado los botones *SaveMask* y *ApplyMask* que permiten, respectivamente, guardar la máscara actual y aplicar la máscara guardada además de un gestor que permite modificar el valor del *threshold* y *dirección* de líneas que forman parte de la máscara actual.

El uso del gestor mencionado es muy simple:

- Identificador. Selecciona una línea a modificar (valores [0 - número de líneas -1])
- Se selecciona un *threshold*.
- Se selecciona una dirección.
- Se pulsa en el botón de modificar.

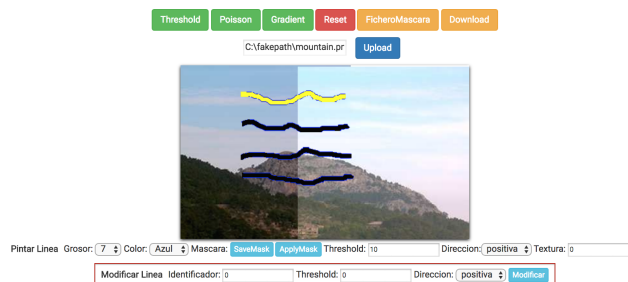


Figura B.8: Ejemplo: Seleccionar primera línea

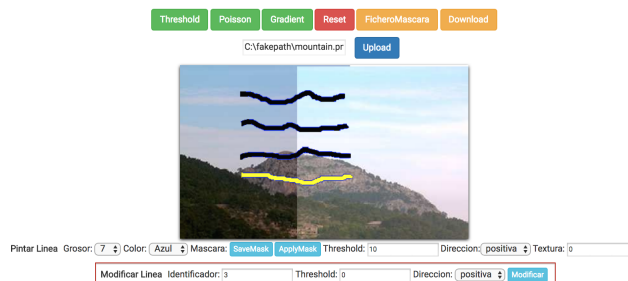


Figura B.9: Ejemplo: Seleccionar última línea.

Botón *Reset*. Permite volver a iniciar el tratamiento de la imagen seleccionada en el *canvas*. Eliminará la máscara seleccionada de píxeles y la visualización de la imagen de

gradiente (en caso de encontrarse visualizada) para volver a la imagen original de la misma.

Botón *Download*. Permite descargar la imagen que se muestra en el *canvas*.

Botón *FicheroMascara*. Permite descargar un fichero que contiene la máscara actual.

Se puede acceder a la aplicación a través desde la siguiente **URL**:

<http://tfg.manahost.es>

Credenciales (login/password): tfg/tfg

la cual estará disponible de manera temporal.

Para más información contactar con el tutor del trabajo José Luis Lisani Roca a través del siguiente correo:

joseluis.lisani@uib.es

BIBLIOGRAFÍA

- [1] T. Perelló, “Mosaic automàtic d’imatges amb correcció de color,” 2014, projecte Final de Carrera, Enginyeria Tècnica de Telecomunicació. **1**
- [2] J. Díaz, “Eliminación de sombras en imágenes digitales,” 2009, projecte Final de Carrera, Enginyeria Informàtica. **1**
- [3] B. A. Forouzan, *Transmisión de datos y redes de comunicaciones*, cuarta ed. Mc Graw Hill, 2007.
- [4] J. Pavón Mestras, “Aplicaciones web/Sistemas web,” *Universidad Complutense Madrid*.
- [5] “Métodos de gradiente,” http://www.varpa.org/mgpenedo/cursos/Ip/Tema7/nodo7_2.html.
- [6] P. Perez, M. Gangnet, and A. Blake, “Poisson Image Editing,” *Microsoft Research UK*, vol. 2, 2013. **4.2.3**
- [7] A.-B. Petro and C. Sbert, “Selective Contrast Adjustment by Poisson Equation,” *Image Processing On Line*, vol. 3, pp. 208–222, 2013. **4.2.3**