



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

**Fotomosaicos empleando
descriptores visuales de
MPEG-7 en la *GPU***

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
por el Bachiller
Víctor Javier Felipe Navarro
para optar al título de
Licenciado en Computación

Tutor: Prof. Esmitt Ramírez Jacobo

Caracas, 05 de junio de 2014

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

ACTA DEL VEREDICTO


Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado presentado por el Bachiller Víctor Javier Felipe Navarro, C.I.: V-18.467.057, con el título *Fotomosaicos empleando descriptores visuales de MPEG-7 en la GPU*, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 05 de junio de 2014, a las 3:00 p.m., para que su autor lo defendiera en forma pública, en el *Centro de Computación Gráfica*, lo cual se realizó mediante una exposición oral de su contenido, y luego respondió satisfactoriamente a las preguntas que le fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas a los cinco días del mes de junio del año dos mil catorce, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Esmitt Ramírez Jacobo.


Prof. Esmitt Ramírez Jacobo


Prof. Zenaida Castillo


Prof. Héctor Navarro

Resumen

Un fotomosaico es una composición de imágenes organizadas en una malla que, vista desde cierta cercanía, permite apreciar dichas imágenes y desde cierta lejanía, la silueta de una imagen conformada por éstas, siendo ampliamente utilizado en distintos ámbitos como el arte y la publicidad. La calidad visual de los fotomosaicos está ligada a la selección de los mejores candidatos, los cuales pueden ser descritos en términos de color, textura o forma. El estándar *MPEG-7* (conocido como *Multimedia Content Description Interface*) presenta diversos descriptores visuales que pueden ser utilizados para obtener características a partir de imágenes. Su costo computacional es muy elevado, dada la cantidad de cálculos que realizan. Sin embargo, estos cálculos pueden acelerarse haciendo uso de arquitecturas que permiten el procesamiento en paralelo, como es el caso de las *GPUs* (*Graphics Processing Units*). En este trabajo se presenta una aplicación que permite generar fotomosaicos haciendo uso de descriptores visuales definidos en el estándar *MPEG-7*, paralelizados en la *GPU*.

Palabras claves: fotomosaicos, descriptores visuales, *MPEG-7*, *GPU*.

Abstract

A photomosaic is a composition of images organized in a mesh that, viewed from a near distance, let appreciate these images and from far, the silhouette of the image formed by these, being widely used in different fields like art and advertising. The visual quality of the photomosaics is related to the selection of the best candidates, who can be described in terms of color, texture and shape. The *MPEG-7* standard (known as *Multimedia Content Description Interface*), presents different visual descriptors, that can be used to get features from images. Their computational cost is very high, given the amount of operations performed. However, these operations can be accelerated taking advantage of the architectures that provide parallel processing like the *GPUs* (*Graphics Processing Units*). This work presents an application that generates photomosaics using visual descriptors defined in the *MPEG-7* standard, parallelized on the *GPU*.

Keywords: photomosaics, visual descriptors, *MPEG-7*, *GPU*.

Tabla de Contenidos

| | |
|--|-----------|
| Introducción | VI |
| 1. Proceso de generación de fotomosaicos | 1 |
| 1.1. Conceptos básicos | 1 |
| 1.2. Flujo de trabajo de una aplicación generadora de fotomosaicos | 2 |
| 1.2.1. Selección de la imagen base | 2 |
| 1.2.2. Generación de los vectores característicos | 2 |
| 1.2.3. Creación de la malla de parches | 3 |
| 1.2.4. Selección del mejor candidato para cada parche | 4 |
| 1.2.5. Redimensionamiento de los candidatos | 5 |
| 1.3. Proceso de adquisición de imágenes | 6 |
| 1.4. Proceso de almacenamiento de imágenes | 7 |
| 1.4.1. Bases de datos relacionales | 7 |
| 1.4.2. Bases de datos <i>NoSQL</i> | 7 |
| 1.4.3. Relacionales vs. <i>NoSQL</i> | 7 |
| 2. Descriptores visuales de <i>MPEG-7</i> | 10 |
| 2.1. Estándar <i>MPEG-7</i> | 10 |
| 2.2. Descriptores visuales | 11 |
| 2.2.1. Color | 11 |
| 2.2.2. Textura | 11 |
| 2.2.3. Forma | 12 |
| 2.2.4. Movimiento | 12 |
| 2.3. Descriptor de colores dominantes | 13 |
| 2.3.1. Proceso de extracción | 13 |
| 2.3.2. Proceso de evaluación | 16 |
| 2.4. Descriptor de histograma de bordes | 16 |
| 2.4.1. Partición del espacio de imagen | 16 |
| 2.4.2. Tipos de bordes | 17 |
| 2.4.3. Semántica del histograma de bordes | 17 |
| 2.4.4. Proceso de extracción | 18 |
| 2.4.5. Histogramas global y semi-global de bordes | 19 |
| 2.4.6. Proceso de evaluación | 20 |
| 2.5. Descriptor de forma basado en regiones | 21 |
| 2.5.1. Coeficientes de la transformada radial angular | 21 |

| | | |
|-----------|---|-----------|
| 2.5.2. | Proceso de discretización | 22 |
| 2.5.3. | Correspondencia en coordenadas polares | 22 |
| 2.5.4. | Cálculo aproximado de los coeficientes | 23 |
| 2.5.5. | Proceso de evaluación | 25 |
| 3. | Planteamiento de la solución | 26 |
| 3.1. | Justificación | 26 |
| 3.2. | Objetivos | 27 |
| 3.2.1. | General | 27 |
| 3.2.2. | Específicos | 27 |
| 3.3. | Requerimientos tecnológicos | 27 |
| 3.3.1. | <i>Hardware</i> | 27 |
| 3.3.2. | <i>Software</i> | 27 |
| 3.4. | Flujo de trabajo de la aplicación | 28 |
| 3.5. | Vista principal | 29 |
| 3.6. | Menú <i>File</i> | 30 |
| 3.7. | Menú <i>Collections</i> | 30 |
| 3.8. | Menú <i>Photomosaic</i> | 33 |
| 4. | Implementación | 34 |
| 4.1. | <i>GUI</i> | 34 |
| 4.1.1. | Componente <i>Frame</i> | 34 |
| 4.1.2. | Componente <i>Panel</i> | 35 |
| 4.1.3. | Componentes <i>Dialogs</i> | 36 |
| 4.2. | Base de datos | 36 |
| 4.2.1. | Semántica de la base de datos | 36 |
| 4.2.2. | Semántica de los vectores característicos | 37 |
| 4.2.3. | Proceso de búsqueda del mejor candidato | 38 |
| 4.3. | Descriptor de colores dominantes | 40 |
| 4.3.1. | Métodos principales | 40 |
| 4.3.2. | <i>CUDA-kernel</i> | 42 |
| 4.4. | Descriptor de histograma de bordes | 42 |
| 4.4.1. | Método principal | 42 |
| 4.4.2. | <i>CUDA-kernel</i> | 44 |
| 4.5. | Descriptor de forma basado en regiones | 44 |
| 4.5.1. | Método principal | 44 |
| 4.5.2. | <i>CUDA-kernel</i> | 46 |
| 4.6. | Medidas de distancia | 46 |
| 4.6.1. | <i>DCD</i> | 46 |
| 4.6.2. | <i>EHD</i> | 47 |
| 4.6.3. | <i>RBSD</i> | 47 |
| 5. | Pruebas y Resultados | 49 |
| 5.1. | Colecciones | 49 |
| 5.2. | Descriptor de colores dominantes | 51 |
| 5.3. | Descriptor de histograma de bordes | 52 |
| 5.4. | Descriptor de forma basado en regiones | 54 |
| 5.5. | Generación de fotomosaicos | 56 |
| 6. | Conclusiones | 60 |
| | Referencias | 61 |

Introducción

Uno de los campos de mayor desarrollo e investigación en el ámbito de la computación gráfica es el procesamiento digital de imágenes, debido a la amplia aplicabilidad que tiene en áreas tales como la medicina, la geografía, la inteligencia artificial, el arte y la publicidad. Estas dos últimas están estrechamente relacionadas debido al alto impacto que tiene la imagen de un producto en su éxito comercial. Es por ello que a través del tiempo se han desarrollado numerosas técnicas que aplicadas a la imagen de un producto, causan un impacto positivo de éste en el mercado.

La aplicación de operadores o filtros puede dar el efecto deseado a una imagen de forma sencilla. Sin embargo, en ocasiones es deseable la generación de una imagen a partir de otras. Un ejemplo de ello son los fotomosaicos, los cuales son composiciones de imágenes que, vistos desde cierta cercanía, permiten apreciar dichas imágenes y desde cierta lejanía, la silueta de una imagen conformada por éstas, teniendo gran impacto en los últimos años en el arte y la publicidad. La desventaja alrededor de éstos radica en el tiempo de ejecución necesario para llevar a cabo su generación.

Existen numerosos trabajos sobre fotomosaicos que exploran las distintas etapas para su generación. Sin embargo, la ausencia de énfasis en la fase de adquisición de las mejores imágenes para su conformación, ha motivado el desarrollo de este trabajo puesto que la selección de los mejores candidatos no es una tarea trivial, dada la gran cantidad de atributos que permiten describir una imagen en términos de su color, textura y forma. Dado que la adquisición de características a partir de imágenes es una tarea computacionalmente costosa, se presenta la necesidad de aprovechar de forma óptima el *hardware* disponible en algunos sistemas, como es el caso de las *GPUs*.

En el primer capítulo del presente trabajo se muestra el proceso de generación de fotomosaicos en sus distintas etapas, mientras que en el segundo capítulo se exponen algunos de los descriptores visuales definidos en el estándar *MPEG-7* y su aplicabilidad para la adquisición de los candidatos óptimos. Por su parte, en el tercer capítulo se presenta la solución desarrollada, describiendo además de forma detallada las consideraciones de implementación en el cuarto capítulo. En el quinto capítulo se expone el conjunto de pruebas y resultados obtenidos a partir del trabajo realizado, presentando finalmente en el sexto capítulo las conclusiones y posibles trabajos futuros.

Capítulo 1

Proceso de generación de fotomosaicos

La generación de fotomosaicos amerita el cumplimiento de una serie de etapas que inherentemente están involucradas en su proceso de generación. En este capítulo, se presentan inicialmente algunos conceptos básicos asociados a los fotomosaicos, presentando posteriormente el flujo de trabajo convencional que debe seguir una aplicación de esta naturaleza, así como también las vías para la adquisición y almacenamiento de las imágenes a utilizar.

1.1. Conceptos básicos

Según Silvers [1] los mosaicos se definen como obras compuestas por piezas de madera, piedra, cerámica, vidrio u otro material, las cuales poseen diversas formas y colores, unidas mediante algún aglomerante, formando composiciones decorativas. De esta definición se desprende que un fotomosaico es entonces un mosaico donde las piezas que lo componen son imágenes.

Las imágenes que componen los fotomosaicos se ordenan en una malla de parches sin que se solapen ni existan vacíos entre ellas, siendo éstas características propias de los teselados como lo indica Tessella Inc. en [2].

La particularidad de los fotomosaicos radica en que, si existe una corta distancia entre éstos y el punto de vista, pueden apreciarse las imágenes que los componen. A medida que esta distancia aumenta, puede percibirse la silueta de una imagen conformada por las imágenes observadas desde una corta distancia.

En la Figura 1.1(b) se expone un fotomosaico generado a partir de la fotografía mostrada en la Figura 1.1(a), utilizando una malla uniforme para organizar las imágenes que lo componen.



(a) Imagen base

(b) Fotomosaico generado

Figura 1.1: Imagen base y su fotomosaico generado, tomados de Slomp et al. [3]

1.2. Flujo de trabajo de una aplicación generadora de fotomosaicos

No hay una metodología adoptada para la creación de una aplicación de esta naturaleza. Sin embargo, a continuación se presentan consideraciones y técnicas asociadas a distintas etapas que inherentemente están involucradas en la generación de fotomosaicos.

1.2.1. Selección de la imagen base

El elemento más importante requerido para la generación de fotomosaicos es la imagen a partir de la cual generarlos. Los fotomosaicos, vistos desde cierta lejanía, deben presentar la silueta de esta imagen. La calidad en su generación está sujeta a la selección de los mejores candidatos a sustituir en cada uno de sus parches.

1.2.2. Generación de los vectores característicos

El vector característico de una imagen es un conjunto de coeficientes obtenidos a partir de una serie de procedimientos aplicados a ésta. Estos procedimientos suelen ser costosos en cuanto a tiempo de ejecución, lo que hace necesario llevar el control de una estructura que almacene esta información, de forma que estos procedimientos sean llevados a cabo una sola vez. En el Capítulo 2 se presentan posibles vectores característicos que pueden ser generados a partir de una imagen, en base a sus atributos de color, textura y forma.

Una de las primeras tareas que debe realizar una aplicación generadora de fotomosaicos es justamente construir los vectores característicos asociados al conjunto de imágenes candidatas que conformarán el fotomosaico a generar. Para ello es necesario llevar a cabo un proceso de adquisición y almacenamiento de éstas como se detalla en las Secciones 1.3 y 1.4. En la Figura 1.2 se ilustra de forma sencilla la generación del vector característico asociado a una imagen candidata, como un sistema con una sola entrada (la imagen candidata) y una sola salida (su vector característico asociado).

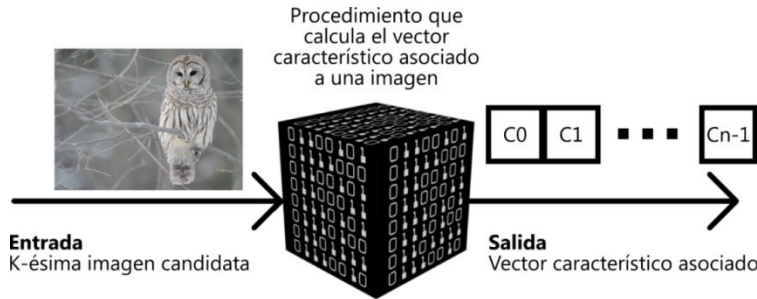


Figura 1.2: Ilustración del proceso de generación del vector de N coeficientes asociado a la k -ésima imagen candidata

1.2.3. Creación de la malla de parches

En el proceso de generación de fotomosaicos la creación de una malla de parches es fundamental, donde cada uno de ellos es reemplazado por alguna de las imágenes candidatas, pudiendo tener una morfología muy variada. Una alternativa corresponde a la utilización de mallas rectangulares uniformes empleadas por Slomp et al. [3]. A su vez, existen variaciones como las mostradas por Di Blasi et al. [4], las cuales poseen distintos niveles de detalle. Ejemplos asociados a estas morfologías se presentan en las Figuras 1.3(a) y 1.3(b).



(a) Fotomosaico con mallado uniforme

(b) Fotomosaico con mallado no uniforme

Figura 1.3: Fotomosaicos tomados del trabajo de Di Blasi et al. [4] con mallado rectangular

Otras morfologías han sido adoptadas para la generación de fotomosaicos tales como formas de rompecabezas empleadas por Kim y Pellacini denominadas *Jigsaw Image Mosaics* [5], tomando como entrada un contenedor y un conjunto de imágenes candidatas con formas arbitrarias, agrupándolas en él según su forma y color. En la Figura 1.4 puede apreciarse un ejemplo de fotomosaico con estas características.



Figura 1.4: Ejemplo de *Jigsaw Image Mosaic*

1.2.4. Selección del mejor candidato para cada parche

Al momento de seleccionar una imagen candidata a ser sustituida en un parche específico, se realiza un análisis de los vectores característicos generados. Luego de realizado, es seleccionado el candidato óptimo bajo algún criterio de distancia o similitud con respecto a la región a sustituir en la imagen base. Estos criterios dependen de la descripción que se realice de la imagen, los cuales son abordados en profundidad en el siguiente capítulo. En la Figura 1.5 se ilustra el proceso de selección del candidato óptimo a sustituir en cada parche de la malla utilizada para la generación del fotomosaico, bajo algún criterio.

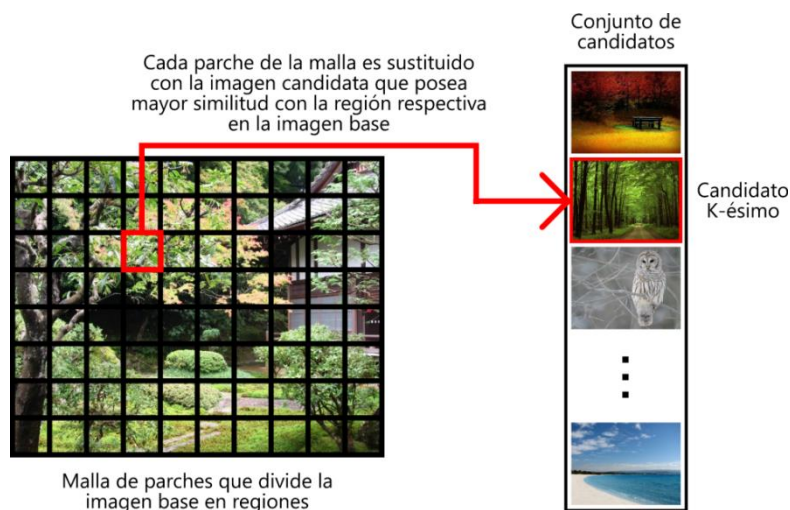


Figura 1.5: Ilustración del proceso de selección del mejor candidato para cada parche

1.2.5. Redimensionamiento de los candidatos

Las imágenes candidatas pueden tener distintas dimensiones, siendo necesario un redimensionamiento de éstas al tamaño del parche respectivo. Existen numerosos algoritmos que han sido desarrollados para el redimensionamiento de imágenes, basados en la interpolación de píxeles.

Una imagen digital es una representación discreta compuesta por muestras distribuidas uniformemente sobre una malla [6]. Cada muestra (o muestras, como en el caso de las imágenes multicanal o a color), se denomina píxel. La interpolación es necesaria para la obtención de nuevos píxeles en coordenadas específicas en base a la información existente.

En la Figura 1.6 se ilustra este proceso, donde los círculos representan los píxeles existentes. La generación de un nuevo píxel en una coordenada específica, viene dada por el símbolo \odot definido por dx y dy . Los círculos de color negro representan los límites de la interpolación en este ejemplo, mediante los cuales el nuevo valor es calculado en la coordenada especificada.

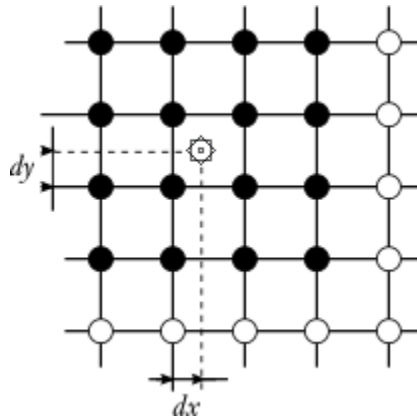


Figura 1.6: Ilustración del proceso de interpolación utilizando 16 vecinos

Interpolación por vecino más cercano

Este algoritmo selecciona el valor del píxel más cercano redondeando las coordenadas del punto de interpolación deseado, como se presenta en la Ecuación 1.1.

$$f(x) = \begin{cases} f([x]) & \text{si } x - [x] < \frac{1}{2} \\ f([x] + 1) & \text{en caso contrario} \end{cases} \quad (1.1)$$

En el caso bidimensional, la fórmula presentada se extiende a dos variables x y y . La operación $[.]$ representa la aplicación de la función parte entera al valor recibido como parámetro. Este método es poco preciso y genera fuertes discontinuidades, especialmente cuando existen rotaciones y escalamientos arbitrarios involucrados. Lo interesante de este algoritmo es que preserva la distribución

del ruido en la imagen transformada, lo cual representa información valiosa en numerosas aplicaciones para el análisis de imágenes.

Interpolación bilineal

Este algoritmo lleva a cabo la interpolación considerando los cuatro píxeles más cercanos, construyendo y evaluando dos funciones de interpolación lineal, correspondientes a cada dirección. En el caso bidimensional, la interpolación bilineal para un punto arbitrario (x, y) viene dada por la Ecuación 1.2.

$$f(x, y) = f_{y_1} + \frac{f_{y_2} - f_{y_1}}{y_2 - y_1} (y - y_1) \quad (1.2)$$

donde

$$\begin{aligned} f_{y_1} &= f_{11} + \frac{f_{21} - f_{11}}{x_2 - x_1} (x - x_1) & f_{y_2} &= f_{12} + \frac{f_{22} - f_{12}}{x_2 - x_1} (x - x_1) \\ x_1 &= \lfloor x \rfloor & x_2 &= \lfloor x \rfloor + 1 & y_1 &= \lfloor y \rfloor & y_2 &= \lfloor y \rfloor + 1 \\ f_{11} &\equiv f(x_1, y_1) & f_{12} &\equiv f(x_1, y_2) & f_{21} &\equiv f(x_2, y_1) & f_{22} &\equiv f(x_2, y_2) \end{aligned}$$

Las principales desventajas asociadas a la interpolación bilineal vienen dadas por la poca preservación de los detalles de la imagen, así como la aparición de artefactos no deseados en imágenes rotadas.

Spline cúbico

Este algoritmo provee resultados aceptables tanto en tiempo de ejecución como en la calidad visual de las imágenes generadas. Actualmente es una de las opciones más utilizadas para la interpolación de píxeles. El algoritmo lleva a cabo la interpolación considerando los 16 píxeles más cercanos. La función que lo define viene dada por la Ecuación 1.3.

$$f(x, a) = \begin{cases} (a + 2)|x|^3 - (a + 3)x^2 + 1 & \text{si } 0 \leq |x| \leq 1 \\ a|x|^3 - 5ax^2 + 8a|x| - 4a & \text{si } 1 < |x| \leq 2 \\ 0 & \text{en caso contrario} \end{cases} \quad (1.3)$$

tal que $-1 \leq a < 0$. El parámetro a del spline cúbico generalmente es fijado en un valor intermedio cercano a $-1/2$.

1.3. Proceso de adquisición de imágenes

Como se ha indicado, la entrada que requiere un sistema generador de fotomosaicos es, además

de la imagen base, un conjunto de imágenes que puede obtenerse fundamentalmente desde un directorio indicado por el usuario o desde *Internet*. Existen repositorios que poseen millones de imágenes de temáticas diversas en múltiples formatos y dimensiones. La mayoría proporciona una *API* (*Application Programming Interface*) para desarrolladores que deseen obtener mayor provecho de estos repositorios con la finalidad de crear una aplicación que haga uso de las funcionalidades que proveen.

Entre los repositorios de imágenes en *Internet* más comúnmente utilizados, pueden destacarse *Flickr* (www.flickr.com), *Tumblr* (www.tumblr.com), *Instagram* (www.instagram.com) y *Photobucket* (www.photobucket.com). Cada uno de ellos posee particularidades acerca de las características de las imágenes que pueden almacenar. Asimismo, existen repositorios que poseen colecciones de imágenes con características específicas para su uso en actividades de investigación, como es el caso del *MIT-Adobe FiveK Dataset* [7].

1.4. Proceso de almacenamiento de imágenes

Existen fundamentalmente dos modelos para el almacenamiento de información de distinta naturaleza, tales como las bases de datos relacionales y las bases de datos *NoSQL* (*Not Only SQL*). En esta sección se presenta una comparación detallada entre ambos modelos.

1.4.1. Bases de datos relacionales

Como lo indica Oracle [8], las bases de datos relacionales presentan la información en tablas, siendo sus filas colecciones de objetos de un mismo tipo. La información contenida en una tabla puede ser recuperada y relacionada con otras tablas de acuerdo a claves comunes. Dispone de un sistema manejador de bases de datos relacionales (*RDBMS*) que gestiona la manera en la cual los datos son almacenados, mantenidos y recuperados.

1.4.2. Bases de datos *NoSQL*

Las bases de datos *NoSQL* surgen como alternativa a las bases de datos relacionales, las cuales permiten de forma rápida y eficiente la organización y el análisis de grandes volúmenes de datos con tipos dispares [9]. Estas bases de datos fueron desarrolladas en respuesta a las necesidades de las aplicaciones modernas que requieren del manejo de gran cantidad de información.

1.4.3. Relacionales vs. *NoSQL*

Las bases de datos relacionales fueron introducidas en los años 70 para proveerle a las aplicaciones, posibilidades de almacenamiento siguiendo un estándar en cuanto al modelo de datos y al lenguaje de consultas. Para ese momento, el almacenamiento era costoso y los esquemas de datos eran relativamente sencillos.

Debido al crecimiento de *Internet* en los últimos años, la cantidad de datos almacenados acerca de usuarios, objetos, productos y eventos se ha visto incrementada de forma drástica, así como también su frecuencia de acceso. Ejemplo de ello son las redes sociales, las cuales generan gran cantidad de datos asociados a las actividades realizadas por sus usuarios.

Las bases de datos relacionales no fueron diseñadas para enfrentar los problemas de escalabilidad y agilidad que presentan las aplicaciones modernas. En el año 2000, surgen las bases de datos *NoSQL* las cuales permiten hacer frente a estas necesidades. La siguiente tabla muestra algunas de las diferencias que ha destacado la compañía 10gen en su sitio web [10] entre las bases de datos relacionales y las *NoSQL*.

| | Bases de datos relacionales | Bases de datos <i>NoSQL</i> |
|--|--|---|
| Tipos | Un solo tipo (<i>SQL</i>) con variaciones menores. | Múltiples tipos: Clave/Valor, basadas en documentos, orientadas a columnas. |
| Modelo de almacenamiento de datos | Registros individuales (e.g. empleados) son almacenados en filas de una tabla, con cada columna almacenando una pieza específica de información acerca del registro (e.g. supervisor, fecha de contratación, etc.), como una hoja de cálculo. Diferentes tipos de registros son almacenados en diferentes tablas y luego unidas, ejecutando consultas más complejas. | Varía en base al tipo de base de datos <i>NoSQL</i> . Por ejemplo, las de tipo Clave/Valor funcionan de forma similar a las bases de datos relacionales, solo que tienen únicamente dos columnas (clave y valor), con información más compleja almacenada en la columna valor. Por otra parte, las basadas en documentos almacenan toda la información relevante en un solo documento en <i>XML</i> , <i>JSON</i> u otro formato, los cuales permiten anidar valores de forma jerárquica. |
| Esquemas | La estructura y los tipos de datos son fijados a priori. Para realizar modificaciones toda la base de datos debe ser alterada, tiempo durante el cual debe estar fuera de servicio. | Suele ser dinámico. Puede añadirse nueva información a los registros en cualquier momento y a diferencia de las tablas de las bases de datos relacionales, datos distintos pueden ser almacenados juntos de ser necesario. En el caso de las bases de datos orientadas a columnas, la dificultad de añadir nuevos campos dinámicamente es mayor. |

| | | |
|----------------------------------|--|---|
| Escalabilidad | Vertical, lo que significa que la capacidad de un servidor puede incrementarse de forma que pueda atender la demanda existente. Es posible utilizar bases de datos relacionales sobre un amplio conjunto de servidores, sin embargo, requiere de un proceso adicional de ingeniería. | Horizontal, lo que significa que un administrador de bases de datos puede añadir mayor capacidad al sistema añadiendo nuevos servidores o instancias en la nube. Las bases de datos <i>NoSQL</i> propagan automáticamente los datos sobre los servidores disponibles cuando es necesario. |
| Manipulación de los datos | Se utiliza un lenguaje específico haciendo uso de sentencias <i>SELECT</i> , <i>INSERT</i> y <i>UPDATE</i> . | A través de <i>APIs</i> orientadas a objetos. |

Capítulo 2

Descriptores visuales de *MPEG-7*

Una de las estructuras de datos más importantes asociadas al proceso de generación de fotomosaicos es el vector característico de las imágenes candidatas. En este capítulo, se presentan algunos de los descriptores definidos en el estándar *MPEG-7* y cómo pueden ser utilizados para la generación de vectores que pueden contener información representativa de las imágenes, en base a sus características tales como el color, la textura y la forma. Se presentan los fundamentos teóricos de estos descriptores, así como sus procesos de extracción y comparación asociados.

2.1. Estándar *MPEG-7*

Establecido en el año 1988, el *Moving Picture Experts Group (MPEG)* [11] ha desarrollado estándares para compresión de contenido digital audiovisual, lo cual ha cambiado la manera en la cual esta clase de contenido es producido por las compañías a través de los distintos canales de distribución y aprovechado por gran variedad de dispositivos.

Uno de los tantos estándares desarrollados por este grupo de expertos ha sido *MPEG-7*, el cual provee descriptores visuales y de audio. Los Esquemas de Descripción Multimedia (*MDS*) proveen descripciones estandarizadas involucrando los descriptores indicados. Por su parte, presentan un lenguaje también estandarizado que permite expresar los esquemas de descripción, denominado Lenguaje de Definición de Descripción (*DDL*). Adicionalmente, provee consideraciones que permiten hacer uso del estándar en entornos reales.

El objetivo principal del estándar *MPEG-7* se centra en proveer descripciones de imágenes, así como también de elementos de audio y video, lo cual contribuye al filtrado y la categorización del contenido por parte de las aplicaciones que hacen uso de éste. Los descriptores definidos por el estándar son utilizados para comparar, filtrar y buscar elementos exclusivamente en base a sus características.

2.2. Descriptores visuales

Los descriptores visuales definidos en el estándar *MPEG-7* describen contenido (e.g. imágenes) de acuerdo a su información visual, el cual puede ser descrito en base a la forma de los objetos presentes, la dimensión de éstos, la textura, el color o incluso el movimiento de los objetos en el caso de videos.

Estos descriptores pueden ser utilizados por motores de búsqueda que los implementen, para buscar o filtrar material visual haciendo uso de medidas de distancia o similitud apropiadas. Asimismo, este tipo de aplicaciones puede requerir de la combinación de descriptores, asignándole una ponderación determinada a cada uno de ellos. Los tipos de descriptores se dividen fundamentalmente en cuatro categorías como se presenta a continuación.

2.2.1. Color

El color es una de las características visuales más utilizadas en motores de búsqueda de imágenes. Esta característica es robusta a cambios en el ángulo de visión, así como también a traslaciones y rotaciones de las regiones de interés. El estándar *MPEG-7* presenta diversos descriptores que representan distintos aspectos asociados al color. En la Sección 2.3 se detalla el descriptor de colores dominantes (*DCD*), utilizado en este trabajo. La Figura 2.1 presenta un conjunto de posibles colores presentes en imágenes.



Figura 2.1.: Conjunto de posibles colores presentes en imágenes

2.2.2. Textura

La textura hace referencia a los patrones visuales que tienen o no propiedades homogéneas y resultan en la presencia de múltiples colores e intensidades en una imagen. Es una propiedad natural de cualquier superficie, como por ejemplo el césped y las paredes hechas de bloques o ladrillos, como se aprecia en la Figura 2.2. Contiene información estructural importante acerca de las superficies y su relación con el entorno. La descripción de texturas en imágenes mediante el estándar *MPEG-7*, es un

medio poderoso para la búsqueda y comparación de imágenes. Al igual que en el caso del color, existen diversos descriptores que permiten obtener información de textura a partir de imágenes. En este trabajo se ha utilizado el descriptor de histograma de bordes (*EHD*), presentado en la Sección 2.4.

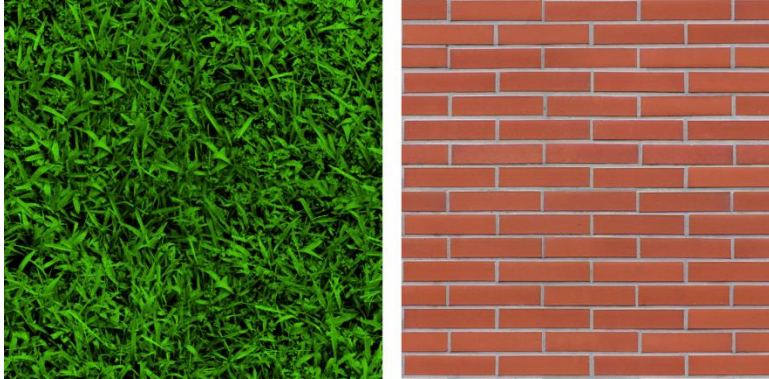


Figura 2.2.: Ejemplo de texturas presentes en el entorno

2.2.3. Forma

En muchas aplicaciones que hacen uso de bases de datos de imágenes, la forma de los objetos en éstas, provee información relevante para su búsqueda y comparación. Algunos objetos con formas elementales se muestran en la Figura 2.3. *MPEG-7* define descriptores basados en regiones y contornos, aplicables a gran cantidad de tareas. Un ejemplo típico es la consulta de imágenes binarias que contienen caracteres o logos. El descriptor de forma basado en regiones (*RBSD*) utilizado en este trabajo, se presenta en la Sección 2.5.

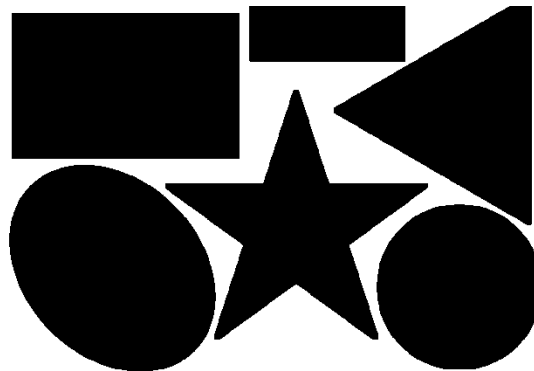


Figura 2.3.: Objetos con formas elementales

2.2.4. Movimiento

Los descriptores de color, textura y forma antes mencionados, pueden ser empleados para la

indexación tanto de imágenes como de secuencias de imágenes o videos. La descripción de características asociadas al movimiento en videos, provee pistas poderosas relacionadas con su contenido. La Figura 2.4 presenta una secuencia de imágenes de una escena. *MPEG-7* ha desarrollado descriptores que permiten capturar características esenciales del movimiento presente en videos de forma concisa y efectiva. Evidentemente, este descriptor no ha sido considerado para la realización de este trabajo.



Figura 2.4.: Secuencia de imágenes de una escena

2.3. Descriptor de colores dominantes

El descriptor de colores dominantes (*DCD*) provee una descripción compacta de los colores representativos en una imagen o en una región de ésta. Su principal uso se centra en la búsqueda por similitud en bases de datos de imágenes de acuerdo a uno o múltiples colores. A diferencia de los descriptores basados en histogramas, los colores representativos son calculados sin necesidad de estar sujetos a un espacio de color, lo que permite obtener una representación precisa y compacta.

De acuerdo a Manjunath et al. [12], el *DCD* se define como un conjunto de duplas como se muestra en la Ecuación 2.1.

$$DCD = \{(c_i, w_i)\} \quad i \in \{0, 1, \dots, k - 2, k - 1\} \quad (2.1)$$

donde k representa el número de colores dominantes, c_i un vector de componentes del espacio de color en cuestión (e.g. un vector de tres dimensiones en el caso *RGB*) y w_i el número de píxeles de la imagen asociados al color dominante c_i , normalizado en el intervalo $[0, 1]$. En consecuencia, $\sum_{\forall i} w_i = 1$.

2.3.1. Proceso de extracción

La extracción de los k colores más representativos de una imagen se realiza mediante el algoritmo k -medias [13], que es un método de agrupamiento por vecindad el cual, parte de un número

determinado de prototipos y un conjunto de ejemplos a agrupar. Este algoritmo busca situar los prototipos en el espacio, de forma que los datos pertenecientes a un mismo prototipo tengan características similares. Todo nuevo ejemplo una vez que los prototipos han sido correctamente situados, es comparado con éstos y asociado a aquel que sea el más próximo, en base a una medida de distancia o similitud previamente elegida. Típicamente, se utiliza la distancia euclídea.

Las regiones se definen minimizando la suma de las distancias cuadráticas entre cada vector de entrada y el centro de su correspondiente clase, representado por el prototipo respectivo. Se debe seleccionar arbitrariamente una partición inicial de forma que cada clase disponga de al menos, un ejemplo.

El procedimiento que sigue el algoritmo es el siguiente:

- Se calcula para cada ejemplo e_i el prototipo más próximo p_i y se incluye en el conjunto de ejemplos E de dicho prototipo, como se muestra en la Ecuación 2.2.

$$E(p_i) = \{e_j\} \quad j \in \{0, 1, \dots, n_i - 2, n_i - 1\} \quad (2.2)$$

donde n_i representa el número de ejemplos asociados al prototipo p_i .

- Se desplaza el prototipo hacia el centro de masa de su conjunto de ejemplos. Este proceso se define mediante la Ecuación 2.3.

$$p_i = \frac{\sum_{j=0}^{n_i-1} e_j}{n_i} \quad (2.3)$$

- El proceso se repite hasta que no existan más desplazamientos de los prototipos.

Mediante este algoritmo el espacio de ejemplos de entrada se divide en k clases o regiones y el prototipo de cada clase se ubica en el centro de masa de la misma. Dichos centros se determinan con el objetivo de minimizar las distancias cuadráticas entre los patrones de entrada y el centro más cercano. Es decir, minimizando el valor de α como se muestra en la Ecuación 2.4.

$$\alpha = \sum_{i=0}^{k-1} \sum_{j=0}^{n_i-1} \beta_{ij} * \left\| p_i - e_j \right\|^2 \quad (2.4)$$

donde el operador $\left\| \cdot \right\|$ representa el cálculo de la distancia euclídea y β_{ij} representa una función de pertenencia del ejemplo j a la región i , tal que toma el valor 1 si el prototipo p_i es el más cercano al ejemplo e_j y 0 en caso contrario. La función de pertenencia se define formalmente mediante la Ecuación 2.5.

$$\beta_{ij} = \begin{cases} 1 & \text{si } \|p_i - e_j\| < \|p_z - e_j\| \quad \forall z \in \{0, 1, \dots, k-2, k-1\} \wedge z \neq i \\ 0 & \text{en caso contrario} \end{cases} \quad (2.5)$$

Debido a la reasignación de los ejemplos a causa del desplazamiento de los prototipos al final de cada iteración, es necesario recalculer los nuevos centros de masa de las regiones. A medida que los prototipos van situándose, los desplazamientos que presentan son cada vez más ligeros hasta desaparecer. La ubicación final de los prototipos representa la solución encontrada por el algoritmo.

La Figura 2.5 ilustra este proceso, donde el número ubicado en la esquina superior izquierda de cada sub-imagen representa la iteración actual del algoritmo, los puntos de color negro representan los ejemplos a agrupar, los símbolos \odot corresponden a los prototipos y las curvas representan las regiones abarcadas por éstos. Los prototipos en la primera iteración, elegidos de forma aleatoria, van mejorando su cobertura a medida que el algoritmo avanza en las iteraciones.

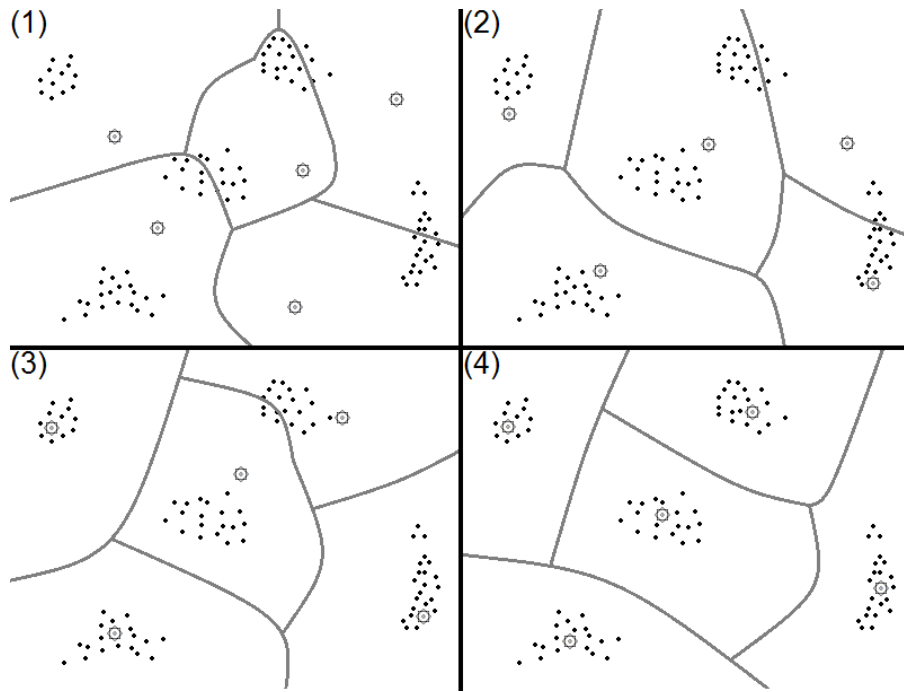


Figura 2.5.: Ejemplo de evolución de los prototipos y grupos formados con el algoritmo k -medias

Una consideración importante es la selección de un valor apropiado de k . La Figura 2.5 muestra un ejemplo ideal, donde se presentan 5 grupos que son capturados por los prototipos a medida que se avanza en las iteraciones. La selección del valor de k como $k = 4$, presentaría un resultado diferente, ya que al menos dos grupos reales acabarían unidos. No obstante, con cinco prototipos podría no existir convergencia y no capturar los cinco grupos reales. Además, la selección de un valor elevado de k causaría que grupos reales sean divididos de forma artificial.

En el contexto del *DCD*, el proceso de obtención de los k colores más representativos en una imagen se denomina cuantización, el cual es ampliamente utilizado no solamente para llevar a cabo reducción de colores en imágenes sino también con fines asociados a la compresión de datos. En este caso, los prototipos representan los colores dominantes de la imagen y los ejemplos, los píxeles de la misma. Además, las ponderaciones involucradas en el *DCD* representan el número de ocurrencias de los colores dominantes en la imagen cuantizada.

2.3.2. Proceso de evaluación

Sergyán [14] propone calcular la distancia entre dos imágenes en base a este descriptor, utilizando el conjunto de duplas generadas por éste, mediante la fórmula presentada en la Ecuación 2.6.

$$D(I^A, I^B) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left| |c_i^A - c_j^B| \right| * |p_i^A - p_j^B| \quad (2.6)$$

donde el operador $|\cdot|$ representa la función valor absoluto, mientras que M y N representan el número de colores dominantes de las imágenes I^A y I^B respectivamente.

2.4. Descriptor de histograma de bordes

El descriptor de histograma de bordes (*EHD*) saca provecho de la distribución espacial de los bordes presentes en una imagen [15]. El estándar *MPEG-7* representa la distribución local de los bordes utilizando un histograma como se presenta en las siguientes secciones.

2.4.1. Partición del espacio de imagen

Para localizar la distribución de los bordes en una región específica de una imagen, el espacio es dividido en 4×4 sub-imágenes como se muestra en la Figura 2.6, generando un histograma que representa la distribución de los bordes en cada una de ellas. Las sub-imágenes son divididas en bloques cuadrados los cuales son clasificados de acuerdo a un tipo de borde específico.

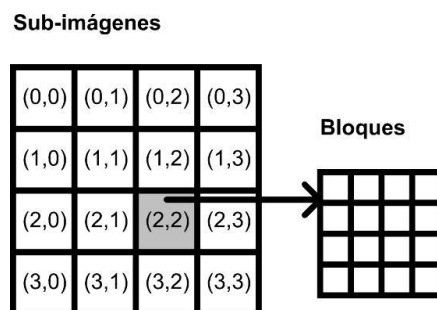


Figura 2.6.: Definición de sub-imágenes y bloques

2.4.2. Tipos de bordes

En la Figura 2.7 se presentan los cinco tipos de bordes definidos por el *EHD*, donde cuatro de ellos son direccionales y uno es no direccional. Los direccionales incluyen bordes verticales, horizontales, diagonales 45° y diagonales 135°, siendo éstas las posibles opciones de clasificación que pueden adoptar los bloques definidos en cada sub-imagen. Si un bloque contiene un borde arbitrario sin direccionalidad en particular, es clasificado como no direccional.

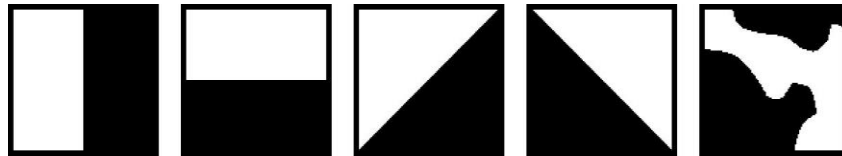


Figura 2.7.: Tipos de bordes considerados por el descriptor

2.4.3. Semántica del histograma local de bordes

Luego de la extracción del tipo de borde de cada bloque, se determina la cantidad de bloques asociados a cada tipo definido. Debido a que se tienen 16 sub-imágenes, se genera un total de $5 \times 16 = 80$ contenedores para representar el histograma que recibe el nombre de histograma local de bordes (*LEH*). Los valores de los contenedores son normalizados en base al número de bloques clasificados, de acuerdo al tipo de borde específico, resultando en valores pertenecientes al intervalo $[0,1]$. La Figura 2.8 presenta la semántica de este descriptor.

| Contenedores | Semántica |
|--------------|--|
| H(0) | Población relativa de bordes verticales en la sub-imagen (0,0) |
| H(1) | Población relativa de bordes horizontales en la sub-imagen (0,0) |
| H(2) | Población relativa de bordes con diagonal de 45° en la sub-imagen (0,0) |
| H(3) | Población relativa de bordes con diagonal de 135° en la sub-imagen (0,0) |
| H(4) | Población relativa de bordes no direccionales en la sub-imagen (0,0) |
| . | . |
| . | . |
| . | . |
| H(75) | Población relativa de bordes verticales en la sub-imagen (3,3) |
| H(76) | Población relativa de bordes horizontales en la sub-imagen (3,3) |
| H(77) | Población relativa de bordes con diagonal de 45° en la sub-imagen (3,3) |
| H(78) | Población relativa de bordes con diagonal de 135° en la sub-imagen (3,3) |
| H(79) | Población relativa de bordes no direccionales en la sub-imagen (3,3) |

Figura 2.8.: Semántica de los contenedores del *LEH*

2.4.4. Proceso de extracción

La extracción de los bordes direccionales y no direccionales es fundamental para el *EHD*. Los bordes no direccionales representan aquellos bordes que no poseen una dirección particular. Los cinco tipos de bordes pueden ser extraídos mediante un esquema en bloques como se indicó previamente, obteniendo la información de los bordes a partir de ellos. Cada sub-imagen es dividida en bloques no solapados, permitiendo así la extracción de la información de borde de cada uno de ellos.

Independientemente de la dimensión de la imagen, ésta es dividida en un número pre-determinado de bloques, existiendo una dependencia entre sus dimensiones y la resolución de la imagen. Las Ecuaciones 2.7 y 2.8 definen la dimensión de los bloques para una imagen de dimensión $M \times N$. Se busca que la dimensión de los bloques sea múltiplo de 2, ignorando los píxeles de los extremos cuando no lo sea.

$$\alpha = \sqrt{\frac{M \times N}{K}} \quad (2.7)$$

$$\beta = \left\lfloor \frac{\alpha}{2} \right\rfloor \times 2 \quad (2.8)$$

donde K representa la cantidad deseada de bloques en cada sub-imagen y β es la dimensión del bloque en cuestión. El tipo de borde es extraído de los bloques siguiendo el esquema presentado en la Figura 2.9, dividiéndolos en cuatro sub-bloques y calculando el valor medio de la luminancia de cada uno de ellos.

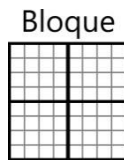


Figura 2.9.: División de un bloque en cuatro sub-bloques

El valor medio de la luminancia de los sub-bloques es utilizado para la detección de bordes, aplicando una operación de convolución sobre éstos, utilizando los filtros definidos en la Figura 2.10, obteniendo así las magnitudes de los bordes.

| | | | | | | | | | |
|---|----|----|----|------------|-------------|-------------|------------|----|----|
| 1 | -1 | 1 | 1 | $\sqrt{2}$ | 0 | 0 | $\sqrt{2}$ | 2 | -2 |
| 1 | -1 | -1 | -1 | 0 | $-\sqrt{2}$ | $-\sqrt{2}$ | 0 | -2 | 2 |

Figura 2.10.: Filtros utilizados para la detección de bordes

Utilizando estos filtros se obtiene un conjunto de cinco magnitudes por cada uno de los sub-bloques del bloque (i, j) , las cuales son obtenidas de acuerdo a las Ecuaciones 2.9 a 2.13.

$$m_v(i, j) = | \sum_{k=0}^3 s_k(i, j) f_v(k) | \quad (2.9)$$

$$m_h(i, j) = | \sum_{k=0}^3 s_k(i, j) f_h(k) | \quad (2.10)$$

$$m_{d45}(i, j) = | \sum_{k=0}^3 s_k(i, j) f_{d45}(k) | \quad (2.11)$$

$$m_{d135}(i, j) = | \sum_{k=0}^3 s_k(i, j) f_{d135}(k) | \quad (2.12)$$

$$m_{nd}(i, j) = | \sum_{k=0}^3 s_k(i, j) f_{nd}(k) | \quad (2.13)$$

donde $s_k(i, j)$ es el promedio de los píxeles en escala de grises del sub-bloque k del bloque (i, j) y $f_{v|h|d45|d135|nd}(k)$ representa el valor del filtro respectivo en la posición k .

Si la mayor de las magnitudes obtenidas es superior a un umbral, el bloque es clasificado con la dirección asociada a dicha magnitud. Por su parte, si la mayor de las magnitudes obtenidas es menor que el umbral establecido, el bloque no es clasificado en ninguna de las cinco categorías. La Ecuación 2.14 define esta selección.

$$\max\{m_v, m_h, m_{d45}, m_{d135}, m_{nd}\} > \psi \quad (2.14)$$

donde ψ representa el umbral establecido.

2.4.5. Histogramas global y semi-global de bordes

El estándar *MPEG-7* define el uso del *LEH* únicamente. Sin embargo, el uso de información local puede no ser suficiente para una descripción correcta de la imagen en cuanto a sus características de textura. El uso de dos histogramas adicionales permite incrementar el rendimiento del descriptor para hacer frente a esta problemática. El primero de ellos es el histograma global de bordes (*GEH*), el cual utiliza información de todo el espacio de imagen, mientras que el segundo es el histograma semi-global de bordes (*SGEH*), que considera distintas distribuciones de los bordes en todo el espacio.

El *GEH* posee únicamente cinco contenedores asociados a los cinco tipos de bordes definidos, a diferencia del *SGEH* que define trece posibles grupos de sub-imágenes como se ilustra en la Figura 2.11. Para cada grupo se consideran también los distintos tipos de bordes, dando como resultado $13 \times 5 = 65$ contenedores para este histograma. Finalmente, se tiene un total de $80(\text{local}) + 5(\text{global}) + 65(\text{semiglobal}) = 150$ contenedores que describen a una imagen en términos de textura.

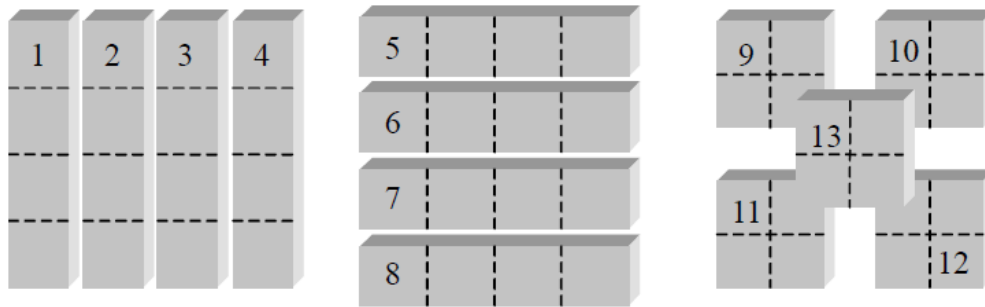


Figura 2.11.: Grupos de sub-imágenes considerados por el *SGEH*

Los valores de los contenedores del *GEH* y el *SGEH* pueden ser obtenidos directamente del *LEH* sin necesidad de llevar a cabo procesamiento adicional sobre la imagen. Además, los trece grupos considerados por el *SGEH* en la Figura 2.11, representan la distribución de bordes en áreas de grandes dimensiones. En particular, los grupos del 1 al 4 enfatizan la conectividad vertical de los bordes. De forma análoga ocurre con el resto de los grupos, de acuerdo a su morfología.

2.4.6. Proceso de evaluación

Dadas dos imágenes, puede determinarse la distancia entre ellas en base a este descriptor, utilizando los valores de los tres histogramas hallados, mediante la fórmula presentada en la Ecuación 2.15.

$$D(I^A, I^B) = \sum_{i=0}^{79} |H_A^L(i) - H_B^L(j)| + \omega \sum_{i=0}^4 |H_A^G(i) - H_B^G(j)| + \sum_{i=0}^{64} |H_A^{SG}(i) - H_B^{SG}(j)| \quad (2.15)$$

donde H^L , H^G y H^{SG} representan los histogramas *LEH*, *GEH* y *SGEH* respectivamente, obtenidos a partir de las imágenes I^A y I^B consideradas. Dado que el número de contenedores del *GEH* es relativamente menor al del *LEH* y *SGEH*, un factor ω es aplicado en la ecuación presentada, típicamente, $\omega = 5$. En la Figura 2.12 se presenta la semántica total de los histogramas considerados en el proceso de evaluación.

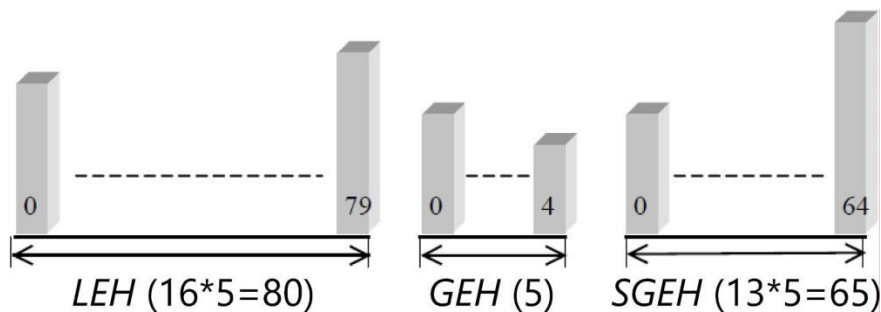


Figura 2.12.: Semántica total de los histogramas *LEH*, *GEH* y *SGEH*

2.5. Descriptor de forma basado en regiones

El descriptor de forma basado en regiones (*RBSD*) representa la distribución de píxeles dentro de un objeto o región en \mathbb{R}^2 . Como está basado tanto en los bordes de los objetos así como en sus píxeles internos, puede describir objetos complejos que posean múltiples regiones discontinuas, así como también objetos simples con o sin agujeros. Este descriptor pertenece a la amplia variedad de técnicas basadas en momentos. Hace uso de la transformada radial angular (*ART*) compleja definida en \mathbb{R}^2 , sobre un disco unitario en coordenadas polares.

2.5.1. Coeficientes de la transformada radial angular

Hosny presenta en su trabajo [16] que los coeficientes de la *ART* de orden p y q de una función de intensidades $f(x, y)$ están definidos por la proyección de la imagen de entrada en las funciones de base de la *ART*, mediante la Ecuación 2.16.

$$F_{pq} = \int_0^{2\pi} \int_0^1 V_{pq}^*(r, \theta) f(r, \theta) r dr d\theta \quad (2.16)$$

La función de base de la *ART* $V_{pq}^*(r, \theta)$ de orden p y q representa polinomios ortogonales contínuos definidos en coordenadas polares sobre un disco unitario, mientras que el asterisco representa su conjugada compleja. Estos polinomios son expresados de forma separable en sus componentes radial y angular, como se presenta en la Ecuación 2.17.

$$V_{pq}(r, \theta) = R_p(r) A_q(\theta) \quad (2.17)$$

donde p y q son enteros no negativos. Las funciones de base radial y angular se definen mediante las Ecuaciones 2.18 y 2.19, siendo $i = \sqrt{-1}$ la unidad imaginaria.

$$R_p(r) = 2\cos(\pi pr) \quad (2.18)$$

$$A_q(\theta) = \frac{1}{2\pi} e^{iq\theta} \quad (2.19)$$

Una de las características más importantes de la *ART* es su invariabilidad a las rotaciones. Las magnitudes de sus coeficientes no se ven afectadas por funciones aplicadas a una imagen antes y después de una rotación. La imagen original es representada por la función de intensidades $f(r, \theta)$. Una rotación en sentido antihorario de acuerdo a un ángulo α resulta en una función de intensidades modificada $g(r, \theta) = f(r, \theta - \alpha)$. En consecuencia, los coeficientes de las imágenes original y rotada, son F_{pq} y $G_{pq} = e^{-iq\alpha} F_{pq}$. En efecto, las magnitudes obtenidas son las mismas, donde $\left| e^{-iq\alpha} \right| = 1$. Gracias a esta propiedad, la *ART* puede ser utilizada como un descriptor de forma por el estándar *MPEG-7*.

2.5.2. Proceso de discretización

La aproximación de los coeficientes de la ART, basada en una correspondencia de forma circular a cuadrada donde las funciones de base están definidas en coordenadas polares sobre un disco unitario, generalmente es definida en coordenadas cartesianas.

Estas correspondencias producen errores de tipo geométrico y numérico. La propagación de estos errores durante los cálculos degrada la precisión de los coeficientes encontrados. Los coeficientes de la ART de orden p y q para una función de intensidades $f(x, y)$, son calculados mediante el orden de aproximación cero, donde la integral doble definida en la Ecuación 2.16 es reemplazada por sumas y la imagen es normalizada dentro del disco unitario. La aproximación de los coeficientes de la ART viene dada entonces por la Ecuación 2.20.

$$\dot{F}_{pq} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} 2 \cos(\pi p r_{xy}) e^{-iq\theta_{xy}} f(x, y) \quad (2.20)$$

donde $r_{xy} = \sqrt{x^2 + y^2}$ y $\theta_{xy} = \arctan\left(\frac{x}{y}\right)$.

2.5.3. Correspondencia en coordenadas polares

El cálculo de los coeficientes de la ART en coordenadas polares se centra en la división del disco unitario en sectores circulares no solapados. Para una imagen de dimensión $N \times N$ como la mostrada en la Figura 2.13, se emplea una nueva correspondencia donde todas las operaciones a realizar se llevan a cabo en coordenadas polares.

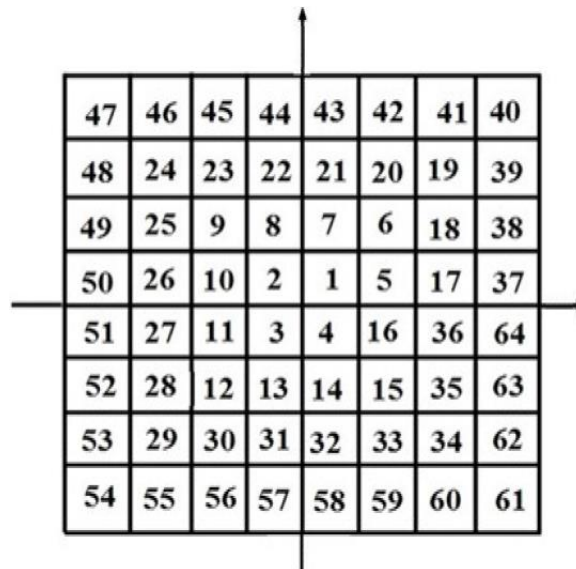


Figura 2.13.: Píxeles en coordenadas cartesianas

En este esquema, el número de píxeles de la imagen de entrada en coordenadas cartesianas es igual al número de sectores circulares en la nueva correspondencia, como se presenta en la Figura 2.14.

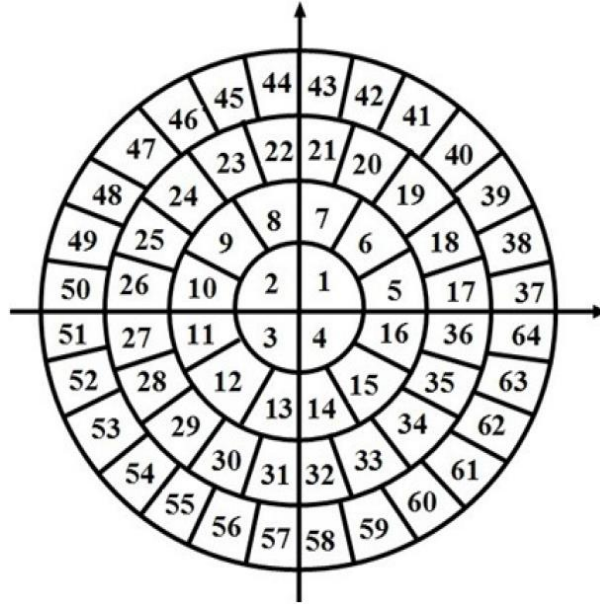


Figura 2.14.: Correspondencia de píxeles en coordenadas polares

La correspondencia puede obtenerse mediante la aplicación de los siguientes pasos:

- El disco unitario es dividido en $\frac{N}{2}$ anillos concéntricos y no solapados.
- Cada anillo es dividido en $8i + 4$ sectores circulares, donde $i \in \{0, 1, \dots, \frac{N}{2} - 2, \frac{N}{2} - 1\}$ siendo $i = 0$ el índice asociado al anillo más interno.
- Los diferentes valores del ángulo θ pueden ser obtenidos mediante la Ecuación 2.21.

$$\theta_{ij} = 2\pi \frac{j+0.5}{8i+4} \quad (2.21)$$

donde $j \in \{0, 1, \dots, 8i + 2, 8i + 3\}$.

2.5.4. Cálculo aproximado de los coeficientes

Los coeficientes de la ART en \mathbb{R}^2 de orden p y q vienen dados por la Ecuación 2.22.

$$F_{pq} = \frac{1}{2\pi} \sum_{\forall i} \sum_{\forall j} \hat{f}(r_i, \theta_{ij}) H_{pq}(r_i, \theta_{ij}) \quad (2.22)$$

siendo H_{pq} , I_p , I_q representadas mediante las Ecuaciones 2.23 a 2.25.

$$H_{pq}(r_i, \theta_{ij}) = I_p(r_i)I_q(\theta_{ij}) \quad (2.23)$$

$$I_p(r_i) = \int_{U_i}^{U_{i+1}} 2 \cos(\pi pr) r dr \quad (2.24)$$

$$I_q(\theta_{ij}) = \int_{V_{ij}}^{V_{ij+1}} e^{-iq\theta} d\theta \quad (2.25)$$

donde $U_{i+1} = \frac{2i+1}{N}$, $U_i = \frac{2i}{N}$, $V_{ij+1} = \theta_{ij} + \frac{\pi}{8i+4}$ y $V_{ij} = \theta_{ij} - \frac{\pi}{8i+4}$. Una vez calculada la integral expuesta en la Ecuación 2.24, se procede a su evaluación como se presenta en la Ecuación 2.26.

$$I_p(r_i) = \left(\frac{2r \sin(\pi pr)}{\pi p} + \frac{2 \cos(\pi pr)}{\pi^2 p^2} \right) \Big|_{U_i}^{U_{i+1}} \quad (2.26)$$

Asimismo, una vez calculada la integral expuesta en la Ecuación 2.25, se procede a su evaluación como se muestra en la Ecuación 2.27.

$$I_q(\theta_{ij}) = \left(\frac{e^{-iq\theta}}{-iq} \right) \Big|_{V_{ij}}^{V_{ij+1}} \quad (2.27)$$

Según Xin et al. [17] la función $\hat{f}(r_i, \theta_{ij})$ es deducida a partir de la función original o imagen de entrada, llevando a cabo un proceso de interpolación como se presenta en la Ecuación 2.28.

$$\hat{f}(r_i, \theta_{ij}) = \sum_{a=k-1}^{k+2} \sum_{b=l-1}^{l+2} f(x_a, y_b) h\left(\frac{r_i \cos(\theta_{ij}) - x_a}{\Delta}\right) h\left(\frac{r_i \sin(\theta_{ij}) - y_b}{\Delta}\right) \quad (2.28)$$

donde $k = \left\lceil \frac{r_i \cos(\theta_{ij})}{\Delta} \right\rceil + \frac{N}{2}$, $l = \left\lceil \frac{r_i \sin(\theta_{ij})}{\Delta} \right\rceil + \frac{N}{2}$ y $\Delta = \frac{2}{N}$ representa el ancho del píxel.

La función $f(x_a, y_b)$ está definida en $[-1, 1]^2$ y representa una correspondencia a partir de la función original $F(a, b)$ definida en el espacio de imagen tal que $f(x_a, y_b) = F(a, b)$, donde $a, b \in \{0, 1, \dots, N-2, N-1\}$, $x_a = \frac{2a-N-1}{N}$ y $y_b = \frac{2b-N-1}{N}$. Además, la función de núcleo definida en \mathbb{R} viene dada por un spline cúbico, como se define en la Ecuación 2.29.

$$h(x) = \begin{cases} \frac{3}{2}|x|^3 - \frac{5}{2}x^2 + 1 & \text{si } |x| \leq 1 \\ -\frac{1}{2}|x|^3 + \frac{5}{2}x^2 - 4|x| + 2 & \text{si } 1 < |x| \leq 2 \\ 0 & \text{en caso contrario} \end{cases} \quad (2.29)$$

2.5.5. Proceso de evaluación

Dadas dos imágenes, puede determinarse la distancia entre ellas en base a este descriptor, utilizando las magnitudes de los coeficientes de la *ART* obtenidos, mediante la fórmula presentada en la Ecuación 2.30.

$$D(I^A, I^B) = \sum_{\forall i} |\lambda_A(i) - \lambda_B(i)| \quad (2.30)$$

donde λ_A y λ_B representan las magnitudes de los coeficientes de la *ART* de las imágenes I^A y I^B respectivamente. Típicamente, $0 \leq p < 3$ y $0 \leq q < 12$, normalizando todas las magnitudes en base a la de orden $p = 0$ y $q = 0$, dando como resultado 35 magnitudes por cada imagen.

Capítulo 3

Planteamiento de la solución

Una vez expuestos los fundamentos teóricos asociados al proceso de generación de fotomosaicos y a los descriptores visuales definidos en el estándar *MPEG-7*, se procede en este capítulo a exponer el planteamiento de la solución, considerando la justificación de la misma, los objetivos trazados, así como también los requerimientos tecnológicos necesarios para el cumplimiento de los objetivos, presentando finalmente el flujo de trabajo de la aplicación y las funcionalidades asociadas a las principales vistas desarrolladas. Las consideraciones de implementación de las funcionalidades desarrolladas son explicadas en profundidad en el siguiente capítulo.

3.1. Justificación

Existen numerosas investigaciones que abordan el proceso de generación de fotomosaicos, pero la ausencia de énfasis en una etapa temprana de gran importancia como la adquisición de las mejores imágenes candidatas, ha motivado la realización de este trabajo.

La generación de vectores característicos que representen de forma óptima a una imagen no es una tarea trivial. Es posible obtener gran cantidad de información a partir de imágenes. Sin embargo, no siempre la información obtenida es útil para la toma de decisiones. Los descriptores visuales definidos en el estándar *MPEG-7* permiten obtener información relevante a partir de imágenes en base a sus características tales como el color, la textura y la forma.

La gran cantidad de operaciones involucradas en la obtención de características mediante estos descriptores, hace necesaria su aceleración. Por ello se ha optado por la paralelización de ciertas tareas asociadas a éstos en la *GPU*, dadas las prestaciones que presenta esta clase de tecnologías.

3.2. Objetivos

A continuación se presenta una serie de objetivos que han sido cumplidos para satisfacer las necesidades planteadas en la sección anterior, considerando además el resto de las fases involucradas en el proceso de generación de fotomosaicos.

3.2.1. General

Desarrollar una aplicación que permita generar fotomosaicos haciendo uso de descriptores visuales definidos en el estándar *MPEG-7*, obteniendo así información asociada a sus características de color, textura y forma, con el fin de poder seleccionar los mejores candidatos para su generación.

3.2.2. Específicos

- Proveer una interfaz gráfica de usuario (*GUI*) que permita llevar a cabo de forma sencilla todas las tareas asociadas a la aplicación.
- Implementar descriptores visuales que permitan obtener características de color, textura y forma, haciendo uso de la *GPU* para acelerar el proceso.
- Generar fotomosaicos haciendo uso de los descriptores visuales considerados para la selección de los mejores candidatos.

3.3. Requerimientos tecnológicos

Para el desarrollo de este trabajo ha sido necesario disponer de un sistema que cumpla los requerimientos tecnológicos mínimos de hardware y software que se presentan a continuación.

3.3.1. Hardware

- PC convencional con al menos 4 *GB* de memoria *RAM*.
- Tarjeta gráfica con soporte para *CUDA* versión 5.0 o superior.

3.3.2. Software

- Sistema operativo *Windows 7* o *Windows 8*.
- *Microsoft Visual Studio 2010* o superior como entorno de desarrollo.
- *MongoDB 2.0* o superior como sistema manejador de bases de datos *NoSQL*.

- Biblioteca *OpenCV* 3.0 para la manipulación de imágenes (*back-end*).
- *wxWidgets* 3.0 como biblioteca para el desarrollo de la *GUI* (*front-end*).

3.4. Flujo de trabajo de la aplicación

En la Figura 3.1 se presenta el flujo de trabajo de la aplicación, desde el punto de vista del usuario. Una vez iniciada, el usuario puede seleccionar un directorio de imágenes y generar los vectores característicos de acuerdo a los descriptores visuales considerados, almacenándolos en la base de datos (1).

Luego, puede seleccionar la imagen base a utilizar para generar el fotomosaico, indicando las características deseadas en cuanto a dimensiones y número de parches en la malla a generar (2). Finalmente, el fotomosaico es generado y desplegado, permitiéndole al usuario guardarlo en diversos formatos para uso futuro (3).

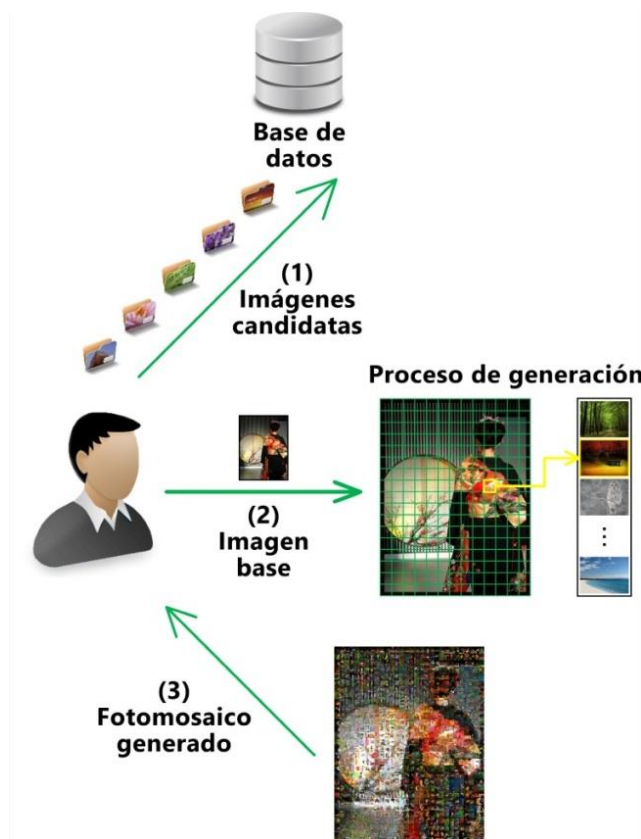


Figura 3.1.: Flujo de trabajo de la aplicación desde el punto de vista del usuario

3.5. Vista principal

De acuerdo a la *GUI* desarrollada en este trabajo, presentada en la Figura 3.2, el proceso de generación de fotomosaicos puede realizarse llevando a cabo las siguientes acciones en la vista principal:

- Seleccionar a través del menú *File*, la imagen base a utilizar para generar el fotomosaico, la cual es cargada en el *panel* principal.
- Generar o seleccionar la colección de imágenes deseada a través del menú *Collections*, así como también eliminar alguna.
- Generar el fotomosaico indicando sus dimensiones y la de los parches, mediante el menú *Photomosaic*, permitiendo también su guardado.
- Obtener a través del menú *About*, la información asociada a la aplicación y a los desarrolladores de la misma.

En las siguientes secciones se presentan con detalle las opciones ofrecidas por la aplicación en la interfaz, que permiten llevar a cabo cada una de las funcionalidades consideradas en este trabajo.

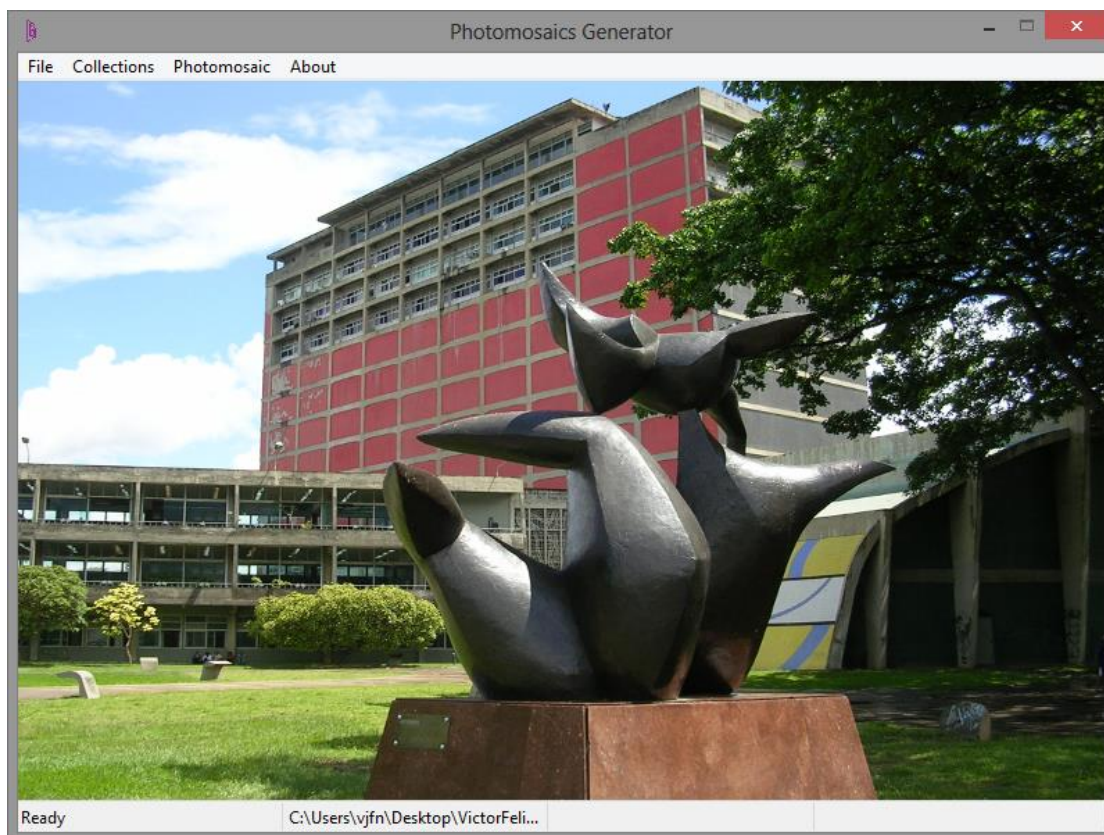


Figura 3.2.: Vista principal de la aplicación con una imagen base de ejemplo desplegada en ella

3.6. Menú *File*

Este menú permite al usuario realizar tareas elementales tales como la selección de la imagen base y abandonar la aplicación. El usuario puede seleccionar una imagen para ser utilizada por la aplicación como la base sobre la cual generar el fotomosaico. Dicha imagen es desplegada en el *panel* de la vista principal.

Debido a que la aplicación maneja *wxWidgets* como *front-end* de la aplicación y *OpenCV* como *back-end*, debe existir una correspondencia entre las estructuras de datos para el manejo de imágenes, provistas por estas dos bibliotecas. Es por ello que se definen métodos que permiten establecer esta correspondencia tal que una imagen cargada en la interfaz gráfica mediante *wxWidgets* (imagen base), sea convertida a una estructura de datos de *OpenCV* para llevar a cabo todo el procesamiento interno, y luego reconvertida nuevamente a *wxWidgets* para su despliegue en la interfaz (fotomosaico generado).

Además, mediante este menú el usuario puede abandonar la aplicación. Esta funcionalidad destruye todo el contexto creado por *wxWidgets*, permitiendo la culminación de la aplicación de forma apropiada, eliminando todas las estructuras de datos creadas durante la ejecución [18]. Previo al abandono de la aplicación, se verifica si ha sido generado un fotomosaico y si éste ha sido guardado por el usuario.

3.7. Menú *Collections*

Mediante este menú, el usuario puede generar nuevas colecciones para ser utilizadas al momento de generar fotomosaicos. La ejecución de esta funcionalidad es costosa en cuanto a tiempo de ejecución debido a que por cada una de las imágenes ubicadas en el directorio seleccionado por el usuario, deben calcularse los vectores característicos de color, textura y forma. La inserción en la base de datos requiere del nombre elegido por el usuario para la colección, el directorio donde se ubican las imágenes y los vectores característicos calculados.

En la barra de estado de la aplicación ubicada en la región inferior de la ventana, se indica la imagen que está siendo procesada en un instante determinado, así como también un mensaje cuando el proceso ha finalizado. Como se hallan vectores por cada tipo de característica, los descriptores visuales llevan a cabo una serie de acciones que se exponen a continuación.

Caso *DCD*

Este descriptor hace uso del algoritmo *k*-medias para el cálculo del vector característico de color, cuantizando una imagen a *k* colores (los más representativos), lo cual es realizado en la *GPU* gracias a la independencia entre los cálculos que son necesarios llevar a cabo, así como también hallar la frecuencia de aparición de cada uno de ellos en la imagen cuantizada.

Los dispositivos de despliegue utilizados por los computadores modernos generalmente siguen

el espacio de color *RGB*. Sin embargo, la distancia entre dos colores pertenecientes a este espacio no es proporcional a la distancia de la percepción humana. Por esta razón, la imagen recibida por este descriptor es transformada al espacio de color *CIE-LUV* con el fin de que los vectores característicos de color calculados sean más precisos [14].

La información generada por este descriptor, es decir, el conjunto de k colores y su frecuencia de aparición en la imagen cuantizada, es almacenado en la base de datos como el vector característico de color de la imagen actual, junto con su nombre. La Ecuación 3.1 presenta la estructura de este vector.

$$vector_{DCD} = (l_0, u_0, v_0, p_0, \dots, l_{k-1}, u_{k-1}, v_{k-1}, p_{k-1}) \quad (3.1)$$

donde l_i , u_i y v_i representan las componentes del color dominante i en el espacio de color *CIE-LUV* y p_i viene dado por la frecuencia de dicho color en la imagen cuantizada.

Caso *EHD*

Este descriptor busca dividir el espacio de imagen como se presentó en la Figura 2.6, considerando la noción de sub-imagen y bloque, recibiendo como parámetro únicamente la imagen considerada.

La primera tarea que lleva a cabo este descriptor es la conversión de la imagen de entrada a escala de grises, definición posteriormente los filtros expuestos en la Figura 2.10, para luego obtener las cinco magnitudes por cada bloque de cada sub-imagen, definidas mediante las Ecuaciones 2.9 a 2.13 presentadas en el Capítulo 2. Debido a que los cálculos por bloque son independientes entre sí, son paralelizados en la *GPU* para acelerar su obtención. El cálculo de estas magnitudes permite asignar un tipo de borde a cada bloque, de acuerdo al procedimiento expuesto en la sección 2.4.4.

La generación del *LEH* es llevada a cabo también en la *GPU*, utilizando la información obtenida de cada uno de los bloques de las sub-imágenes. El número de bloques de cada sub-imagen con un tipo de borde asociado también es calculado, ya que es utilizado para llevar a cabo el proceso de normalización del histograma.

Dado que el *LEH* contiene la información de bordes asociada a cada sub-imagen, es posible hallar el *GEH* y el *SGEH* a partir de éste ya que, en el caso del *GEH*, se acumulan los valores del *LEH* asociados a todas las sub-imágenes, mientras que el *SGEH* considera las distribuciones de sub-imágenes presentadas en la Figura 2.11.

El vector característico asociado a este descriptor se presenta en la Ecuación 3.2, el cual está compuesto por los valores de los contenedores del *LEH*, *GEH* y *SGEH*, una vez normalizados. La Ecuación 3.2 presenta la estructura de este vector.

$$vector_{EHD} = (leh_0, \dots, leh_{x-1}, geh_0, \dots, geh_{y-1}, sgeh_0, \dots, sgeh_{z-1}) \quad (3.2)$$

donde leh_i , geh_i y $sgeh_i$ representan el contenedor i de los histogramas LEH , GEH y $SGEH$, de dimensiones x , y y z respectivamente.

Caso *RBSD*

El tiempo requerido para el cálculo de la discretización de la *ART* presentada en la Ecuación 2.20 es sumamente elevado. Por ello el primer paso que se lleva a cabo es el redimensionamiento de la imagen de entrada a un valor fijado en 100×100 píxeles, considerando que el descriptor trabaja sobre imágenes cuadradas y de cantidad par de píxeles por dimensión.

La imagen es convertida a escala de grises y posteriormente binarizada haciendo uso del umbral obtenido mediante el algoritmo de *Otsu* [19]. El proceso de binarización es aplicado en la *GPU* dada la independencia entre los píxeles al momento de decidir su valor binario en base al umbral.

Los coeficientes de la *ART* dados por la Ecuación 2.22 son calculados en la *GPU* gracias a la independencia en el cálculo de los coeficientes de distinto orden de la transformada, mientras que la magnitud de éstos es calculada en la *CPU*. Como se trata de coeficientes complejos, el cálculo de su magnitud viene dado por la Ecuación 3.3.

$$m = \sqrt{a^2 + b^2} \quad (3.3)$$

donde a y b corresponden a los coeficientes de un número complejo de la forma $a + bi$, siendo estas magnitudes las componentes del vector característico asociado a este descriptor, como se presenta en la Ecuación 3.4.

$$vector_{RBSD} = (m_{0,1}, \dots, m_{p,q-1}, m_{p,q}) \quad (3.4)$$

donde $m_{p,q}$ representa la magnitud del coeficiente de orden p y q .

Mediante este menú, el usuario también puede seleccionar una colección específica para ser utilizada al momento de generar el fotomosaico. Las imágenes utilizadas durante el proceso de generación, corresponden a las ubicadas en el directorio asociado a la colección seleccionada.

Además, puede eliminar una colección, lo que conlleva a la eliminación de toda su información almacenada en la base de datos, incluyendo su nombre, directorio y los vectores característicos de color, textura y forma de cada una de sus imágenes asociadas. La eliminación de todas las colecciones de la base de datos también es posible, siendo necesaria la creación de al menos una para su utilización durante el proceso de generación de fotomosaicos.

3.8. Menú *Photomosaic*

El usuario puede llevar a cabo mediante este menú la generación del fotomosaico indicando una serie de parámetros que se presentan a continuación, permitiéndole además su posterior guardado para uso futuro, en diversos formatos, especificando mediante un diálogo, nombre, formato y ruta de guardado deseados. Los formatos disponibles para el proceso de guardado son *BMP*, *JPEG*, *TIFF* y *PNG*.

Los parámetros para la generación del fotomosaico son indicados por el usuario mediante este menú, entre los que destacan las dimensiones deseadas y la ponderación de los descriptores. A continuación se presentan los parámetros indicados:

- Ancho de los parches del fotomosaico a generar
- Ancho del fotomosaico en cuanto a número de parches
- Alto del fotomosaico en cuanto al número de parches
- Ponderación del *DCD* en la decisión de selección del mejor candidato para cada parche
- Ponderación del *EHD* en la decisión de selección del mejor candidato para cada parche
- Ponderación del *RBSD* en la decisión de selección del mejor candidato para cada parche
- Número máximo de veces que una imagen puede ser seleccionada como mejor candidata durante el proceso de generación

El alto de los parches del fotomosaico a generar no es solicitado al usuario ya que se calcula en base al resto de los parámetros, de forma que el fotomosaico generado mantenga el radio aspecto de la imagen base. Además, las ponderaciones de los descriptores visuales deben pertenecer al intervalo $[0,1]$, siendo su suma igual a 1.

Una vez seleccionada la colección a utilizar y establecidos los parámetros expuestos, se procede a calcular el alto de los parches del fotomosaico a generar, para tener completamente definida toda la información requerida asociada a la malla de parches.

Se procede entonces a calcular los vectores característicos de cada una de las regiones de la imagen base y se determinan sus mejores candidatos presentes en la colección de imágenes seleccionada, los cuales son redimensionados apropiadamente para cubrir completamente las regiones definidas por los parches respectivos. Finalmente, el fotomosaico es desplegado en el *panel* de la vista principal, sustituyendo a la imagen base.

Capítulo 4

Implementación

En este capítulo se presentan de forma detallada las funcionalidades implementadas más importantes, desglosándolas de acuerdo al tipo de requerimiento que satisfacen, presentando además algunos códigos simplificados que ilustran las tareas que realizan.

4.1. *GUI*

La aplicación desarrollada utiliza la biblioteca *wxWidgets* para la *GUI*, la cual define un conjunto de clases que permiten el despliegue de elementos de interfaz como *frames*, *panels* y *dialogs*, así como también permite manejar los eventos llevados a cabo por el usuario a través de dispositivos de entrada tales como *mouse* y *keyboard*.

4.1.1. Componente *Frame*

En principio, la aplicación define una clase que hereda de *wxApp*, que es la clase utilizada por *wxWidgets* como base para el desarrollo de aplicaciones utilizando esta herramienta. Luego, se define una clase que hereda de *wxFrame*, la cual representa la ventana principal de la aplicación.

En esta ventana se define el conjunto de elementos que conforman la vista principal tales como *menu bar*, *status bar* y *panel* contenedor de la imagen base o fotomosaico generado según sea el caso. En la Figura 4.1 se presentan estas clases.

Asimismo, se definen los eventos asociados al componente de interfaz *menu bar*, validando que las acciones necesarias para la realización de cada una de las actividades, ya hayan sido cumplidas por el usuario.

```

class frame : wxFrame {
public:
    frame() {
        menu_bar = new wxMenuBar(menus);
        panel = new panel();
        status_bar = new wxStatusBar();
        this->attach(menu_bar);
        this->attach(panel);
        this->attach(status_bar);
    }
private:
    wxMenu menus[] = {*file, *collections, *photomosaic, *about};
    wxMenuBar *menu_bar;
    wxPanel *panel;
    wxStatusBar *status_bar;
}

class app : wxApp {
public:
    main() {
        wxFrame *frame = new wxFrame();
        frame->show();
    }
}

```

Figura 4.1.: Definición de las clases *app* y *frame*

4.1.2. Componente *Panel*

La clase desarrollada que hereda de *wxPanel*, posee dos métodos fundamentales que permiten convertir las imágenes manejadas por el *front-end* (*wxWidgets*) al *back-end* (*OpenCV*) y viceversa, como se expuso en la sección 3.6. La Figura 4.2 ilustra esta clase.

```

class panel : wxPanel {
public:
    panel();
    void wxtocv(wxImage wx_image, cvImage &cv_image);
    void cvtowx(cvImage cv_image, wxImage &wx_image);
    void paint(wxImage wx_image);
    void clear();
}

```

Figura 4.2.: Definición de la clase *panel*

Los métodos *wxtocv* y *cvtowx* llevan a cabo un análisis de la imagen de entrada, determinando el número de canales y el tipo de dato de éstas, lo que permite definir las acciones a realizar. En el caso de las imágenes que siguen el espacio de color *RGB*, se evalúa el orden de los canales de las imágenes,

ya que *OpenCV* adicionalmente soporta el ordenamiento *BGR* [20]. Los métodos *paint* y *clear* permiten dibujar una imagen en el panel o eliminarla.

4.1.3. Componentes *Dialogs*

Los *dialogs* de la aplicación son creados como clases que heredan de *wxDialog*, los cuales permiten definir diversos elementos de interfaz que pueden ser incorporados tales como *labels*, *text fields* y *buttons*. La Figura 4.3 presenta la estructura básica de los diálogos utilizados en la aplicación.

```
class dialog : wxDialog {
public:
    dialog() {
        label = new wxLabel();
        text_input = new wxTextInput();
        button = new wxButton();
        this->attach(label);
        this->attach(text_input);
        this->attach(button);
    }
private:
    wxLabel *label;
    wxTextInput *text_input;
    wxButton *button;
}
```

Figura 4.3.: Definición de la clase *dialog*

La biblioteca permite manejar los posibles eventos asociados a los diálogos de forma sencilla. En la aplicación desarrollada, el evento más comúnmente manejado es la validación de los campos indicados por el usuario una vez que éste hace clic en el botón de aceptación, mostrando mensajes informativos según sea el caso.

4.2. Base de datos

El *DBMS* elegido para el desarrollo de la aplicación ha sido *MongoDB* por la facilidad y versatilidad que presenta al momento de manejar estructuras de datos de grandes dimensiones. Se ha elegido un sistema de base de datos *NoSQL* debido a sus prestaciones para aplicaciones de estas características, como se presentó en la Sección 1.4.

4.2.1. Semántica de la base de datos

La estructura básica de la base de datos creada para esta aplicación se presenta en la Figura 4.4. *MongoDB* en lugar de trabajar con tablas como las bases de datos *SQL*, define el concepto de colección, que es una estructura que permite el almacenamiento de información de diversa naturaleza.

La aplicación define una colección de control que almacena los nombres de las colecciones creadas por el usuario, así como también el directorio asociado a éstas. Una colección en este contexto representa un directorio de imágenes elegido por el usuario para la generación de fotomosaicos.

El resto de las colecciones corresponden a las creadas por el usuario, las cuales contienen cinco vectores, donde la posición i de cada uno de ellos representa un atributo de la imagen i . Estos atributos corresponden al nombre de las imágenes, sus vectores de color, textura y forma, así como también el número de veces que han sido seleccionadas como mejores candidatos en el fotomosaico actual.

```

database photomosaics {
    collection information {
        vector<string> collection_names;
        vector<string> collection_paths;
    }
    collection name_of_collection_0 {
        vector<string> image_names;
        vector<vector<float>> dcd;
        vector<vector<float>> ehd;
        vector<vector<float>> rbsd;
        vector<int> used;
    }
    ...
    collection name_of_collection_n-1 {
        vector<string> image_names;
        vector<vector<float>> dcd;
        vector<vector<float>> ehd;
        vector<vector<float>> rbsd;
        vector<int> used;
    }
}

```

Figura 4.4.: Semántica de la base de datos definida

4.2.2. Semántica de los vectores característicos

La semántica de los vectores característicos asociados a los descriptores visuales considerados viene dado por la Figura 4.5. Como se mencionó, el *DCD* utiliza el algoritmo k -medias para obtener los k colores dominantes de una imagen. Se ha utilizado $k = 8$ dado que esta cantidad de colores permite representar apropiadamente de forma general los colores presentes en una imagen. Adicionalmente, el espacio de color utilizado para representar las imágenes con este descriptor (*CIE-LUV*), requiere de la utilización de tres canales por cada color, almacenando también el número de ocurrencias de cada uno de ellos en la imagen cuantizada, normalizando esta ponderación en el intervalo $[0,1]$. En consecuencia, el vector característico de color posee $8 * (3 + 1) = 32$ elementos.

La Sección 2.4 presenta el *EHD*, que considera los histogramas de bordes locales, globales y semi-globales (*LEH*, *GEH* y *SGEH*), los cuales poseen 80, 5 y 65 contenedores respectivamente. Por

lo tanto, el vector característico de textura posee $80 + 5 + 65 = 150$ elementos. Asimismo, la Sección 2.5 presenta el *RBSD*, que considera la magnitud de los coeficientes de la *ART* de orden p y q , con $0 \leq p < 3$ y $0 \leq q < 12$, excluyendo la magnitud del coeficiente de orden $p = 0$ y $q = 0$, ya que es utilizada para normalizar el resto. De acuerdo a esto, el vector característico de forma posee $(3 * 12) - 1 = 35$ elementos.

```
// Vector característico de 32 elementos del DCD
vector<float> dcd = {l0, u0, v0, p0, ..., l7, u7, v7, p7};
// Vector característico de 150 elementos del EHD
vector<float> ehd = {leh0, ..., leh79, geh0, ..., geh4, sgeh0, ..., sgeh64};
// Vector característico de 35 elementos del RBSD (0 <= p < 3 && 0 <= q < 12)
vector<float> rbsd = {art1, ..., art35};
```

Figura 4.5.: Semántica de los vectores característicos de color, textura y forma

4.2.3. Proceso de búsqueda del mejor candidato

Se ha definido un conjunto de métodos que permiten el manejo de la base de datos llevando a cabo consultas, inserciones y eliminaciones. Sin embargo, el método de mayor trascendencia que permite obtener el nombre del mejor candidato asociado a una región de la imagen base, se presenta en la Figura 4.6.

Este método recibe como parámetro el nombre de la colección seleccionada por el usuario para ser utilizada durante la generación del fotomosaico. Además, recibe los vectores característicos de la región de la imagen base correspondiente, así como también la ponderación asociada a cada descriptor visual, establecidas previamente por el usuario.

El método lleva a cabo dos iteraciones, donde la primera de ellas calcula la distancia entre la región de la imagen base y cada una de las imágenes de la colección, en base a los criterios de color, textura y forma, almacenando cada una ellas en vectores para uso posterior. Además, se determina la menor y mayor distancia generada por cada uno de los descriptores visuales con el fin de poder llevar a cabo un proceso de normalización.

La segunda iteración normaliza en el intervalo $[0,1]$ las distancias obtenidas de la comparación de la región de la imagen base con cada una de las imágenes candidatas, utilizando los valores mínimo y máximo obtenidos por cada descriptor, con el fin de poder combinarlos. La combinación se lleva a cabo mediante el cálculo del promedio ponderado de las distancias obtenidas por los tres descriptores visuales, utilizando las ponderaciones recibidas por parámetro.

La imagen elegida es aquella que minimice el promedio ponderado, retornando su nombre para la posterior consulta de sus vectores característicos. Esta imagen es considerada como el mejor candidato a sustituir en el parche respectivo del fotomosaico, en base a la información obtenida de la región de la imagen base recibida por parámetro.

```

string best_candidate_name( string collection_name,
                           vector<float> dcd_of_region,
                           vector<float> ehd_of_region,
                           vector<float> rbsd_of_region
                           float dcd_weight,
                           float ehd_weight,
                           float rbsd_weight) {
int collection_size;
float min, current, min_dcd, max_dcd, min_ehd, max_ehd, min_rbsd, max_rbsd;
min = min_dcd = min_ehd = min_rbsd = max_float_value_supported;
max_dcd = max_ehd = max_rbsd = min_float_value_supported;
vector<float> dcd_of_candidate;
vector<float> ehd_of_candidate;
vector<float> rbsd_of_candidate;
collection_size = get_number_of_elements(collection_name);
vector<float> dcd_distances(collection_size);
vector<float> ehd_distances(collection_size);
vector<float> rbsd_distances(collection_size);
pointer *p; string name;
for(int a = 0; a < collection_size; ++a) {
    p = get_pointer_to_element(collection_name, a);
    dcd_of_candidate = p->get_attribute_of_element("dcd", a);
    ehd_of_candidate = p->get_attribute_of_element("ehd", a);
    rbsd_of_candidate = p->get_attribute_of_element("rbsd", a);
    dcd_distances[a] = compute_dcd_distance(dcd_of_region, dcd_of_candidate);
    if(dcd_distances[a] < min_dcd) min_dcd = dcd_distances[a];
    if(dcd_distances[a] > max_dcd) max_dcd = dcd_distances[a];
    ehd_distances[a] = compute_ehd_distance(ehd_of_region, ehd_of_candidate);
    if(ehd_distances[a] < min_ehd) min_ehd = ehd_distances[a];
    if(ehd_distances[a] > max_ehd) max_ehd = ehd_distances[a];
    rbsd_distances[a] = compute_rbsd_distance(rbsd_of_region, rbsd_of_candidate);
    if(rbsd_distances[a] < min_rbsd) min_rbsd = rbsd_distances[a];
    if(rbsd_distances[a] > max_rbsd) max_rbsd = rbsd_distances[a];
}
for(int a = 0; a < collection_size; ++a) {
    dcd_distance = normalize_between_0_and_1(dcd_distances[a], min_dcd, max_dcd);
    dcd_distance = dcd_weight * dcd_distance;
    ehd_distance = normalize_between_0_and_1(ehd_distances[a], min_ehd, max_ehd);
    ehd_distance = ehd_weight * ehd_distance;
    rbsd_distance = normalize_between_0_and_1(rbsd_distances[a], min_rbsd, max_rbsd);
    rbsd_distance = rbsd_weight * rbsd_distance;
    current = dcd_distance + ehd_distance + rbsd_distance;
    if(current < min) {
        min = current;
        name = p->get_attribute_of_element("name", a);
    }
}
return name;
}

```

Figura 4.6.: Obtención del nombre del mejor candidato asociado a una región de la imagen base

4.3. Descriptor de colores dominantes

4.3.1. Métodos principales

El *DCD* recibe como parámetros las estructuras necesarias para hallar su vector característico asociado, expuesto en la Figura 4.5. Los parámetros recibidos corresponden a la imagen de entrada (*image*), el vector de colores dominantes (*colors*) y su nivel de presencia en la imagen (*weights*). Además, se reciben los parámetros necesarios para la ejecución del algoritmo *k*-medias, tales como el número de colores dominantes, la cantidad máxima de iteraciones a realizar y el umbral de decisión como criterio de finalización.

El primer paso que debe realizarse es la conversión del tipo de dato de la imagen a flotante, así como su normalización al intervalo $[0,1]$, además de la llamada al algoritmo *k*-medias con los parámetros indicados, como se presenta en la Figura 4.7.

```
void dcd(matrix<int3> image,
        vector<float3> &colors,
        vector<float> &weights,
        int k,
        int max_iterations,
        float threshold) {
    image.convert_to("float3");
    image.normalize_between_0_and_1(0, 255);
    kmeans(image, colors, weights, k, max_iterations, threshold);
}
```

Figura 4.7.: Parámetros iniciales y llamada al algoritmo *k*-medias

El algoritmo *k*-medias actúa como un método iterativo, definiendo dos condiciones de parada, donde la primera de ellas viene representada por la convergencia exitosa del método y la segunda por el alcance del máximo número de iteraciones permitido. Antes de dar comienzo a la primera iteración del método, se inicializa el conjunto de colores dominantes de forma aleatoria siguiendo una distribución uniforme.

Dentro de la estructura iterativa *while*, una de las primeras tareas que se lleva a cabo es la generación de una matriz del tamaño de la imagen original, donde cada entrada de la misma hace referencia al color dominante más cercano al valor del píxel de esa posición en la imagen. Este proceso es realizado en la *GPU* dada la independencia existente entre los cálculos a realizar.

En base a esta matriz de valores, se obtiene el promedio de los píxeles de la imagen, asociado a cada uno de los colores dominantes mediante los vectores *accumulators* y *amounts*, siendo estos promedios los nuevos colores a utilizar en la siguiente iteración. Si todos los colores hallados son lo suficientemente cercanos a sus predecesores, el método iterativo finaliza al término de la iteración

actual. Existe la posibilidad de que alguno de los colores sea generado nuevamente de forma aleatoria si ningún píxel de la imagen ha sido asociado a éste, con el fin de tratar de encontrar un candidato en la siguiente iteración que pueda representar píxeles en la imagen. La Figura 4.8 presenta el algoritmo *k*-medias, adaptado al proceso de cuantización de imágenes.

```

void kmeans( matrix<float3> image, vector<float3> &colors, vector<float> &weights,
            int k, int max_iterations, float threshold) {
    int success, current_iteration;
    float3 color;
    matrix<int> color_positions(image.size);
    vector<float3> accumulators(colors.size);
    cudaMatrix<int> cuda_color_positions(image.size);
    cudaMatrix<float> cuda_image;
    cudaVector<float3> cuda_colors;
    cuda_image.upload(image);
    colors.fill_with_random_values();
    cuda_colors.upload(colors);
    current_iteration = 0;
    while(true) {
        if(success == k) break;
        if(current_iteration >= max_iterations) break;
        accumulators.initialize_with_zeros();
        weights.initialize_with_zeros();
        cuda_kernel(cuda_image, cuda_colors, cuda_color_positions, k);
        cuda_color_positions.download(color_positions);
        for(int a = 0; a < image.height; ++a) {
            for(int b = 0; b < image.width; ++b) {
                accumulators[color_positions[a][b]] += image[a][b];
                weights[color_positions[a][b]]++;
            }
        }
        success = 0;
        for(int a = 0; a < k; ++a) {
            if(weights[a] > 0) {
                color = accumulators[a] / weights[a];
                if(euclidean_distance(color, colors[a]) <= threshold) {
                    success++;
                }
                colors[a] = color;
            } else {
                colors[a].generate_random_value();
            }
        }
        current_iteration++;
    }
}

```

Figura 4.8.: Algoritmo *k*-medias para la cuantización de imágenes

4.3.2. *CUDA-kernel*

El *CUDA-kernel* que permite hallar la matriz de correspondencia entre los colores dominantes y la imagen, se presenta en la Figura 4.9. Se busca para cada píxel de la imagen, la distancia entre éste y cada uno de los colores dominantes, determinando la menor de ellas y estableciendo su ubicación en la matriz indicada. Este proceso tiene un costo computacional muy elevado cuando es implementado de forma secuencial, debido al tiempo de ejecución asociado al cálculo de las medidas de distancia.

```
void cuda_kernel(    cudaMatrix<float3> cuda_image,
                   cudaVector<float3> cuda_colors,
                   cudaMatrix<int3> &cuda_color_positions,
                   int k) {
    int min_distance_location;
    float distance, min_distance;
    cudaIndex i, j;
    min_distance = max_float_value_supported;
    i.initialize();
    j.initialize();
    for(int a = 0; a < k; ++a) {
        distance = euclidean_distance(cuda_image[i][j], cuda_colors[k]);
        if(distance < min_distance) {
            min_distance = distance;
            min_distance_location = a;
        }
    }
    cuda_color_positions[i][j] = min_distance_location;
}
```

Figura 4.9.: *CUDA-kernel* asociado al *DCD*

4.4. Descriptor de histograma de bordes

4.4.1. Método principal

Además del cálculo del *LEH*, la incorporación del *GEH* y el *SGEH* permite incrementar la precisión del descriptor al momento de clasificar imágenes. El algoritmo implementado recibe por parámetro únicamente la imagen (*image*) y una referencia al vector asociado a la combinación de los tres histogramas indicados (*eh*).

Dada la semántica de estos histogramas, se almacena en sus posiciones la frecuencia de los distintos tipos de borde (vertical, horizontal, diagonal 45°, diagonal 135° y no direccional), presentes en cada una de las sub-imágenes. Además de la consideración de estos histogramas, se definen histogramas adicionales que representan el número de bloques por sub-imagen que no han sido asociados a ninguno de los bordes presentados, denominados en esta sección como *LEH-NA*, *GEH-NA* y *SGEH-NA* según sea el caso.

Los cinco tipos de borde son almacenados en un vector y son copiados junto con la imagen en la memoria de la tarjeta gráfica, donde es calculado el *LEH* y el *LEH-NA*, definiendo una correspondencia de uno a uno entre los *CUDA-blocks* y las sub-imágenes, así como también entre los *CUDA-threads* y los bloques.

El primer ciclo *for* permite calcular el *GEH-NA* el cual considera la información de todas las sub-imágenes, mientras que el segundo permite calcular el *SGEH-NA* que considera las 13 configuraciones de sub-imágenes mostradas en la Figura 2.11. Por su parte, los dos últimos ciclos permiten determinar el *GEH* y el *SGEH* considerando las mismas sub-imágenes que los dos histogramas anteriores, llevando a cabo un proceso de normalización en base a éstos, resultando en el intervalo [0,1] cada valor de los contenedores.

Finalmente el *LEH* también es normalizado, generando un único histograma con la información de éste, del *GEH* y del *SGEH*. La Figura 4.10 presenta este descriptor, mostrando las estructuras involucradas y los cálculos realizados.

```
void ehd(matrix<int3> image, vector<float> &eh) {
    vector<float4> edges[5]; cudaVector<float4> cuda_edges[5];
    cudaVector<float> leh(80); cudaVector<float> leh_na(80);
    cudaVector<float> geh(5); cudaVector<float> geh_na(5);
    cudaVector<float> sgeh(65); cudaVector<float> sgeh_na(65);
    cudaMatrix<uint3> cuda_image;
    cudaMatrix<float> cuda_strengths(sub_images_w * block_w, sub_images_h * blocks_h);
    cuda_image.upload(image); cuda_image.convert_to_grayscale();
    cuda_image.download(image); edges.fill_with_the_five_edge_types();
    cuda_kernel(cuda_image, cuda_edges, leh, leh_na);
    for(int a = 0; a < sub_images_w * sub_images_h; ++a) {
        geh_na += leh_na[a];
    }
    for(int a = 0; a < 5; ++a) {
        for(int b = 0; b < sub_images_w * sub_images_h; ++b) {
            geh[a] += leh[(b * 5) + a];
        }
        geh[a] /= geh_na;
    }
    for(int a = 0; a < 13; ++a) {
        sgeh_na[a] = calculate_sgeh_na(leh_na, edges[a]);
        sgeh[a] = calculate_sgeh(leh, sgeh_na[a]);
    }
    for(int a = 0; a < 5; ++a) {
        for(int b = 0; b < sub_images_w * sub_images_h; ++b) {
            leh[(b * 5) + a] /= leh_na[b];
        }
    }
    eh = merge(merge(leh, geh), sgeh);
}
```

Figura 4.10.: Método para la generación del *EHD*

4.4.2. *CUDA-kernel*

La Figura 4.11 presenta el *CUDA-kernel* que permite obtener el *LEH*. Por cada bloque, se halla el promedio de las intensidades de los sub-bloques, calculando posteriormente las cinco posibles magnitudes asociadas a los cinco tipos de borde considerados, obteniendo la mayor de ellas, así como también su posición en el vector de tipos de borde recibido por parámetro e inicializado en el método que invoca al *kernel*. Si la mayor de las magnitudes es menor que el umbral establecido, no se asocia el bloque actual con un tipo de borde específico. En base a esta información, se genera el *LEH* y el *LEH-NA*.

```
void cuda_kernel(    cudaMatrix<uint3> cuda_image, cudaVector<float4> cuda_edges,
                   cudaVector<float> leh, cudaVector<float> leh_na) {
    cudaIndex i, j;
    i.initialize();
    j.initialize();
    vector<float> means[4];
    for(int a = 0; a < 4; ++a) {
        means[a] = sub_block_average(block[i][j]);
    }
    for(int a = 0; a < 5; ++a) {
        possible_strengths[a] = strength(block[i][j], cuda_edges[a]);
    }
    max_strength = max_value(possible_strengths);
    max_position = max_index(possible_strengths);
    if(max_strength > threshold) {
        strengths[i][j] = max_position;
    } else {
        strengths[i][j] = -1;
    }
    if(strengths[i][j] >= 0) {
        leh[strengths[i][j] + displacement]++;
    } else {
        leh_na[strengths[i][j] + displacement]++;
    }
}
```

Figura 4.11.: *CUDA-kernel* asociado al *EHD*

4.5. Descriptor de forma basado en regiones

4.5.1. Método principal

Este descriptor lleva a cabo gran cantidad de cálculos para obtener la magnitud de los coeficientes de la *ART*. Es por ello que se realiza un redimensionamiento de las imágenes de entrada a un valor de 100x100 píxeles.

La imagen es redimensionada en la *GPU* utilizando un método provisto por la biblioteca utilizada para el tratamiento de imágenes (*OpenCV*), basado en interpolación vía spline cúbico. Posteriormente, es convertida a escala de grises y binarizada, utilizando el umbral retornado por el método de *Otsu* [19], el cual recibe como parámetro el histograma de la imagen en escala de grises, calculado en la *GPU* y cargado en la memoria del computador.

Las sumas iteradas presentes en la Ecuación 2.22, vienen representadas a nivel de implementación por una matriz de acumuladores, la cual es generada en la *GPU* de acuerdo a la acumulación del producto compuesto por las funciones presentes en la ecuación indicada. La suma de los valores presentes en cada fila de la matriz, multiplicado por $\frac{1}{2\pi}$, representa los coeficientes de las componentes reales e imaginarias de la *ART*. El descriptor considera la magnitud de estos coeficientes, excluyendo la primera de ellas, la cual es utilizada como valor de normalización para el resto. La Figura 4.12 presenta el método principal asociado a este descriptor.

```
void rbsd(matrix<float> f, matrix<uint3> image) {
    int n, otsu_threshold;
    vector<int> histogram;
    vector<float> r(36);
    vector<float> i(36);
    matrix<float> accumulators;
    cudaVector<int> cuda_histogram;
    cudaMatrix<uint3> cuda_image;
    cuda_image.upload(image);
    if(image.width > 100 || image.height > 100) {
        cuda_image.cuda_resize(100);
    }
    cudaMatrix<float> cuda_accumulators(36, ((8 * image.size) + 4) * 2);
    cuda_image.convert_to_grayscale();
    cuda_histogram.calculate_histogram(cuda_image);
    cuda_histogram.download(histogram);
    otsu_threshold = otsu(histogram, image.size);
    cuda_thresholding(otsu_threshold, cuda_image);
    cuda_image.convert_to("float3");
    cuda_image.normalize_between_0_and_1(0, 255);
    cuda_accumulators.initialize_with_zeros();
    cuda_kernel(cuda_image, cuda_accumulators);
    cuda_accumulators.download(accumulators);
    r.initialize_with_zeros();
    i.initialize_with_zeros();
    for(int a = 0; a < 36; ++a) {
        r[a] = (1 / (2 * pi)) * sum(accumulators[begin:half]);
        i[a] = (1 / (2 * pi)) * sum(accumulators[half:end]);
    }
    f = sqrt((r[1:end] ^ 2) + (i[1:end] ^ 2));
    f = f / sqrt((r[0] ^ 2) + (i[0] ^ 2));
}
```

Figura 4.12.: Generación de los coeficientes de la *ART* y cálculo de sus magnitudes

4.5.2. *CUDA-kernel*

El *CUDA-kernel* mostrado en la Figura 4.13 asociado a este descriptor permite obtener los valores devueltos por las funciones definidas en las Ecuaciones 2.26, 2.27 y 2.28, en base a los valores de la imagen recibida como parámetro. El número de *CUDA-threads* utilizados pudiese modificarse de forma prudente, dando cobertura cada uno de ellos a una región mayor o menor sobre la cual operar.

```
float iq(int q, float vj, float vi);
float ip(int p, float uj, float ui);
float f(cudaMatrix<float> cuda_image, int n, float rho, float theta);

void cuda_kernel(cudaMatrix<float> cuda_image, cudaMatrix<float> cuda_accumulators) {
    cudaIndex i, j;
    i.initialize();
    j.initialize();
    accumulator = 0.0f;
    // Los valores "rho", "theta", "ui", "uj", "vi" y "vj" dependen de "i" y "j"
    foreach(i) {
        foreach(j) {
            alpha = f(cuda_image, n, rho, theta);
            beta = ip(p, uj, ui);
            gamma = iq(q, vj, vi);
            accumulator(i, j) += (alpha * beta * gamma);
        }
    }
}
```

Figura 4.13.: *CUDA-Kernel* para el cálculo de los acumuladores de las componentes reales e imaginarias de los coeficientes de la *ART*

4.6. Medidas de distancia

En esta sección se presentan los códigos asociados a las medidas de distancia utilizadas al momento de determinar el mejor candidato de una colección, con respecto a una región de una imagen base con el fin de generar un fotomosaico.

Asimismo, todas las medidas de distancia presentadas, retornan un valor real que posteriormente es normalizado en el intervalo [0,1] como se ha indicado anteriormente, con el fin de poder combinarlos.

4.6.1. *DCD*

La medida de distancia utilizada por este descriptor y propuesta por Sergyán [14], acumula las distancias euclídeas entre cada par de colores dominantes de las dos imágenes involucradas, multiplicado por el valor absoluto de la diferencia entre sus ponderaciones normalizadas en el intervalo [0,1], como se presenta en la Figura 4.14.

```

float distanceDCD( vector<color> ca, vector<float> pa,
                  vector<color> cb, vector<float> pb) {
    float c;
    float p;
    float distance;
    distance = 0.0f;
    for(int a = 0; a < ca.size; ++a) {
        for(int b = 0; b < cb.size; ++b) {
            c = euclidean_distance(ca[a] - cb[b]);
            p = absolute_value(pa[a] - pb[b]);
            distance += (c * p);
        }
    }
    return distance;
}

```

Figura 4.14.: Medida de distancia utilizada por el *DCD*

4.6.2. *EHD*

En el caso del *EHD* la medida de distancia utilizada viene dada por la acumulación de las diferencias en valor absoluto de los contenedores de los histogramas *LEH*, *GEH* y *SGEH* de las imágenes consideradas, ponderando además en un factor de 5 el *GEH* como se indica en [15], de forma que los bordes considerados desde un punto de vista global, tengan un mayor peso en la decisión. La Figura 4.15 presenta esta medida de distancia.

```

float distanceEHD(vector<float> ha, vector<float> hb) {
    float local, global, semiglobal, total;
    local = 0.0f;
    for(int a = 0; a < 80; ++a)
        local += fabsf(ha[a] - hb[a]);
    global = 0.0f;
    for(int a = 80; a < 85; ++a)
        global += fabsf(ha[a] - hb[a]);
    semiglobal = 0.0f;
    for(int a = 85; a < 150; ++a)
        semiglobal += fabsf(ha[a] - hb[a]);
    total = local + (5 * global) + semiglobal;
    return total;
}

```

Figura 4.15.: Medida de distancia utilizada por el *EHD*

4.6.3. *RBSD*

Esta medida de distancia lleva a cabo menor cantidad de operaciones aritméticas con respecto a las anteriormente presentadas, dada la cantidad de magnitudes que almacena el descriptor y la

aplicación de la función valor absoluto para la acumulación de las diferencias entre las distintas magnitudes de los coeficientes de orden p y q de la *ART* de las imágenes involucradas. Esta medida de distancia se presenta en la Ecuación 4.16.

```
COE_DIM_X 3
COE_DIM_Y 12
float distanceRBSD(vector<float> x, vector<float> y) {
    int a;
    float z;
    a = 0;
    z = 0.0f;
    while(a < (COE_DIM_X * COE_DIM_Y) - 1) {
        z = z + absolute_value(x[a] - y[a]);
        a++;
    }
    return z;
}
```

Figura 4.16.: Medida de distancia utilizada por el *RBSD*

Capítulo 5

Pruebas y Resultados

En este capítulo se presentan las pruebas realizadas sobre la aplicación desarrollada, las cuales abarcan el tratamiento de cada uno de los descriptores visuales con imágenes específicas, así como también su incorporación en este caso de estudio, la generación de fotomosaicos.

Del mismo modo, se presentan comparaciones de tiempos que permiten analizar distintos casos. El sistema sobre el cual ha sido probada la aplicación corresponde a una PC convencional sin características particulares con procesador *Intel^R CoreTM 2 Quad* con 4 GB de memoria RAM y una tarjeta gráfica con chip *NVIDIA GeForce GT 520*.

5.1. Colecciones

Dado que una de las etapas de mayor costo computacional del proceso de generación de fotomosaicos corresponde al procesamiento de las colecciones de imágenes, es importante la selección de una que no posea un número excesivo de imágenes. Evidentemente, la calidad visual de los fotomosaicos está estrechamente ligada a la cantidad y a la homogeneidad de las imágenes presentes en ella. En contraposición a esto, se ha utilizado una colección de imágenes lo más heterogénea posible con el fin de evaluar el rendimiento de los descriptores visuales implementados, en los cuales se centra este trabajo.

Se han recolectado imágenes de paisajes, automóviles, comida, ciudades y fantasía con características diversas que conforman una colección de 2340 imágenes, utilizadas para realizar las pruebas sobre los descriptores visuales y la generación de fotomosaicos. En la Figura 5.1 se aprecia un sub-conjunto de 230 imágenes de la colección considerada, las cuales poseen distintos formatos y dimensiones.

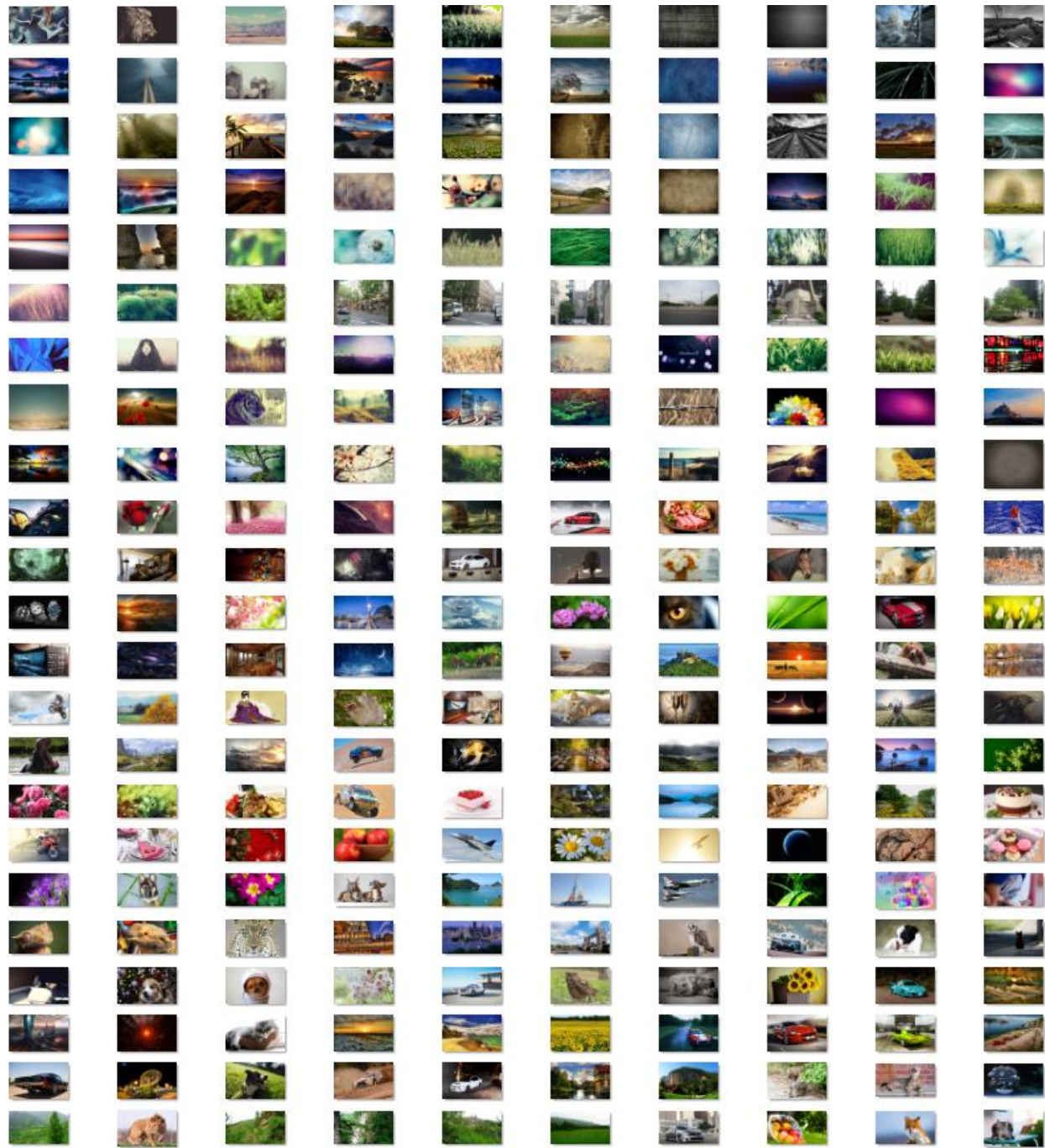


Figura 5.1.: Sub-conjunto de 230 imágenes presentes en la colección considerada

Cada imagen de la colección es sometida al cálculo de sus vectores característicos de color, textura y forma de acuerdo a los descriptores visuales considerados. El tiempo empleado para el procesamiento de la colección se presenta en la Figura 5.2. En ella, se aprecia que el tiempo de

ejecución ha sido de aproximadamente 82 minutos, siendo en promedio 2 segundos el tiempo empleado para cada imagen.

| Elemento | Tiempo (ms) | Tiempo (s) | Tiempo (m) |
|-----------|--------------|------------|------------|
| Colección | 4.900.010 ms | 4.900,01 s | 81,66 m |
| Imagen | 2.094 ms | 2,09 s | 0,03 m |

Figura 5.2.: Tiempo empleado para el procesamiento de la colección considerada

5.2. Descriptor de colores dominantes

Uno de los parámetros más importantes que requiere el algoritmo k -medias utilizado por este descriptor es el valor de k , dado que representa el número de colores dominantes de la imagen de entrada, utilizados para el cálculo de la medida de distancia. Se ha realizado una comparación entre cinco posibles valores que puede adoptar este parámetro, cuantizando una imagen de 2048×1536 píxeles de acuerdo a éstos. En la Figura 5.3(a) se presenta la imagen original, mientras que las imágenes (b), (c) y (d) representan sus cuantizaciones con $k = 4$, $k = 8$ y $k = 16$ respectivamente.

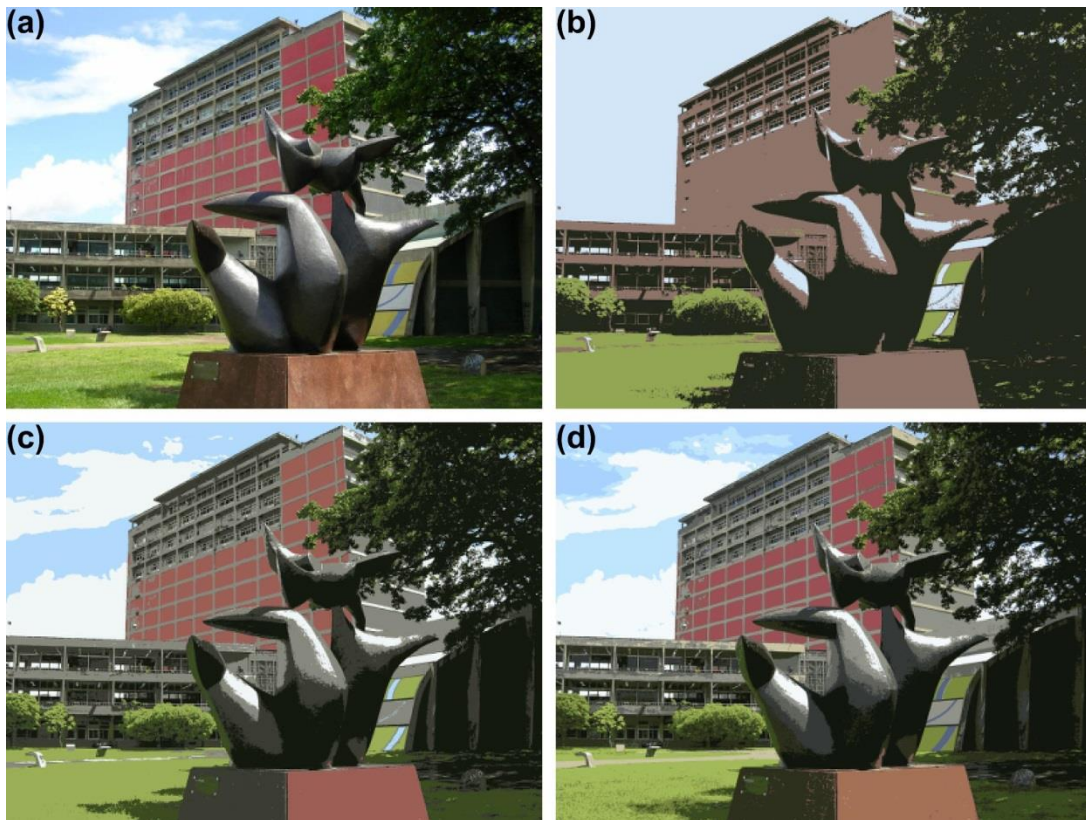


Figura 5.3.: Imagen cuantizada utilizando 4, 8 y 16 colores dominantes

El valor de k elegido en la aplicación desarrollada ha sido 8, dado que dicho número de colores es suficiente para representar la información de color relevante presente en una imagen, tal y como se muestra en la figura. En este ejemplo, se aprecia que con $k = 4$ no se tiene información suficiente para representar los colores presentes en las edificaciones y la obra ubicada en primer plano. Sin embargo, la información dada por una cuantización con $k = 8$ o superior es muy similar. El costo computacional del proceso de cuantización en cuanto a tiempo de ejecución para la imagen dada, se presenta en la Figura 5.4. Los tiempos presentados han sido promediados para 10 ejecuciones.

| k (número de colores dominantes) | Tiempo (ms) |
|------------------------------------|-------------|
| 2 colores | 274 ms |
| 4 colores | 504 ms |
| 8 colores | 1.021 ms |
| 16 colores | 1.580 ms |
| 32 colores | 2.694 ms |

Figura 5.4.: Tiempos de ejecución del proceso de cuantización de una imagen de 2048x1536 píxeles

5.3. Descriptor de histograma de bordes

Según [15], el cálculo del GEH y el $SGEH$ a partir del LEH , permite incrementar la precisión de este descriptor al considerar no solamente la información local de bordes sino también los patrones presentes en regiones más amplias. En la Figura 5.5 se presenta el rendimiento de este descriptor, considerando los histogramas GEH y $SGEH$, para una imagen de ejemplo.



Figura 5.5.: Rendimiento del EHD considerando la información global y semi-global de bordes

En la figura se muestran los 9 mejores candidatos pertenecientes a la colección considerada, con respecto a una imagen de entrada, de acuerdo a sus características de textura. Se aprecia que el candidato ganador es efectivamente el que presenta mayor similitud en cuanto a textura, ya que en ambas imágenes predominan múltiples hojas de helechos, las cuales poseen un patrón particular.

Dado que la colección posee imágenes de temáticas diversas entre las que destacan paisajes, automóviles, comida, ciudades y fantasía, se aprecia que el descriptor es lo suficientemente preciso para al menos, considerar como mejores candidatos, imágenes de temáticas similares a la imagen de ejemplo. Por su parte, dada la misma imagen de ejemplo, los resultados obtenidos prescindiendo de los histogramas *GEH* y *SGEH*, se presentan en la Figura 5.6.



Figura 5.6.: Rendimiento del *EHD* prescindiendo de la información global y semi-global de bordes

En este caso, prescindiendo de los histogramas *GEH* y *SGEH*, se aprecia que la precisión del descriptor no es tan elevada como en el caso previamente expuesto, estando el anterior ganador, esta vez en la octava posición. Sin embargo, la mayoría de los candidatos seleccionados como mejores, siguen siendo en general, de la misma temática de la imagen de entrada, a excepción del candidato que obtuvo el sexto lugar.

Dado que el color de las imágenes no es considerado por este descriptor, su criterio de decisión se basa únicamente en los niveles de las intensidades de éstas, de acuerdo al esquema de sub-imágenes y bloques presentado en la sección 2.4.1.

5.4. Descriptor de forma basado en regiones

El *RBSD* es junto con el *EHD*, el descriptor que más provecho saca del paralelismo provisto por la arquitectura, en este caso para generar los coeficientes de la *ART*. En esta sección, se presenta una comparación en cuanto a tiempos de ejecución entre este descriptor implementado en *CUDA C/C++* y una versión secuencial implementada en *MATLAB*, siguiendo las mejores prácticas. La Figura 5.7 presenta el código principal de este descriptor desarrollado en *MATLAB*, donde los métodos *Ip*, *Iq* y *f*, representan el cálculo de las Ecuaciones 2.26 a 2.28.

```
function f = art_coefficients(image_path)
image = imread(image_path);
level = graythresh(image);
binary = im2bw(image, level);
binary = double(binary);
m = size(binary, 1);
n = size(binary, 2);
coefficient_of_order_0_0 = [0 0];
f = zeros(35, 1);
space = 1 / (n / 2);
for p = 0 : 2
    for q = 0 : 11
        accum = [0 0];
        for i = 0 : ((n / 2) - 1)
            for j = 0 : (((8 * i) + 4) - 1)
                rho = space * (i + 1);
                theta = 2 * pi * ((j + 0.5) / ((8 * i) + 4));
                uj = ((2 * i) + 1) / n;
                ui = (2 * i) / n;
                vj = theta + (pi / ((8 * i) + 4));
                vi = theta - (pi / ((8 * i) + 4));
                beta = Ip(p, uj, ui);
                gamma = Iq(q, vj, vi);
                alpha = f(binary, n, rho, theta);
                accum(1, 1) = accum(1, 1) + (alpha * beta * gamma(1, 1));
                accum(1, 2) = accum(1, 2) + (alpha * beta * gamma(1, 2));
            end
        end
        if p == 0 && q == 0
            coefficient_of_order_0_0(1, 1) = (1 / (2 * pi)) * accum(1, 1);
            coefficient_of_order_0_0(1, 2) = (1 / (2 * pi)) * accum(1, 2);
        else
            r = (1 / (2 * pi)) * accum(1, 1);
            s = (1 / (2 * pi)) * accum(1, 2);
            f((p * 12) + q, 1) = sqrt((r^2)+(s^2));
        end
    end
end
f = f / sqrt((coefficient_of_order_0_0(1, 1)^2)+(coefficient_of_order_0_0(1, 2)^2));
```

Figura 5.7.: Método principal del *RBSD* implementado en *MATLAB*

El cálculo de la Ecuación 2.28 tiene un costo computacional sumamente elevado, dada la cantidad de operaciones aritméticas y trigonométricas que requiere realizar. Gracias a la independencia existente entre los cálculos que son necesarios llevar cabo para hallar los coeficientes de orden p y q de la *ART*, con $0 \leq p < 3$ y $0 \leq q < 12$, ha sido posible su paralelización en la GPU, permitiendo acelerar sustancialmente el tiempo requerido para el cálculo de los coeficientes de la transformada.

En ambos casos, es necesario que el proceso se lleve a cabo sobre imágenes pequeñas binarizadas (fijadas a 100x100 píxeles) dada la cantidad de cálculos a realizar. Como la forma fundamental de las imágenes varía ligeramente cuando se aplican redimensionamientos, es posible hacer este tratamiento inicial. Lo contrario sucede con los patrones de textura, los cuales se deforman en gran medida cuando las imágenes son redimensionadas, llegando a perderse.

La Figura 5.8 muestra los tiempos de ejecución presentados por este descriptor en sus versiones secuencial (*MATLAB*) y paralela (*CUDA C/C++*). En ella se aprecia que la mejora en tiempo de ejecución de la versión paralela con respecto a la secuencial para el caso considerado (100x100 píxeles) es aproximadamente 46x.

| Dimensión de la imagen (px) | MATLAB (ms) | CUDA (ms) | Mejora (CUDA con respecto a MATLAB) |
|------------------------------------|--------------------|------------------|--|
| 10x10 px | 259 ms | 12 ms | 22x |
| 20x20 px | 1.094 ms | 44 ms | 25x |
| 30x30 px | 2.524 ms | 72 ms | 35x |
| 40x40 px | 4.488 ms | 127 ms | 35x |
| 50x50 px | 6.992 ms | 186 ms | 38x |
| 60x60 px | 10.094 ms | 236 ms | 43x |
| 70x70 px | 13.765 ms | 325 ms | 42x |
| 80x80 px | 18.321 ms | 366 ms | 50x |
| 90x90 px | 22.641 ms | 486 ms | 47x |
| 100x100 px | 27.952 ms | 612 ms | 46x |

Figura 5.8.: Tiempos de ejecución del *RBSD* en sus versiones secuencial y paralela

La figura 5.9 presenta una comparación visual de los tiempos de ejecución empleados por ambas versiones del descriptor, los cuales difieren entre sí en mayor medida cuando las dimensiones consideradas aumentan.

En la figura, el crecimiento observado se aproxima a una función lineal. Los tiempos presentados por la versión paralela muestran las bondades que presenta la plataforma *CUDA* para el aprovechamiento del hardware gráfico disponible, con respecto a la versión secuencial considerada en este caso.

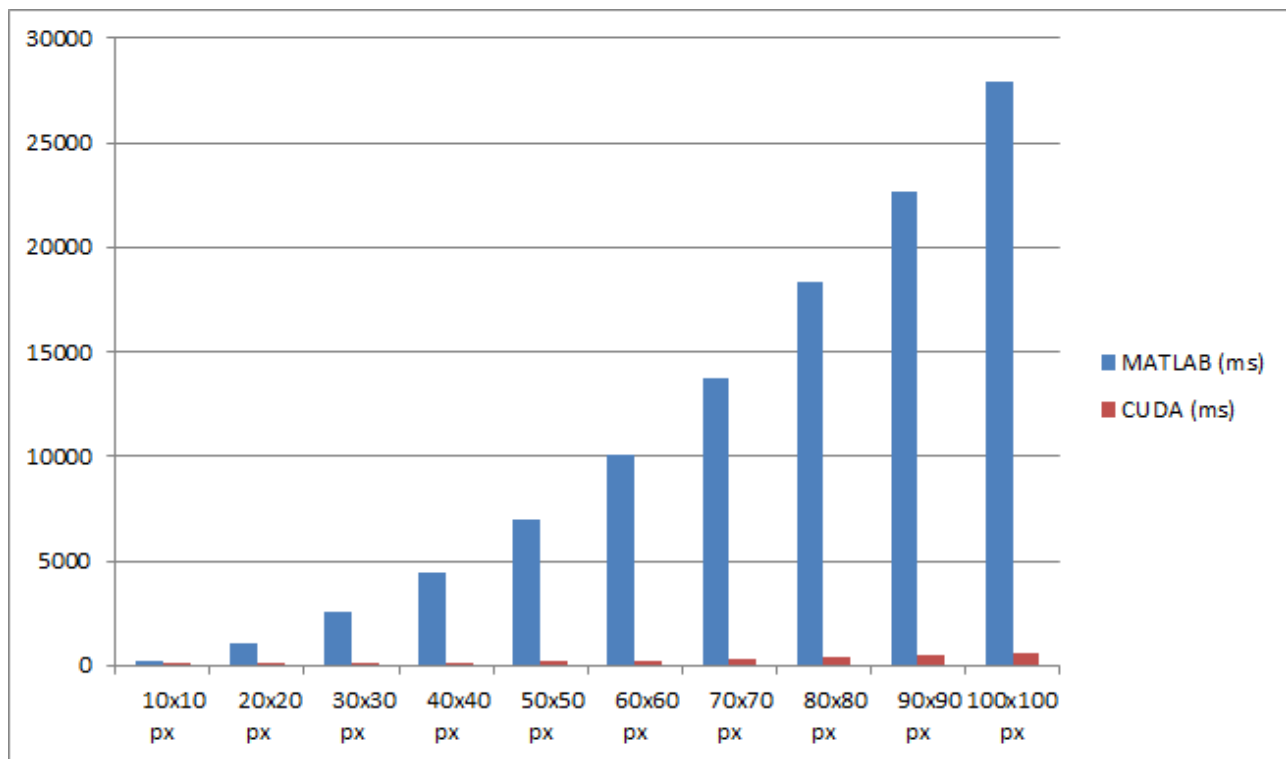


Figura 5.9.: Comparación visual de los tiempos de ejecución de las versiones consideradas del *RBSD*

5.5. Generación de fotomosaicos

Dado que se ha utilizado una colección de imágenes lo más heterogénea posible con el fin de evaluar el rendimiento de los descriptores visuales implementados, la generación de fotomosaicos no presenta alto nivel de calidad visual, como sí puede ocurrir haciendo uso de colecciones de imágenes homogéneas o de un mismo tipo (e.g. colección de imágenes de playas, bosques, rostros, logos u objetos particulares de una misma clase).

Se han realizado diversas pruebas modificando los parámetros de generación tales como la cantidad de parches presentes a lo ancho y a lo alto del fotomosaico, las dimensiones de éstos y de los parches, así como también la ponderación de los descriptores visuales, dado que se encuentran normalizados en el intervalo $[0,1]$. El número máximo de veces que un candidato puede ser seleccionado como el mejor, ha sido fijado a 100 para todos los casos, con el fin de evitar su continua aparición en la malla de parches.

La colección mostrada ha sido la utilizada para la realización de estas pruebas, sobre una imagen de 812×812 píxeles, que presenta distintos colores, texturas bien definidas y diversas formas en regiones específicas. La Figura 5.10 presenta esta imagen.



Figura 5.10.: Imagen base considerada para la generación de fotomosaicos

En la Figura 5.11 se presentan los fotomosaicos generados a partir de la imagen considerada. Todas ellas presentan gran cantidad de ruido, debido a la elevada diferencia visual entre cada parche y sus adyacentes. El uso de una colección homogénea de imágenes, permite la atenuación de este ruido, ya que se dispone de mayor cantidad de imágenes similares que pueden ser seleccionadas por los descriptores visuales para ser sustituidas en la malla de parches.

El control del número máximo de veces que un candidato puede ser seleccionado como el mejor, también agrava el ruido presente cuando se hace uso de una colección de imágenes heterogénea, ya que la cantidad reducida de buenos candidatos que pueden ser sustituidos en una región de la malla de parches, solo pueden ser seleccionados un número determinado de veces, presentándose la necesidad de seleccionar candidatos no óptimos para abarcar toda la región. Este fenómeno se aprecia en el fondo de la imagen base considerada, el cual es homogéneo, presentando los fotomosaicos un patrón totalmente distinto al esperado.

Los fotomosaicos de la primera columna de la Figura 5.11 han sido generados estableciendo la misma ponderación para cada uno de los descriptores visuales considerados. Estos fotomosaicos capturan la silueta básica de la imagen. Sin embargo, dada la heterogeneidad de las imágenes presentes en la colección (lo cual se ve reflejado en la diversidad de texturas y formas presentes), degradan el rendimiento del *EHD* y el *RBSD*.

Por su parte, los fotomosaicos de la segunda columna de la figura, han sido generados estableciendo una ponderación total para el *DCD*, ignorando la información dada por el *EHD* y el *RBSD*. Estos fotomosaicos presentan una mejora sustancial en su calidad visual, en especial el fotomosaico (d) con respecto al (c), presentándose el ruido de forma más homogénea en toda la imagen. Esto muestra que la sensibilidad del *DCD* es menor que la del resto de los descriptores visuales considerados, cuando se trabaja sobre colecciones de imágenes heterogéneas.

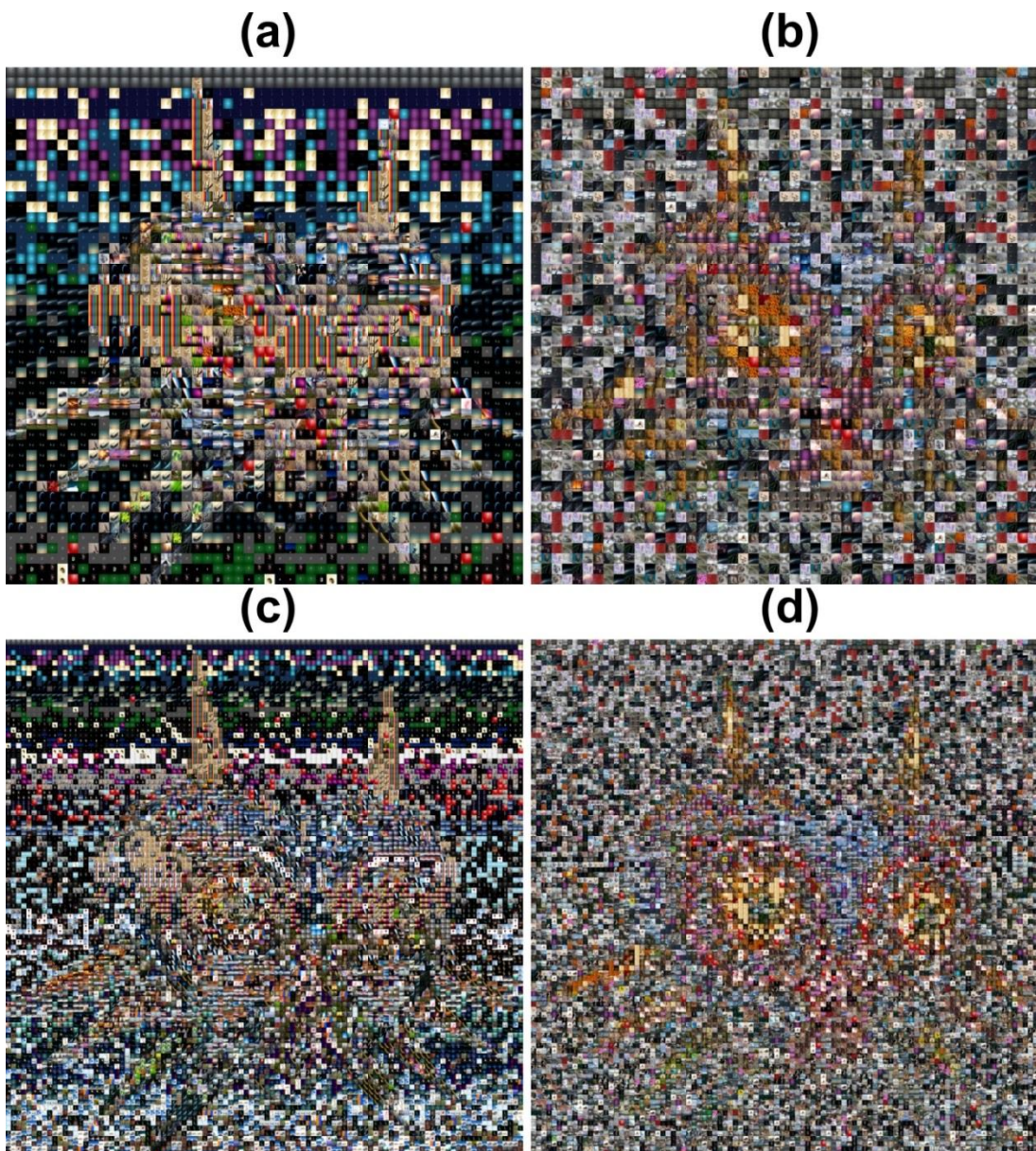


Figura 5.11.: Fotomosaicos generados variando sus parámetros

Los fотомосаicos de la primera fila han sido generados empleando 2500 parches de 10000 píxeles cada uno, mientras que los ubicados en la segunda fila emplean 10000 parches de 10000 píxeles cada uno. A partir de estas imágenes, se observa uno de los fenómenos más comunes presentes en los fотомосаicos que corresponde a su precisión. Mientras más cantidad de parches posee un fотомосаico, más precisa es su representación de la imagen base, si la comparación se lleva a cabo entre fотомосаicos que cumplen las mismas condiciones de generación, como en los casos presentados.

La Figura 5.12 presenta de forma compacta, toda la información relacionada a los fотомосаicos generados, presentando los parámetros utilizados y el tiempo empleado para la generación de cada uno de ellos.

| | Número de parches | Dimensión de los parches | Dimensión del fотомосаico | Ponderación (<i>dcd, ehd, rbsd</i>) | Tiempo (m) |
|------------|--------------------------|---------------------------------|----------------------------------|--|-------------------|
| (a) | 50x50 (2.500) | 100x100 px | 5.000x5.000 px (71,5 MB) | (0,34; 0,33; 0,33) | 16,83 m |
| (b) | 50x50 (2.500) | 100x100 px | 5.000x5.000 px (71,5 MB) | (1,00; 0,00; 0,00) | 17,35 m |
| (c) | 100x100 (10.000) | 100x100 px | 10.000x10.000 px (286,0 MB) | (0,34; 0,33; 0,33) | 63,98 m |
| (d) | 100x100 (10.000) | 100x100 px | 10.000x10.000 px (286,0 MB) | (1,00; 0,00; 0,00) | 65,21 m |

Figura 5.12.: Parámetros utilizados para la generación de los fотомосаicos y tiempos obtenidos

Capítulo 6

Conclusiones

En este trabajo se han implementado descriptores visuales definidos en el estándar *MPEG-7*, haciendo uso de la *GPU* para acelerar los procesos involucrados, los cuales permiten obtener con gran nivel de precisión, imágenes similares a otras. El caso de estudio considerado ha sido la generación de fotomosaicos como sistema de tipo *CBIR* (*Content-based Image Retrieval*) [21], que involucra la búsqueda de imágenes en base a sus características visuales en lugar de haciendo uso de metadatos (e.g. nombre de las imágenes), como ocurre en los motores de búsqueda convencionales.

Dado que la generación de fotomosaicos es un proceso computacionalmente costoso, su integración con descriptores de esta naturaleza, dificulta su generación en tiempo real. Por otra parte, la calidad visual de éstos puede verse incrementada de forma sustancial haciendo uso de estos descriptores, lo cual es dependiente de la selección de una colección de imágenes apropiada, que sea lo suficientemente representativa en cuanto a las necesidades de una aplicación de esta naturaleza.

Esta clase de aplicaciones es sumamente sensible a la colección de imágenes utilizada, por lo que es ideal que su contenido sea lo más homogéneo posible, tal que no se generen cambios visuales drásticos entre los parches de los fotomosaicos, como sucede con la colección considerada en este trabajo, dado su alto nivel de heterogeneidad. Se ha visto que los descriptores visuales pueden ser sensibles a las imágenes utilizadas, como es el caso del *EHD* y el *RBSD*, los cuales se ven degradados en mayor medida que el *DCD* cuando la colección considerada no es lo suficientemente homogénea.

Se propone como trabajos futuros la implementación de descriptores visuales adicionales definidos en el estándar *MPEG-7* que permitan obtener distintos atributos asociados a características de color, textura y forma, así como también la realización de pruebas de rendimiento en aplicaciones de distinta naturaleza, como por ejemplo, sistemas para el reconocimiento de patrones específicos en imágenes. Además, se propone su uso en aplicaciones para el análisis de secuencias de imágenes o videos, incorporando descriptores de movimiento definidos en el estándar que pueden ser útiles para el análisis de contenido en elementos multimedia.

Referencias

- [1] R. Silvers, "Photomosaics: Putting Pictures in their Place", June 1996.
- [2] Tessella Inc. (2014) *What are Tessellations?*. [En línea]. Disponible en: <http://tessella.com/what-are-tessellations/>
- [3] M. Slomp, M. Mikamo, B. Raytchev, T. Tamaki, K. Kaneda, "GPU-based SoftAssign for Maximizing Image Utilization in Photomosaics", in *International Journal of Networking and Computing*, July 2011, vol. 1, no. 2, pp. 211-229.
- [4] G. Di Blasi, G. Gallo, M. Petralia, "Smart Ideas for Photomosaic Rendering", in *Proceedings of Eurographics Italian Chapter*, February 2006, pp. 267-271.
- [5] J. Kim, F. Pellacini, "Jigsaw Image Mosaics", in *SIGGRAPH Proceedings of the 29th annual conference on computer graphics and interactive techniques*, 2002, pp. 657-664.
- [6] PixInsight Inc. (2011) *Interpolation Algorithms in PixInsight*. [En línea]. Disponible en: <http://pixinsight.com/doc/docs/InterpolationAlgorithms/InterpolationAlgorithms.html>
- [7] MIT-Adobe FiveK Dataset (June 2011). [En línea]. Disponible en: http://groups.csail.mit.edu/graphics/fivek_dataset/
- [8] Oracle Inc. (2014) *A Relational Database Overview*. [En línea]. Disponible en: <http://docs.oracle.com/javase/tutorial/jdbc/overview/database.html>
- [9] Planet Cassandra Inc. (2014) *What is a NoSQL database?*. [En línea]. Disponible en: <http://planetcassandra.org/what-is-nosql>
- [10] 10gen Inc. (2008) *What is NoSQL?*. [En línea]. Disponible en: <http://www.mongodb.com/nosql-explained>

- [11] MPEG (1988) *The Moving Picture Experts Group*. [En línea]. Disponible en: <http://mpeg.chiariglione.org/>
- [12] B. S. Manjunath, P. Salembier, T. Sikora, "Introduction to MPEG-7", *John Wiley & Sons, LTD*, March 2003.
- [13] J. Orallo, M. Ramírez, C. Ferri, "Introducción a la Minería de Datos", Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, *Pearson Prentice Hall*, 2008.
- [14] S. Sergyán, "Precision Improvement of Content-based Image Retrieval Using Dominant Color Histogram Descriptor", in *Proceeding of 1st WSEAS International Conference on Image Processing and Pattern Recognition at Budapest, Hungary*, December 2013, pp. 197.
- [15] D. Park, Y. Jeon, C. Won, "Efficient use of local edge histogram descriptor", in *MULTIMEDIA '00 Proceedings of the 2000 ACM workshops on Multimedia*, pp. 51-54.
- [16] K. Hosny, "Accurate Computation of ART Coefficients for Binary Images", in *Proceeding of 1st Taibah University International Conference on Computing and Information Technology*, 2012, vol. 1.
- [17] Y. Xin, M. Pawlak, S. Liao, "Accurate Computation of Zernike Moments in Polar Coordinates", in *IEEE Transactions on Image Processing*, February 2007, vol. 16, pp. 581-587.
- [18] J. Smart, K. Hock, S. Csomor, "Cross-Platform GUI Programming with wxWidgets", *Prentice Hall PTR*, 2006.
- [19] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms", in *Systems, Man and Cybernetics*, January 1979, vol. 9, pp. 62-66.
- [20] OpenCV Documentation (2014) *Load, Modify and Save an Image*. [En línea]. Disponible en: http://docs.opencv.org/trunk/doc/tutorials/introduction/load_save_image/load_save_image.html
- [21] A. Smeulders, M. Worring, S. Santini, A. Gupta, R. Jain, "Content-based Image Retrieval at the End of the Early Years", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, December 2000, vol. 22, pp. 1349-1380.