PROYECTO FIN DE CARRERA
DEPARTAMENTO DE INFORMÁTICA
E INGENIERÍA DE SISTEMAS

DISEÑO DE UN ENTORNO DE VISIÓN PARA EL TRATAMIENTO DE IMÁGENES

Zaragoza, 14 de octubre de 1997

MEMORIA

Área de Ingeniería de Sistemas y Automática

Director: Dr. Carlos Sagüés Blázquiz.

Autor: D. Joaquín Mur Amada.

El presente documento constituye la memoria del proyecto de fin de carrera de D. Joaquín Mur Amada. Este proyecto se comenzó a realizar en junio de 1996 y se ha finalizado en octubre de 1997.

El proyecto ha sido desarrollado en el Área de Ingeniería de Sistemas y Automática del Dpto. de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza, dentro del grupo de trabajo dedicado a la visión artificial.

AGRADECIMIENTOS

Quiero agradecer públicamente a todos los profesores/as que, desde las primeras etapas de mi vida hasta la Universidad, me han enseñado todo lo que sé. De ellos guardo un cariñoso recuerdo y admiro la paciencia que han tenido conmigo.

No podría haber llevado a cabo el proyecto sin la ayuda de Dr. Carlos Sagüés y de Dr. José Jesús Guerrero. Este proyecto tampoco se hubiera realizado sin D. José Enrique Nieto, del cual partió la idea inicial del proyecto. Sin Miguel Angel Martínez también hubiera sido imposible desarrollar el algoritmo multirresolución presentado en el capítulo 7.

Por último, no me olvido de todos aquellos sin los cuales no sería nada: mi familia, que siempre ha estado a mi lado y nunca me ha fallado, mis amigos/as con los cuales he pasado momentos inolvidables y Yolanda, que con sus ánimos ha hecho más llevadero el proyecto.

ÍNDICE DE LA MEMORIA

CAPÍTULO 1: INTRODUCCIÓN 1

1.1 Consideraciones iniciales 1

1.2 Objeto del proyecto 2

1.3 Alcance del proyecto 3

1.4 Organización de la memoria 7

CAPÍTULO 2: ESTUDIO DE LOS PROGRAMAS EXISTENTES PARA EL TRATAMIENTO DE IMÁGENES 11

2.1 Introducción 11

2.2 Programas y librerías de visión artifical 12

2.3 Utilidades relacionadas con la conversión de formatos 16

NetPBM 16

IM Raster Toolkit 17

Utah RLE Toolkit 17

Fuzzy Pixmap Manipulation 17

Img Software Set 17

Xim 17

xloadimage 17

TIFF Software 18

ALV 18

GIF tools 18

popi 18

ImageMagick 18

JPEG package 18

BIBLIOGRAFÍA: 19

CAPÍTULO 3: ESTUDIO DE FORMATOS PARA EL ALMACENAMIENTO DE IMÁGENES EN FICHEROS 21

3.1 Introducción 21

3.2 Tipos de ficheros para almacenar imágenes 21

3.2.1 El color en las imágenes 22

3.3 Formatos “raster” más comunes 25

3.4 Algoritmos de compresión más usuales 28

3.4.1 Compresión de longitud variable 28
3.4.2 Compresión Huffman 29
3.4.3 Compresión LZW 31
3.4.4 Compresión aritmética 32
3.4.5 Compresión con pérdidas 33

3.5 Estudio de los formatos "raster" más extendidos 35
3.5.1 Formatos recomendados para almacenar imágenes en tonos de gris o en color de alta densidad 35
3.5.2 Formatos recomendados para el almacenamiento de imágenes con 256 colores o menos 36
3.5.3 Formatos recomendados para el almacenamiento de imágenes bicolores o en blanco y negro 36
3.5.4 Formato IFF de Amiga 36
3.5.5 Formato GIF de CompuServe 37
3.5.6 Formato PCX de ZSoft 37
3.5.7 Formato BMP de Microsoft 38
3.5.8 Formato Scitex CT 38
3.5.9 Formato Targa 38
3.5.10 Formato TIFF 39
3.5.11 Formato JPEG 40
3.5.12 Formato PBM 41
3.5.13 Formato PGM 42
3.5.14 Formato PPM 44
3.5.15 Formato XWD 45
3.5.16 Formato XBM 46
3.5.17 Formato XPM 47
3.5.18 Formato RAW 50
3.5.19 Formato Vista 50
3.5.20 Formato PNG 51
3.5.21 Otros formatos. 51

3.6 Estudio de los formatos vectoriales y meta–ficheros más extendidos 52
3.6.1 Formato PostScript 53
3.6.2 Formato EPS 53
3.6.3 Microsoft Windows Metafile (WMF) 54
3.6.4 Computer Graphics Metafile (CGM) 56
3.6.5 Hewlett-Packard Graphics Language (HPGL) 56
3.6.6 Drawing Interchange Format (DXF) 57

BIBLIOGRAFÍA: 58

CAPÍTULO 4: GENERALIDADES DEL DISEÑO DEL ENTORNO DE VISIÓN 64

4.1 Introducción 64

4.2 Programación orientada a objetos. 65

4.3 Jerarquía de objetos utilizada en el entorno de visión en modo texto 65

4.4 Jerarquía de objetos utilizada en el entorno de visión en Microsoft Windows 69

BIBLIOGRAFÍA: 78

CAPÍTULO 5: LA CLASE IOFILE 79

5.1 Introducción 79

5.2 Funcionalidad de la clase IOfile 80

5.3 Estructura de la clase IOfile 81

5.4 Datos de la clase IOfile 83

5.5 Funciones públicas de la clase IOfile 91

5.5.1 Lectura y escritura de una imagen en un fichero. 91

5.5.2 Gestión de memoria 92

5.5.3 Acceso a la imagen 93

5.5.4 Funciones de la interfaz del usuario 96

BIBLIOGRAFÍA 99

CAPÍTULO 6: LA CLASE IMGBASIC 100

6.1 Introducción 100

6.2 Funcionalidad de la clase ImgBasic 101

6.3 Jerarquía de la clase ImgBasic 102

6.4 Estructura de la clase ImgBasic 103

6.5 Estructura general de la clase 104

BIBLIOGRAFÍA 108

CAPÍTULO 7: ALGORITMO CANNY MULTIRRESOLUCIÓN 109

7.1 Introducción 109

7.2 Definición del algoritmo Canny Multirresolución. 110

7.2.1 Filtrado iterativo de la imagen 112

7.2.2 Estimación del módulo y la dirección del gradiente 116

7.2.3 Tratamiento de las imágenes en color 120

7.2.4 Eliminación de puntos no máximos 121

7.2.5 Agrupamiento de puntos de contorno en segmentos 122

7.2.6 Cálculo de la correspondencia entre segmentos 124

7.2.7 Caracterización de los segmentos 127

7.3 Coste computacional 129

7.4 Resultados del algoritmo 132

7.5 Conclusiones 138

BIBLIOGRAFÍA 139

CAPÍTULO 8: CONCLUSIONES 140

Capítulo 1:

Introducción.

1.1 Consideraciones iniciales

El desarrollo de proyectos de investigación en visión artificial obliga a programar los algoritmos que se necesitan en cada aplicación. La experiencia muestra que la mayoría de estos algoritmos ya están realizados, pero no se pueden reutilizar en el nuevo proyecto por problemas de compatibilidad o falta de generalidad de las implementaciones. Esta situación provoca que se tenga que repetir el trabajo ya realizado en otros proyectos, con la consiguiente pérdida de tiempo y recursos humanos.

Como el investigador no tiene como objetivo prioritario utilizar técnicas de Ingeniería de Software que le faciliten la reutilización de código, desarrolla software no reutilizable. Para conseguir que se desarrolle software reutilizable hay que crear un entorno donde la adaptación suponga un esfuerzo menor que la reimplementación y, además, el entorno ha de tener el atractivo necesario para que el investigador decida utilizarlo.

Una primera solución es la creación de librerías de funciones de Procesamiento de Imágenes que implementan las técnicas más comunes en visión artificial. Éstas tienen el atractivo de ahorrar el trabajo de implementación de muchas técnicas que se utilizarán en un proyecto de investigación, pero no se adaptan al método de trabajo más utilizado de desarrollo en visión artificial:

1. Analizar las características de una imagen problema.
2. Crear un prototipo de procesamiento.
3. Si los resultados obtenidos con el prototipo no son correctos, hay que buscar las causas de los fallos de procesamiento y modificar convenientemente el prototipo.
4. Volver a probar.

Este proceso de prueba y error necesita un alto grado de interacción entre el investigador y la máquina que no consiguen las librerías. El proceso de compilación de

los programas y los detalles de implementación alargan el ciclo de desarrollo y reducen el atractivo de utilización de estas librerías. Además, se añade la problemática que muchas de ellas no son portables por estar ligadas a hardware específico y que no definen un método simple para la reutilización de nuevas funciones que desarrolle el investigador.

La solución a los problemas de las librerías, son los entornos de visión artificial. Estos se basan en una librería que se le ha añadido una interfaz de usuario que agiliza la interacción hombre–máquina y un método estandarizado para ampliar la librería. De esta forma las implementaciones realizadas por el investigador podrán añadirse al entorno para fomentar la reutilización y así, se asegura la actualización continua del entorno para que cubra las necesidades de la investigación actual y futura.

1.2 Objeto del proyecto

El objetivo del presente Proyecto Fin de Carrera es diseñar y realizar un entorno de visión 2D general, extensible y compatible. Este entorno será la base sobre la cual se puedan añadir algoritmos y funciones, con vistas a utilizar el entorno en prácticas de visión y en contactos con la industria.

El entorno funcionará sobre plataformas compatibles PC, debido a su uso generalizado, su reducido coste y su implantación en la industria. Otra razón adicional para utilizar ordenadores compatibles PC es la disponibilidad de equipos para hacer prácticas en el Centro Politécnico Superior de la Universidad de Zaragoza, y la mayor predisposición en la industria a implantar equipos de visión en plataformas PC que en caros y complejos sistemas UNIX.

El entorno soportará la utilización de imágenes de cualquier dimensión, en blanco y negro, 256 tonos de gris y en color, y se han de proporcionar las funciones adecuadas para poder leer y escribir los formatos de ficheros más populares. La organización de la imagen en memoria ha de ser simple e independiente del formato del fichero de la imagen.

El entorno mostrará imágenes en pantalla y proporcionar las funciones adecuadas para interactuar con el usuario y mostrar el resultado de los algoritmos, tanto de forma textual como gráfica.

La aplicación proporcionará una interfaz orientada a objetos, sencilla y potente, que abstraiga toda dependencia con aspectos de hardware y software, y que facilite la portabilidad a otro tipo de plataformas. También debe permitir un sencillo mantenimiento y la reutilización de las rutinas de alto nivel.

Dada la amplitud del proyecto, se ha fijado una prioridad de objetivos: definir una clase para el manejo de imágenes en memoria, codificar funciones para leer y escribir imágenes en ficheros en los formatos más utilizados, realizar una interfaz orientada a carácter, realizar una interfaz basada en ventanas y recopilar algoritmos de visión artificial.

1.3 Alcance del proyecto

En todo el proyecto se ha preferido dejar un entorno depurado, sencillo y robusto antes que incluir muchos componentes pero de baja fiabilidad que dificulten la reutilización del proyecto. Una prueba de ello es que actualmente hay dos proyectos fin de carrera cuyo objetivo es de ampliar el presente proyecto con algoritmos de inspección industrial.

Los pasos básicos que se han seguido durante la realización del proyecto han sido:

— **Elección del lenguaje de programación para implementar el entorno de visión.**

El lenguaje escogido para implementar el diseño del entorno de visión ha sido el lenguaje C, ya que aúna la velocidad del lenguaje *C* con la potencia de las clases y los objetos. Además, al ser el lenjuaje *C++* un superconjunto del *C*, la compatibilidad con el código escrito en `"ANSI C"` está asegurada. Como se verá en el capítulo segundo, otros entornos de visión artificial también han optado por utilizar el lenguaje de programación *C++*.

El entorno de visión ha sido realizado en la plataforma *MS-Windows*, yel compilador que se ha utilizado es *Microsoft Visual C++*. La interfaz del usuario se basa en la biblioteca de distribución gratuita `"Microsoft Fundation Classes"`, versiones 4.0 y 4.2, para el manejo del entorno *MS-Windows*. A lo largo de la memoria, me referiré a dicha biblioteca con las siglas `MFC`. La biblioteca `MFC` simplifica

notablemente las interacciones del programa con el sistema operativo, especialmente el manejo de la cola de eventos del sistema operativo, sin perder capacidad de control sobre la interfaz del usuario. La biblioteca `MFC` está disponible en las plataformas *PC, Macintosh, Power Macintosh, PowerPC* y *Alpha*.

— **Diseño del entorno de visión artificial**

Para facilitar la portabilidad, el mantenimiento y la expansión del entorno, se ha dividido en tres capas:

- Capa 1 o *núcleo del sistema*: en ella se soportan los tipos de datos básicos para el manejo de imágenes así como a las rutinas directamente relacionadas con la máquina en la que opera el sistema.
- Capa 2 o *librerías del sistema de visión*: en ella se encuentran las distintas rutinas de preprocesamiento, segmentación y que compondrán las librerías de tratamiento de imágenes del sistema. Las distintas rutinas que componen esta capa se han implementaedo, al igual que en la capa anterior, en código C++ estándar y permiten la portabilidad del sistema.
- Capa 3 o *interfaz del sistema*: en ella se elaborará el interfaz gráfico e interfaz hardware que proporciona la facilidad de manejo del sistema. Esta capa está íntimamente relacionada con el sistema operativo Windows y por lo tanto se utilizarán las herramientas de desarrollo incluidas en el compilador Microsoft Visual C++.

— **Definición de la estructura para el manejo de imágenes en memoria**

En esta primera fase se ha definido una única clase para el almacenamiento en memoria de las imágenes en color (utilizando el formato RGB o la paleta de colores), en tonos de gris y en blanco y negro.

Se ha implementado totalmente esta estructura, aunque los únicos formatos que se han mostrados útiles para el tratamiento de imágenes con color ha sido el formato RGB con 24 bits por pixel de profundidad. Para imágenes en tonos de gris, se ha utilizado la representación clásica de un byte por pixel, correspondiendo el valor 0 al negro y el valor 255 al blanco.

El formato de imagen con mapa de colores se ha mantenido ya que hay un gran número de formatos de ficheros de imagen, como GIF, TIF, BMP, PCX, etc., que emplean este formato para guardar las imágenes en disco. Además el formato en tonos de gris se puede considerar como una imagen con 256 colores, con un mapa predeterminado que contiene una gradación lineal desde el negro al blanco. Otra razón para soportar este tipo de imágenes es que este formato utilizado para visualizar imágenes en las pantallas con sólo 256 colores distintos.

Análogamente, se ha definido el formato bicolor para imágenes, utilizando el criterio "*min is black*", en otras palabras, se asigna el valor cero para el color negro y uno para el blanco. Hay bastantes formatos de ficheros que tratan de forma especial estas imágenes, y es conveniente unificar el criterio para tratar imágenes bicolores ya que existe el criterio "*min is black*", que es el que se ha utilizado, y el criterio "*min is white*", que asigna el valor cero para el color blanco y uno para el negro.

Como parte integrante de la clase, se ha diseñado las rutinas pertinentes para la gestión y ahorro de memoria, así como la capacidad de deshacer los cambios que se efectúen sobre la imagen.

— Implementación de funciones para leer y escribir imágenes en disco en los formatos más extendidos.

Al termino del proyecto, se han implementado los procedimientos para leer y escribir los formatos MS-Windows BMP, FITS, Compuserve GIF, RGB IRIS, JPEG, Utah Raster Toolkit, SUN Rasterfile, TARGA, TIFF, ZSoft PCX, PDS/VICAR, PM, PBM, XBM y XWD. A tal efecto, se han adaptado librerías de uso gratuito como *LIBTIFF* de Silicon Graphics y *JPEGLIB* del grupo independiente JPEG, y se ha buscado información sobre estos formatos en la bibliografía disponible e INTERNET. Como resumen de los formatos estudiados, se ha incluido en la memoria del proyecto un capítulo con las principales características.

— Realización de un interfaz de usuario en modo texto

En esta fase se han realizado una serie de menús, en C++ estándar y por tanto utilizando un entorno tipo terminal de texto, que permita cargar y grabar imágenes desde ficheros a memoria y efectuar algoritmos. Todas las operaciones de interacción

con el usuario se realizarán a través de funciones, de forma que los mensajes se puedan mostrar genéricamente, ya sea en un entorno orientado a carácter u otro sistema gráfico.

— **Realización de una interfaz gráfica para MS-Windows®**

En todo sistema de tratamiento de imágenes, es muy importante poder contemplar de forma rápida y sencilla el efecto que provoca un algoritmo sobre la imagen.

El interfaz gráfico programado tiene el aspecto de una aplicación Windows comercial y muestra las imágenes en pantalla con resoluciones de ocho bits de profundidad (modificando la paleta de colores del sistema, según sea la imagen de la ventana seleccionando) y con color de alta densidad (16 y 24 bits por pixel, sin necesidad de modificar la paleta de colores). Se ha incluido una opción para ahorrar memoria en ordenadores PC de gama baja y con la versión del sistema operativo MS-Windows 3.1. También se ha habilitado la utilización del portapapeles para cortar, copiar y pegar imágenes y la impresión de imágenes directamente desde el programa.

Inicialmente se pensó en implementar el entorno en un compilador de 16 bits, pero esta opción se ha desechado ya que en este caso las imágenes de más de 256 x 256 pixels se tendrían que representar en varios segmentos, además de contar con la limitación de 640 Kbytes de memoria convencional, aparte de los problemas de portabilidad que ello presenta. Por tanto se ha elegido como herramienta de trabajo un compilador de 32 bits, denominado Visual C++ versión 4.0. La versión 4.0 del compilador produce programas que pueden ejecutarse bajo MS-Windows 3.1 si están instaladas las librerías gratuitas Win32s. El entorno funciona de la misma manera si se compila con la versión 4.2 de Visual C++, pero en este caso el código generado es de 32 bits puro y sólo puede ejecutarse en los sistemas Windows 95 y Windows NT.

— **Recopilación de algoritmos de visión artificial**

Dada la amplitud de este proyecto, sería deseable poder incluir parte de programas o librerías existentes que faciliten el crecimiento y desarrollo futuro del entorno que se está realizando. En particular, se han estudiado las librerías "*Target*

Junior" e "*Image Algebra Classes — iac++*" y la aplicación "*Vista*". "*Target Junior*" ésta orientado a visión geométrica en 3-D, está programada en plataforma *X-Windows* con *Motif* y es una librería en constante evolución y cambio. La librería "*Image Algebra Classes*" trata aspectos más afines a un entorno 2-D, pero su funcionalidad es proporcionar plantillas genéricas con las cuales poder aplicar algoritmos de álgebra de imágenes. "*Vista*" es un entorno extensible y de disposición gratuita, basado en *Motif*, pero adolece de bastantes funciones. Por las razones que he comentado, he preferido referenciar la documentación obtenida sobre estas librerías pero no integrar estos paquetes informáticos en la aplicación final

Algunos algoritmos ya han sido incluidos en la versión actual, pero otros se han de adaptar al entorno, y preferiblemente debieran tratar la información contenida en imágenes en color, sin perjuicio de que traten de forma más eficiente las imágenes en tonos de gris.

— **Captura de imágenes directamente desde el entorno**

Inicialmente se pensó en captar directamente las imágenes con el entorno, pero la falta de controladores para ordenadores PC del sistema de visión que actualmente tiene el departamento hizo desestimar este objetivo.

1.4 Organización de la memoria

La memoria se distribuye en ocho capítulos. El primer capítulo sirve de introducción al proyecto y define el objeto y alcance del mismo.

Una vez que se ha definido el proyecto fin de carrera, se realiza un estudio de los programas existentes para el Tratamiento de Imágenes. Un estudio general de las herramientas disponibles, con sus virtudes y limitaciones, puede dar una idea de las características deseables de un entorno de Visión. Los programas estudiados van desde los programas de retoque fotográfico a los entornos de visión artificial, pasando por las librerías para el procesamiento de imágenes de aplicaciones de carácter general. En los

paquetes informáticos que se pueden obtener por Internet, se han incluido la dirección URL.

En el tercer capítulo se efectúa un estudio de los formatos gráficos. Dado que se han de habilitar la lectura y escritura de imágenes en varios formatos, sería ilógico no incluir un capítulo con el resumen de la información obtenida. Ya que los formatos gráficos son un campo en evolución continua, se han incluido, además de las referencias bibliográficas, algunas direcciones URL en donde se puede encontrar información de última hora o de las nuevas variaciones de los formatos.

El cuarto capítulo entra de lleno en el diseño del entorno de visión artificial. En este capítulo se intentará dar una visión general del entorno, de cómo están hechas las cosas y por qué se han hecho así. En la primera parte del capítulo se justificará la elección del lenguaje de programación con el cual se va a realizar el entorno. A continuación se presentará la estructura de las clases utilizadas en el caso más sencillo: cuando construimos el entorno en modo texto (dado que en modo texto sólo se utilizan las bibliotecas estándar de *C* y no existen las complejidades asociadas al entorno de ventanas, este caso es muy ilustrativo). La visión del entorno en modo texto será suficiente para que cualquier desarrollador de algoritmos tenga la información necesaria para realizar su cometido. Una vez que se ha estudiado y comprendido la disposición del entorno en modo texto, se procederá a comentar la organización del programa en una interfaz de ventanas. Dada la complejidad de las aplicaciones que utilizan ventanas, este apartado sólo será necesario para las personas que quieran ampliar las capacidades gráficas del entorno de visión. Aunque intentaré simplificar y esquematizar el entramado de clases que forman una aplicación gráfica, sería conveniente que quien leyera este apartado estuviera familiarizado con la arquitectura de programación "*documento–vista"* y con las clases `MFC`.

En el quinto capítulo se trata la clase base sobre la que se construye el entorno. Esta clase se ha denominado `IOfile` ya que contiene la estructura de la imagen en memoria y las rutinas necesarias para cargar una imagen desde un fichero a la memoria y para guardar una imagen contenida en memoria en un fichero. El nombre `IOfile`, algo esotérico, hace referencia a que esta clase gestiona las entradas y salidas del

programa –“Input-Output” en inglés–, y el almacenamiento de ficheros –“file” en inglés–. Este capítulo es imprescindible para cualquier persona que vaya a ampliar el entorno, ya que todo el entorno de visión gira en torno a esta clase.

Si bien el objeto del proyecto no es proporcionar una biblioteca con funciones de visión artificial, dentro de mis posibilidades intentaré proporcionar algunas funciones básicas de Visión Articial y Tratamiento de Imágenes que faciliten la labor de otros compañeros que implementen algoritmos más específicos. Las funciones las organizaré en la clase `ImgBasic`, que se deriva de la clase base `IOfile.` La organización de la clase `ImgBasic` se mostrará en el sexto capítulo. Además, esta clase puede servir como ejemplo y punto de partida para que los futuros usuarios puedan incluir en el entorno sus algoritmos, sin tener que aprender toda la estructura del entorno. El nombre es un apócope del término inglés “*IMaGe BASIC treatement*”, que indica que la clase `ImgBasic` proporciona rutinas para el tratamiento básico de la imagen.

Los capítulos quinto y sexto, junto con los manuales de referencia de la clase `IOfile` e `ImgBasic`, constituye la mayor parte de la documentación que han de consultar las personas que desarrollen algoritmos dentro del entorno.

En el séptimo capítulo se incluirá una breve descripción de un algoritmo multirresolución para la extracción de contornos. Este algoritmo, que inicialmente lo incluí en la clase ImgBasic, proviene del PFC de mi amigo D. Miguel Angel Martínez Esteban. Dado que ha habido cambios sustanciales en el algoritmo, he creído conveniente documentar dichas modificaciones. El resto de algoritmos no están comentados, ya que son algoritmos clásicos cuya descripción se puede encontrar en la bibliografía sobre visión artificial y tratamiento de imágenes.

En el último capítulo de la memoria expondré las conclusiones que he obtenido durante la realización del Proyecto Fin de Carrera.

Como anexos a la memoria se presentarán el manual de referencia de las clases `IOfile` e `ImgBasic`, en dónde aparecerán todas sus funciones y variables públicas. También se presentará como documentación los listados de los ficheros fuentes del

entorno, en forma de un segundo anexo. Dado que los listados ocupan más de 500 hojas en letra pequeña y su interés se reduce a aquellos programadores que desarrollen la funcionalidad de alguna rutina ya incluida, sólo se depositará una copia de dicho anexo en la biblioteca del Departamento de Informática e Ingeniería de Sistemas.

Aparte de la documentación, se pondrá a disposición pública el entorno de visión artificial. El paquete de "*software*" contendrá el programa compilado, listo para ejecutarse en ordenadores compatibles PC con el sistema operativo Windows '95, Windows NT 4.0 o Windows 3.1 con las librerías Win32. Además del programa ejecutable se proporcionarán todos los ficheros fuentes y "*makefiles*" necesarios para construir la aplicación. De esta forma, todas las personas interesadas en ampliar el entorno tendrán a su disposición los medios para compilar su propia aplicación. Las librerías JPEGLIB y LIBTIFF, necesarias para leer imágenes en formato JPEG y TIFF, también se incluyen junto a los ficheros fuentes del entorno, ya que son de pública disposición y no vienen incluidas con los compiladores de C++. En el segundo capítulo de la memoria se indicará también cómo se pueden obtener las librerías de las organizaciones que las han desarrollado.

Capítulo 2:

Estudio de los programas existentes para el tratamiento de imágenes

2.1 Introducción

En el primer capítulo habíamos fijado los objetivos y el alcance del proyecto. Antes de proceder al diseño del entorno de visión artificial, es necesario conocer las herramientas disponibles en la actualidad. Con sus virtudes y limitaciones se pueden determinar las características deseables del futuro entorno.

Actualmente existen aplicaciones informáticas útiles para la visión artificial, tan dispares como programas de retoque fotográfico, utilidades de conversión de formatos gráficos, aplicaciones matemáticas, aplicaciones de procesado de señales, etc.

El presente capítulo tiene dos objetivos:

— determinar las cualidades básicas de las herramientas disponibles y su posible utilidad para un futuro uso dentro del departamento.

— ayudar a detectar las necesidades que ha de satisfacer el futuro entorno de visión.

El capítulo está divido en dos bloques. En el primero se tratarán programas relacionados con la visión artificial en general y en el segundo bloque se estudiarán varias utilidades relacionadas específicamente con la conversión y manipulación de imágenes.

2.2 Programas y librerías de visión artifical

Las filosofías seguidas en los entornos de visión por ordenador son de lo más variadas y muy dependientes de los objetivos marcados. Algunos ejemplos son: Morpholog, Visilog y LaboImage, basados en menús que simplifican la interacción pero que no dan métodos flexibles para la creación de prototipos; HIPS, XITE basados en interfaces de comandos que permiten una interrelación suficiente y facilitan la construcción de prototipos, pero cuyo lenguaje base es demasiado simple; OBVIUS, GRILISP, PLIPS y Morphlisp basados en el lenguaje LISP que permite una interacción suficiente y supera la simplicidad de las interfaces de comandos, etc.

Jet Propulsion Lab de la NASA ha desarrollado el sistema **IMDISP** para funcionar en plataformas de tipo PC y destaca su sencillez de uso, ajustándose bien al área de la educación. Sin embargo, las rutinas de tratamiento de imágenes que incorpora son muy limitadas y no proporciona ningún sistema que permita añadir nuevas rutinas [IMD90].

GYPSY: "*a Growing but still Young Portable SYstem*". Sistema diseñado en Italia como parte del proyecto MODIAC, orientado a la automatización industrial. A pesar de no tratarse de un sistema adecuado para la enseñanza, su portabilidad, modularidad y facilidad de ampliación debe ser ejemplo para el presente trabajo [GIN85].

KHOROS: desarrollado por el Departamento de Ingeniería Eléctrica e Informática de la Universidad de Nuevo México desde Enero de 1987. Se trata del sistema de tratamiento de imágenes más completo de todos los analizados (incluye un lenguaje de programación visual, generador de código para extender el lenguaje visual y añadir de paquetes al sistema, un editor interactivo para el usuario, y una extensa librería de rutinas para el procesamiento de señales e imágenes). Su interfaz gráfico es muy completo y de fácil manejo, facilitando la interacción con el usuario novel. Sin embargo, el hecho de trabajar tanto con señales como con imágenes digitales incrementa el número de rutinas implementadas, dificultando que el alumno pueda

centrarse en las operaciones básicas de la visión artificial [KHO88]. Se puede obtener por ftp en `pprg.unm.edu:pub/khoros/`

IRAF: *Image Reduction and Analisys Facility*. Sistema orientado al trabajo con imágenes procedentes de equipos especializados en Astronomía que permite trabajar con otros tipos de datos. Su complejidad y dificultad de manejo la hacen inadecuada como herramienta educativa [IRA90].

VIEW–Station Environment: Sistema desarrollado para combinar el software existente en plataformas Unix junto con la potencia de computación de un procesador especializado en el tratamiento de imágenes. A pesar de tratarse de una buena herramienta para ser utilizada en un laboratorio de visión artificial, resulta inadecuada, por sus elevados requerimientos, dentro del ámbito educativo [SAT90].

MIRANDA: Entorno de programación funcional para le tratamiento de imágenes digitales que incluye una extensa librería de funciones y un editor para codificar las sentencias del lenguaje. El sistema está diseñado para trabajar en un entorno X-Windows. Su diseño se encuentra más encaminado hacia un lenguaje de programación de alto nivel que hacia una herramienta gráfica especializada en la enseñanza de las técnicas básicas de la visión artificial [POO90]**.**

Vista: es un entorno de visión que proporciona para casi todos los algoritmos una versión en línea de comandos y otra con un interfaz de usario basado en Motif. Es portátil, extensible y de disposición gratuita. Si bien es un entorno de uso adecuado, adolece de bastantes funciones y las posibilidades de programar el interfaz gráfico son limitadas. Proporciona un método simple para probar algoritmos de visión [VIS94]. Se puede obtener por ftp en `ftp.cs.ubc.ca/pub/local/vista`

iac++ es la librería de la clase *Image Algebra C++* de la universidad de Florida, encaminada a proporcionar un método general de representar tipos de imágenes a través de la capacidad de las plantillas de clases y a los operadores de C++. La librería es muy general y reutilizable y su diseño está orientado al tratamiento de imágenes, pero se centra en el álgebra y en clases abstractas. En un futuro próximo, cuando esté más desarrollada, puede ser una buena idea incluirla en el entorno de visión [IAC96]. Se puede obtener por ftp en `ftp.cis.ufl.edu/pub/ia/iac++`

TargetJr es un potente entorno de visión, compuesto por varios paquetes, que utiliza la capacidad de abstracción y encapsulamiento de *C++*. Se compone básicamente de la librería *COOL* de dominio público, de una serie de librerías de tratamiento de imágenes y visión artificial y de una librería de clases que sirve de interfaz entre *C++* y *X-Windows* con *Motif*. Este entorno es potente, y está encaminado a la visión tridimensional. Puede ejecutarse en plataformas SOLARIS, Silicon Graphics, LINUX, etc., pero no funciona bajo Microsoft Windows [TAR97]. Se puede obtener a partir de las direcciones `rabosa.inf.uji.es/software/targetjr.htm`, `www.balltown.cma.com` y en `robots.oxford.ac.uk`

La "*toolbox*" o herramienta de tratamiento de imágenes de **MATLAB** proporciona un entorno de programación muy potente, y una biblioteca de filtros y algoritmos para el tratamiento de imágenes apropiada, pero demanda unos consumos de recursos muy altos que hacen excesivamente lentas las operaciones sobre imágenes de tamaño real. Esta herramienta se ha utilizado en asignaturas de visión artificial del CPSUZ debido a sus aptitudes didácticas. El proceso de prueba y error es inmediato al utilizar un lenguaje interpretado y se dispone de un potente entorno de cálculo numérico y matricial. Sin embargo, la facilidad de utilización de vectores y matrices contrasta con la poca adecuación para estructuras complejas de datos –registros, árboles, etc.– [MAT95].

Algunas de las herramientas analizadas obligan a disponer de costosos equipos que reducen la posibilidad de establecer grupos de prácticas para un gran número de estudiantes. Otras son muy cerradas a nuevas ampliaciones lo cual minimiza drásticamente su utilidad en un campo tan creciente como es el de la visión artificial. Por último, otras son costosas y complicadas en su manejo, lo cual precisa la dedicación de unos recursos y un tiempo de los que no siempre se dispones. En cualquier caso, el análisis de las herramientas nos ha ayudado a determinar las características que deberían formar parte de una herramienta educativa.

También desde un punto de vista de investigación y desarrollo es de máximo interés disponer de una herramienta propia de prototipado y de un conjunto de librerías que ayude en la preparación de versiones definitivas de los productos. Esto se

ha tenido siempre presente en el diseño, separando el interfaz del núcleo del sistema, efectuando un desarrollo multiplataforma y poniendo especial énfasis en la comunicación de datos entre los diferentes algoritmos que constituyen el núcleo.

2.3 Utilidades relacionadas con la conversión de formatos

En el mercado existen bastantes programas para el tratamiento de imágenes sobre ordenadores personales, como "*Adobe Photoshop*" y "*Paint Shop Pro*", que poseen una velocidad de procesamiento muy aceptable y una interfaz de usuario muy conseguida. Pero estos programas se encaminan al retoque fotográfico y no poseen métodos sencillos para la expansión –es decir, poder añadir más filtros y rutinas de tratamiento sin tener que ser un experto del sistema operativo– ni sus códigos fuentes son accesibles. Además estos programas comerciales no son de libre distribución.

Por otra parte existen programas en plataforma UNIX como la aplicación "*xv*" que aúnan la potencia de los programas de tratamiento fotográfico y la capacidad de expansión, ya que el código fuente está disponible. En general, poseen bastantes operaciones de procesado de imágenes, pero a lo más que suelen llegar en temas de visión artificial es a la extracción de contornos y siempre se limitan a 2D.

Una buena parte de los programas UNIX de libre distribución que se comentarán a continuación se pueden obtener por ftp en ftp.cps.unizar.es/pub/graphics con una velocidad de acceso mucho mayor que en los nodos originales. Seguidamente se muestran una serie de herramientas que pueden ser útiles para la transformación de formatos de imagen.

NetPBM

Es una colección de herramientas que amplían el paquete PBMPLUS. Proporcionan programas de conversión de casi cualquier formato al formato PBM, PGM o PPM. Además contienen programas para la manipulación de imágenes. Se puede obtener en wuarchive.wustl.edu /graphics/graphics/packages/NetPBM.

IM Raster Toolkit

Es una herramienta portable y eficiente. El formato soporta pixels con diferente numero de canales, componentes y precisión, disponiendo de compresión e independencia del orden de byte de la máquina. Soporta manipulación de imagen y conversión de imagen. Se puede encontrar en awpaeth@watcgl.uwaterloo.ca vía ftp (Universidad de Waterloo).

Utah RLE Toolkit

Es un paquete de conversión y manipulación de imágenes, similar a PBMPLUS. Está disponible vía ftp en cs.utah.edu /pub/toolkit-2.0.tar.Z y ucsd.edu /graphics/utah-rater-toolkit.tar.Z

Fuzzy Pixmap Manipulation

Es un paquete de conversión y manipulación, similar a PBMPLUS. La versión 1.0 se puede obtener por ftp en nl.cs.cmu.edu /usr/mlm/ftp/fbm.tar.Z, uunet.uu.net /pub/fbm.tar.Z y ucsd.edu graphics/fbm.tar.Z. Autor: Michael Mauldin (mlm@nl.cs.cmu.edu).

Img Software Set

Lee y escribe en su propio formato de imagen, mostrando los resultados en una ventana X11, y realiza algunas manipulaciones de la imagen. La versión 1.3 está diponible en ftp.x.org:contrib/img_1.3.tar.Z, y venera.isi.edu:pub/img_1.3.tar.Z. Autor: Paul Raveling (raveling@venera.isi.edu).

Xim

Lee y escribe en su propio formato de imagen, mostrando los resultados en una ventana X11, y realiza algunas manipulaciones de la imagen. Necesita X11R4 y OSF/Motif. Lee y escribe GIF, xwd, xbm, tiff, rle, xim y otros formatos. Autor: Philip R. Thompson.

xloadimage

Lee imágenes y las muestra en una ventana X11. Disponible por ftp en ftp.x.org:contrib/xloadimage*. Author: Jim Frost (madd@std.com).

TIFF Software

Es un paquete de conversión, manipulación y tratamiento de imágenes. Se puede obtener por ftp en sgi.com:pub/graphics/*.tar.Z o uunet.uu.net:graphics/tiff.tar.Z. Autor: Sam Leffler (sam@sgi.com).

ALV

Un paquete de herramientas específico de Sun. La versión 2.0.6 se puede encontrar en comp.sources.sun. También se puede obtener por correo electrónico por alv-users-request@cs.bris.ac.uk.

GIF tools

Contiene una serie de utilizar para crear imágenes entrelazadas, transparentes, incluir varias imágenes en un archivo GIF, etc. Una copia de este paquete se puede encontrar en ftp.cps.unizar.es /pub/graphics/giftool*.tgz

popi

Es un lenguaje de manipulación de imágenes. La versión 2.1 se puede obtener en comp.sources.misc.

ImageMagick

Es un paquete de conversión, manipulación y tratamiento interactivo de imágenes. Utiliza su propio formato (MIFF) e incluye algunos convertores. Se puede obtener por ftp en ftp.x.org:contrib/ImageMagick.tar.Z.

JPEG package

Este paquete experimental incluye programas para comprimir ficheros gif y ppm al formato JPEG. Se puede obtener por ftp en uunet.uu.net:graphics/jpeg/jpegsrc.v1.tar.Z.

BIBLIOGRAFÍA:

[SIV96] A. Arroyo, F. Serradilla, José G. Zato. *SIVA: Sistema interactivo para la enseñanza de la visión artificial*. Universidad Politécnica de Madrid.

[GIN85] G. GINI y M. Gini. *A Software Laboratory for Visual Inspection and Recognition*, Informe Técnico, Pattern Recognition, Vol 18, nº 1, pp. 43-51, 1985.

[IMD90] *IMDISP Reference Manual*, Informe Técnico, Jet Propulsion Lab. & Goddard Space Center, NASA, 1990

[IRA90] *IRAF Reference Manual*, Informe Técnico, Central Computer Service. Nacional Optical Astronomic Observatories. 1990.

[KHO88] *Khoros Reference Manual*, Informe Técnico, Department of Electrical Engineering. University of New Mexico, 1988.

[POO90] I. Poole, *A Functional Environment for Imagen Analysis*, Informe técnico, Medical Research Council, Human Genetics Unit, pp. 124-127, IEEE, 1990

[SAT90] H. Sato, H. Okazaki, T. Kawai y H. Tamura, *The VIEW-Station Environment: Tools and Architecture for a Platform-Independent Image-Processing Workstation*, Information System Research Center, Canon Inc,. Pp. 576-583, IEEE 1990.

[VAL93] Enrique Julián Vallejos. *Interpretación y Desarrollo de un Entorno de Tratamiento y Análisis de Imágenes basado en LISP*. Facultad de Ciencias, Universidad Autónoma de Barcelona, 1993.

[DOB91] M. R. Dobie and P.H. Lewis, "Data Structures for Image Processing in C", Pattern Recognition Letters 12 (1991) 457-466.

[GEE91] S. J. Gee, "FLIPS: Friendly Lisp Image Processing System", SPIE Vol. 1472 Image Undestanding an the Man-Machine Interface III (1991).

[MUR91] J. J. Murillo, J. N. Wilson "An ADA Interpretative Sistem for Image Algebra", SPIE Vol. 1769 Image Algebra and Morphological Image Processing III (1992).

[PRA92] W. K. Pratt, "An Overview of the ISO/IEC Programmer's Imaging Kernel System: Application Program Interface" SPIE Vol. 1659 Image Processing and Interchange (1992).

[SAN93] J. Sánchez, J. Vitrià, "ViLi+: Extended LISP for Image Processing and Computer Vision" 7ICIAP 20-22 de septiembre de 1993.

[TAK92] T. Matxuyama, "A Survey of Image Processing and Computer Vision Software Systems on Workstations" Department of Information Technology's Report, Faculty of Engineering, Okayama University, 1992.

[VAL93] E. Vallejos "Definición y Desarrollo de un Entorno de Tratamiento y Análisis de Imágenes Basado en LISP" Trabajo Experimental. Septiembre 1993.

[MAT89] MATLAB for Macintosh Computers. User's Guide. The MATH WORKS Inc.

[IAC96] Software User Manual for the iac++ Class Library. Versión 2.2. January 12, 1995. Center of Comtuter Vision and Visualization, University of Florida.

[TAR97] Targer Jr Developper's Guide. D412, Penyeta Roja, UJI.

Capítulo 3:

Estudio de formatos para el almacenamiento de imágenes en ficheros

3.1 Introducción

En el capítulo anterior se han estudiado los programas existentes para el tratamiento de imágenes. Los programas que manipulan imágenes deben ser capaces de leer y escribir en ficheros las imágenes contenidas en memoria. Sin embargo, en la actualidad son tantos los formatos gráficos que se utilizan para almacenar imágenes que la elección de uno u otro es algo no trivial. Además, el entorno de visión ha de ser capaz de leer y escribir imágenes en varios formatos si quiere tener un uso general

.

En el presente capítulo se efectuará un estudio de los formatos gráficos. Parte de la información que aparece en este capítulo se ha obtenido durante la implementación de rutinas para leer los diversos formatos y son de procedencia muy diversa. Ya que los formatos gráficos son un campo en evolución continua, se han incluido, además de las referencias bibliográficas, algunas direcciones URL en donde se puede encontrar información de última hora o de las nuevas variaciones de los formatos.

3.2 Tipos de ficheros para almacenar imágenes

Un formato de tipo "*raster*", mapa de puntos, mapa de bits o "bitmap", divide la imagen en una rejilla igualmente espaciada, cuyos elementos se denominan "*pixels*" y almacena las propiedades cada pixel. El número de colores que el archivo puede contener está determinado por el número de bits que se almacenan por cada pixel.

Los meta–ficheros y los ficheros de imágenes vectoriales contienen información geométrica de figuras, segmentos, objetos, texto, etc. para representar una imagen. La imagen se guarda como una colección de fórmulas matemáticas. Las figuras vectoriales no pueden reproducir imagen foto–realistas, pero cuentan con dos ventajas sobre los formatos "*raster*": se pueden escalar sin distorsión y producen ficheros más pequeños.

3.2.1 El color en las imágenes

En la literatura se denominan imágenes monocromas[1] ("*monochrome images*") a las que únicamente contienen dos colores: el blanco y el negro. Cuando la imagen contiene el blanco, negro y/o tonos grises se denomina imagen de tonos o niveles de gris ("*grayscale images*"). Estas imágenes tienen un bit de profundidad ya que un pixel sólo puede tener dos estados: blanco y negro.

Las imágenes en color se pueden dividir en las que contienen sólo un número reducido de colores tomado de una tabla, mapa o paleta de colores ("*limited spot color*") y las que poseen una gama muy amplia de colores ("*color mixing*", aunque según la profundidad del color y del tipo de representación utilizada se suele utilizar la denominación HI-COLOR, TRUE-COLOR, etc.). En estos casos se combinan varios colores primarios para generar la tonalidad deseada, como en RGB (los colores rojo, verde y azul se suman. Se utiliza sobretodo en dispositivos que emiten luz, como monitores) y en CMYK (los colores cian, magenta y amarillo se restan, más el negro para mejorar el contraste, que típicamente se usa en las impresoras en donde cada tinta absorbe la luz de longitud de onda asociada al color).

Otro método de representación de color es HSI o HSL, en donde se especifica el tono, la saturación y la intensidad o luminosidad del color. El tono es el color que queda si eliminamos los tonos grises y aumentamos al máximo el ajuste del color del monitor; la

[1] Debido a la gran variedad de las fuentes de documentación y a la utilización de imágenes en campos tan diversos, los términos y acepciones que se emplearán de aquí en adelante pueden variar de unas fuentes a otras. No obstante, he intentado utilizar en la documentación el criterio seguido por la mayoría.

saturación describe la cantidad de color blanco de la imagen y la intensidad o luminosidad indica el brillo del color.

En aplicaciones de vídeo, se utiliza la representación YCbCr, en donde se especifica la luminosidad, la crominancia para el color azul primario y la crominancia para el color rojo primario. La separación de la luminancia de la crominancia permite técnicas de compresión que aprovechan la menor precisión del ojo frente al color que al brillo.

Las componentes de color se pueden almacenar contiguas, pixel a pixel (por ejemplo, para el caso RGB, se entrelazan los colores rojo, verde y azul de cada pixel: $R_1G_1B_1$ $R_2G_2B_2$ $R_3G_3B_3$ $R_4G_4B_4$...) o por planos, en donde el color se almacena como tres o cuatro imágenes monocromas, una para cada color primario, análogamente a la separación de color de la industria de impresión gráfica. Los planos de color o "*color planes*" también se denominan canales de color o "*color channels*". Para el caso RGB sería:

R_{11} R_{12} R_{13} R_{14}	G_{11} G_{12} G_{13} G_{14}	B_{11} B_{12} B_{13} B_{14}
R_{21} R_{22} R_{23} R_{24}	G_{21} G_{22} G_{23} G_{24}	B_{21} B_{22} B_{23} B_{24}
R_{31} R_{32} R_{33} R_{34}	G_{31} G_{32} G_{33} G_{34}	B_{31} B_{32} B_{33} B_{34}
R_{41} R_{42} R_{43} R_{44}	G_{41} G_{42} G_{43} G_{44}	B_{41} B_{42} B_{43} B_{44}

Los bits por pixel que tiene una imagen guardada en un fichero o en memoria es la cantidad de bits necesaria para contener un pixel o su equivalente si el formato comprime la imagen. Si la imagen está guardada por planos, el número de bits por pixels se determina multiplicando los bits por plano por el número de planos. Si el resultado no es una potencia de dos, se redondea a la potencia de dos más próxima por lo alto. Por ejemplo, si hay dos bits por plano y tres planos, el formato tiene ocho bits por pixel: $2 \times 3 = 6$ y $2^2<6<2^3$.

Con el término de profundidad de color nos referimos a la cantidad de colores que el ojo humano puede ver. En principio puede coincidir con los bits por pixel, pero para la impresión y en algunos monitores se produce una dispersión de puntos de distintos colores que producen el efecto óptico del color promedio de esa zona.

La mayoría de los formatos de imagen raster soportan mas de un nivel de bits por pixel, y por tanto, más de un nivel de color. La siguiente tabla contiene el número de bits por pixel que se utilizan en los formatos con el número máximo de colores que pueden soportar (en algunos formatos éste número puede cambiar debido a los algoritmos de compresión que utilizan).

Bits por pixel	Máximo número de colores
1	2
4	16
8	256
16	32.768 o 65.536, dependiendo del formato
24	16.777.216
32	16.777.216 o 4294.967.296, según formato

3.3 Formatos "*raster*" más comunes

Debido a que los formatos de imágenes están contínuamente evolucionando, es muy difícil hacer una lista exhaustiva de todos los formatos y sus características, ya que continuamente aparecen nuevos formatos o los que hay se amplían. Por una parte, hay formatos abiertos como el *TIFF* o *JPEG* que permiten que fabricantes de hardware o software definan sus propios algoritmos de almacenamiento. Por una parte, esto proporciona una interesante posibilidad de expansión de los formatos, pero esto es a costa de que algunos archivos *TIFF* o *JPEG* sólo puedan ser leídos por los programas más actuales.

Por otra parte, hay formatos gráficos que están patentados y es necesario pagar *royaltys* para poder utilizarlos. Un ejemplo de esto lo podemos encontrar en el formato GIF. En un principio el algoritmo de compresión que utilizaba el formato GIF –algoritmo LZW– era *freeware*, es decir, de dominio público, pero los derechos del mismo han sido comprados a su creador y los programas que lo utilicen deberán pagar, previsiblemente, un canon. También se da el caso de que compañías de software que reciben dinero por soportar diversos tipos de imágenes y de esta forma potenciar el uso de formatos que normalmente se han creado con un propósito específico.

En general, existe un compromiso entre la reducción del tamaño del fichero y complejidad para codificar y descodificar la imagen; y una mayor flexibilidad del formato supone que algunos programas posiblemente no serán capaces de utilizar todas sus características.

En la siguiente tabla se muestran los formatos más comunes con el número de bits por pixel que soportan. La presencia de un asterisco en una columna indica que el formato soporta esa profundidad de color. La columna "*8 Color*" se refiere a imágenes en color con un máximo de 256 colores, normalmente referenciados a una paleta de colores. La columna "*8 Grey*" indica que el formato admite una representación en tonos de gris de 256 tonos, correspondiendo los valores 0 y 255 a los colores negro o blanco.

Formato	Sub–Formato / Descripción	Fuente/Estándar	Bits por Pixel / Tipo de color						
			1	4	8 Grises	8 Color	16	24	32
BMP	codificación RGB	OS/2	*	*		*		*	
BMP	codificación RGB	Microsoft Windows	*	*		*		*	
BMP	codificación RGB + RLE	Microsoft Windows		*		*			
CLP	Bitmap	Windows Clipboard	*	*		*		*	
CLP	Device Independent Bitmap	Windows Clipboard	*	*		*		*	
CUT		Dr. Halo				*			
DIB	codificación RGB	OS/2	*	*		*		*	
DIB	codificación RGB	Microsoft Windows	*	*		*		*	
DIB	codificación RGB + RLE	Microsoft Windows		*		*			
EPS	Sólo imagen	Adobe	*	*		*		*	
GIF	Ver. 87a (entrelazado)	CompuServe	*	*		*			
GIF	Ver. 87a (no entrelazado)	CompuServe	*	*		*			
GIF	Ver. 89a (entrelazado)	CompuServe	*	*		*			
GIF	Ver. 89a (no entrelazado)	CompuServe	*	*		*			
IFF	Con compresión	Electronic Arts	*	*		*			
IFF	Sin compresión	Electronic Arts	*	*		*			
IMG	Versión antigua	GEM Paint	*	*		*			
IMG	Versión nueva	GEM Paint	*	*		*			
JIF	compresión Huffman	J.P.E.G.			*		*		
JPG	compresión Huffman	J.P.E.G.			*		*		
LBM	Con compresión	Deluxe Paint	*	*		*			
LBM	Sin compresión	Deluxe Paint	*	*		*			
MAC	With header	MacPaint	*						
MAC	Without header	MacPaint	*						
MSP	Versión nueva	Microsoft Paint	*						
MSP	Versión antigua	Microsoft Paint	*						
PBM	Portable Bitmap ascii	UNIX	*						
PGM	Portable Greymap ascii	UNIX			*				
PPM	Portable Pixelmap ascii	UNIX					*		
PBM	Portable Bitmap raw	UNIX	*						
PGM	Portable Greymap raw	UNIX			*				
PPM	Portable Pixelmap raw	UNIX					*		
PCD		Kodak Photo CD					*		
PCX	Version 0	ZSoft Paintbrush	*						
PCX	Ver. 2 (con paleta colores)	ZSoft Paintbrush	*	*					
PCX	Ver. 3 (sin paleta colores)	ZSoft Paintbrush	*	*					
PCX	Version 5	ZSoft Paintbrush	*	*		*		*	

Formato	Sub–Formato / Descripción	Fuente/Estándar	Bits por Pixel / Tipo de color						
			1	4	8 Grey	8 Color	16	24	32
PIC		Pictor/PC Paint	*	*		*			
PNG	Portable Network Graphics		*	*		*	*	*	*
PSD	RGB or indexed	Photoshop	*			*	*		
RAS	Type 1 (Estilo moderno)	Sun Microsystems	*			*		*	*
RAW	Valor del pixel sin codificar				*				
RLE	CompuServe	CompuServe	*						
RLE	Windows	Microsoft Windows		*		*			
TGA	Sin compresión	Truevision				*	*	*	*
TGA	Con compresión	Truevision				*	*	*	*
TIFF	Compresión Huffman	Aldus Corporation	*						
TIFF	Sin compresión	Aldus Corporation	*	*		*		*	
TIFF	Compresión Pack bits	Aldus Corporation	*	*		*		*	
TIFF	Compresión LZW	Aldus Corporation	*	*		*		*	
TIFF	Compresión Fax Group 3	Aldus Corporation	*						
TIFF	Compresión Fax Group 4	Aldus Corporation	*						
V	Vista		*			*	*	*	*
WPG	Version 5.0	WordPerfect	*	*		*			
WPG	Version 5.1	WordPerfect	*	*		*			
WPG	Version 6.0	WordPerfect	*	*		*	*		

Para buscar más información sobre el tema, las siguientes páginas de Internet pueden servir como punto de partida:

http://www.cica.indiana.edu/graphics/image.formats.html

http://cui.unige.ch/eao/www/Img.File.Fmt.html

Otra buena fuente de información general para cada formato puede encontrarse [GFF95]**.** Sin embargo, si se necesita implementar un codificador o decodificador, se tendrá que acudir a la especificación de cada formato emitida por organizaciones y a los grupos de trabajo **Unsenet**, como **comp.graphics**. Dada la disponibilidad de código fuente en programas para plataformas UNIX y en bibliotecas de uso público (*freeware*), otra fuente de documentación es el propio código fuente, que aunque necesite un esfuerzo extra para poder comprender cada formato, representa una ayuda inestimable para todos los que necesitamos programar codificadores o decodificadores.

3.4 Algoritmos de compresión más usuales

La compresión de imágenes consiste en representar la información más eficientemente. Toma ventaja de tres cualidades comunes de los datos gráficos: son normalmente redundantes, predecibles o innecesarios. Un esquema como la codificación de longitud variable (RLE) que utiliza la redundancia, diría “hay tres pixel rojos idénticos” en vez de “hay un pixel rojo, hay un pixel rojo, hay un pixel rojo”. El algoritmo Huffman y la codificación aritmética se basan en un modelo estadístico para explotar la predecibilidad, utilizando códigos más cortos para los valores más frecuentes de los pixels. La presencia de información innecesaria sugiere el uso de algoritmos con pérdidas “*lossy compression*”. Por ejemplo, el ojo humano requiere menor precisión para la información del color que para la intensidad y se puede desechar altas resoluciones del color, tal como se realiza la compresión de vídeo en color.

En general existe un compromiso entre eficiencia del algoritmo de compresión y consumo de CPU, especialmente importante en secuencias de vídeo.

3.4.1 Compresión de longitud variable

Uno de los métodos más sencillos para comprimir un fichero es la codificación de longitud variable, en el cual se sustituye una serie de valores de pixels repetidos por un valor y un contador. Por ejemplo, si utilizamos letras para representar valores, la serie de valores “*abbbbbbbccdddeeddd*” se sustituiría por “1a7b2c4d2e3d”. Esta aproximación es simple de implementar y trabaja bien con cadenas largas de valores repetidos. Imágenes con grandes áreas con tono o intensidad constante, como las producidas por programas de dibujo, son buenas candidatas para esta clase de compresión.

Una implementación popular de compresión de longitud variable es *PackBits*, utilizado para mapas de bits en ordenadores Apple MacIntosh. Asumiendo datos de 8 bits, PackBits codifica valores repetidos en dos bytes, el primer byte contiene un número *n* entre -127 y -1, inclusive; indicando *-n+1* repeticiones del segundo byte. Los valores no

repetidos, como las series "*abcde*", se codifican con un código inicial *m* entre 0 y 127, inclusive; indicando la longitud de la cadena, *m+1* (*m* sería 4 en este ejemplo). Las cadenas no pueden ser más largas de 128 bytes y se deben dividir en múltiples cadenas. Generalmente, la compresión no cruza de una línea de muestreo a la siguiente.

El código de longitud variable se utiliza en bastantes formatos de mapas de bits, como MacPaint, TIFF, GEM, PCX, BMP y FLI.

3.4.2 Compresión Huffman

La codificación Huffman [HUF52] es un algoritmo común para la compresión. Trabaja sustituyendo eficientemente códigos por datos. Esta aproximación, que fue creada en 1952 para ficheros de texto, ha experimentado algunas variaciones. El esquema básico es asignar un código binario a cada valor único, con códigos variables en longitud. Los códigos más cortos son los usados para los valores más frecuentes. Por tanto es necesario añadir, al inicio del fichero, una tabla de conversión de colores ordenados por su frecuencia.

Por ejemplo, hay seis únicos valores en "*abbbcccddeeeeeeeeef*". Las frecuencias con las cuales aparecen son a:1 b:3 c:3 d:2 e:9 f:1. Para construir un código mínimo para cada uno, se construye un árbol binario como muestra la figura 1.

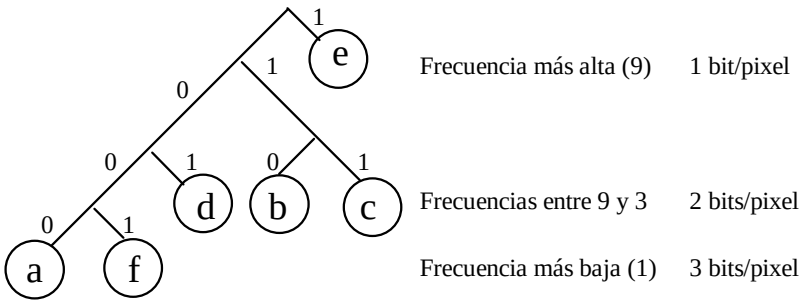


Fig. 1: Arbol binario para la codificación Huffman.

Los códigos usados menos frecuentemente son *a* y *f*, y por tanto aparecen en la parte más baja del árbol: *a* se asigna a la rama 0 y *f* se asigna a la rama 1. Esto significa que 0

y 1, respectivamente, serán los últimos dígitos significativos de *a* y *f*. Los bits más significativos vienen del árbol, tal como se ha construido.

Las secuencias de estos primeros dos son combinadas, para un total de 2. Ya que 2 es ahora la secuencia más baja, este par se combina con *d* (que además tiene una frecuencia de 2). El par original es asignado a la rama 0 de este árbol, y d se asigna a la rama 1. Entonces, el código para *a* terminará en 00; *f* terminará en 01, *d* terminará en 1 y será un bit más corto que los códigos para *a* y *f*.

El árbol continúa de esta manera, de forma que los valores menos frecuentes son descritos por códigos más largos y los más frecuentes por una secuencia muy pequeña de bits y por tanto no hay códigos desperdiciados y ninguno es más largo de lo que tendría que ser.

La eficiencia de la compresión del código Huffman varía con el algoritmo preciso y el tipo de imagen. El algoritmo varía con los diferentes formatos, pero pocos llegan a obtener un ratio de compresión mayor de 8:1. El código Huffman suele dar peores resultados para ficheros conteniendo largas filas de “pixels” idénticos, que pueden ser comprimidos juntos usando códigos de longitud variable u otras codificaciones.

La codificación Huffman necesita estadísticas precisas de la frecuencia de cada valor en la imagen original. Sin estadísticas precisas, el fichero final no presenta buena compresión e incluso puede ser más grande que el fichero original. Por tanto, para funcionar adecuadamente, Huffman se realiza en dos pasos. En la primera pasada, se crea el modelo estadístico; en el segundo, los datos son codificados. Como resultado de utilizar códigos que ocupan un número de bits variable, la compresión y descompresión son relativamente lentas.

Un problema adicional con Huffman (también encontrado en otros esquemas de compresión de longitud variable) es la sensibilidad a los bits añadidos o eliminados. Ya que todos los bits se archivan seguidos, sin importar los límites de byte, el codificador sólo conoce un código cuando el código ha finalizado alcanzando el final de una rama.

Si un byte es añadido o eliminado, el decodificador empieza en el medio de un código y el resto de los datos carecen de sentido.

3.4.3 Compresión LZW

LZW es el algoritmo más reciente para comprimir, desarrollado por Welch, y basado en el trabajo de Lempel y Ziv [WEL84]. Este algoritmo, diseñado originalmente para implementación sobre hardware, tiene variaciones muy populares. Por ejemplo, la compresión LZW se utiliza en los formatos GIF y TIFF.

Al contrario que Huffman, LZW no necesita construir una tabla de códigos antes de la codificación. Empezando con una tabla simple, construye una tabla de códigos más efectiva mientras va procesando la imagen, es decir, es adaptativo. Tampoco las implementaciones LZW utilizan códigos más cortos para los valores más frecuentes. El código LZW explota la redundancia en las tramas que encuentra.

LZW empieza con una tabla con una entrada para cada posible valor de los datos; 256 entradas, por ejemplo, para datos de 8 bits. Entonces añade entradas a la tabla para cada trama única de valores que encuentra. Es necesario fijar la longitud máxima de la tabla, de forma que la longitud del código se pueda establecer (aunque algunas implementaciones de LZW, como la usada en el formato GIF de Compuserve, permiten una longitud variable).

Consideremos, por ejemplo, la secuencia "*ababaaacaaaad*" donde cada letra representa, por simplicidad, valores de 2 bits. El codificador LZW y el decodificador empieza con la misma tabla, y realiza un seguimiento de como la tabla se expande. La tabla inicial sería, por ejemplo, la resultante de utilizar el orden alfabético:

a:00 b:01 c:10 d:11

El algoritmo LZW busca la trama más larga que puede reconocer –en otras palabras, la trama más larga que existe en la tabla–. Encuentra el primer valor, *a*, y lo reconoce, entonces intenta encontrar la trama *ab*, y no la encuentra en la tabla incial. Entonces se

almacena el valor singular de *a* (000) y se crea una nueva entrada para *ab* en la tabla de codificación con el siguiente valor disponible (100). La tabla de codificación se convierte en:

a:000 b:001 c:010 d:011 ab:100

El codificador analiza el siguiente punto no almacenado, *b*, e intenta establecer una trama para el próximo valor, *ba*. Como no encuentra la entrada para *ba*, almacena el valor singular *b* y crea una nueva entrada para *ba* en la tabla de codificación. Por tanto un código de 3 bits puede reemplazar dos códigos de 2 bits –no una ganancia muy grande en este caso, pero mucho mayor en los casos más realistas–. Incluso en este ejemplo, todas las iteraciones subsecuentes del valor *a* deberá ser introducida en la tabla y por tanto trasladada a un código de 3 bits.

La codificación LZW proporciona ratios de compresión típica entre 1:1 y 3:1, aunque imágenes con muchas tramas pueden ser comprimidas a veces hasta 10:1. Imágenes ruidosas, con variaciones aleatorias en los datos, son difíciles de comprimir con LZW. La aleatoriedad es la antítesis de la trama, y LZW se fundamenta en encontrar tramas. Para resolver problemas de ruido, los bits menos significativos pueden truncarse o se pueden promediar.

3.4.4 Compresión aritmética

La compresión aritmética, como la codificación Huffman, utiliza códigos más cortos para las secuencias más cortas y códigos largos para secuencias infrecuentes. Es un esquema más eficiente, sin embargo, que al igual que LZW comprime secuencias de valores, no solamente valores. Es más, cada esquema de codificación hace un trabajo mucho mejor en archivos en los cuales la mayoría de los datos consisten en la misma secuencia, repetida. De echo, la compresión aritmética se acerca al límite teórico de la compresión. Algunas referencias útiles de la teoría de la compresión aritmética son los artículos escritos por Abrahamson [ABR89], Langdon [LAN84], Rissanen [RIS79] y Witten [WIT87].

Hay varios algoritmos de compresión aritmética, todos de ellos muy complicados para detallar aquí, y algunos con derechos de autor. La compresión aritmética implica mapear cada diferente secuencia de valores de “pixels” a una región de un número imaginario entre 0 y 1. Esta región se representa entonces como una fracción binaria de precisión variable (número de bits variable). Las secuencias menos comunes requieren una mayor precisión (más bits).

Al igual que con la compresión de Huffman, la compresión aritmética codifica eficientemente las ocurrencias más frecuentes, es decir, los valores de los “pixels” más probables se representan con menos bits. La compresión aritmética puede reducir el tamaño de los ficheros dramáticamente, dependiendo de la fuente y la precisión del modelo estadístico empleado, incluso hasta 1:100.

3.4.5 Compresión con pérdidas

En la bibliografía se refiere a compresión con pérdidas cualquier técnica en la cual algún aspecto de la imagen original se descarta, porque es superflua para la aplicación final. Estas técnicas no pueden ser utilizadas en aplicaciones críticas como la visión por computador, pero son adecuadas para fotografía general y televisión.

La información que se descarta incluye aquellos aspectos que el ojo no necesita para percibir una situación dada, como contornos muy definidos o alta resolución espacial del color.

La compresión con pérdidas se está popularizando, debido al actual interés en aplicaciones multimedia, que generalmente incluyen animaciones o vídeo, y provocan que los diseñadores de hardware y sistemas operativos incluyan estas capacidades comP parte del sistema.

Los algoritmos con pérdidas más utilizados son los definidos en JPEG –Joint Photographic Experts Group of the International Standards Organization (ISO) [JPG91a]-. Básicamente, JPEG define una codificación con pérdidas y otra sin pérdidas

basadas en la modulación de pulsos diferenciales –aunque no se suele utilizar esta última debido a la baja relación de compresión que ofrece–. Sobre ambos esquemas se aplican múltiples algoritmos de compresión en combinación, incluidos un algoritmo Huffman modificado y la compresión aritmética.

Los algoritmos JPEG separan la crominancia de la luminancia para aprovechar la tolerancia del ojo humano para peores resoluciones del color (en las imágenes en tonos de gris sólo se almacena la luminancia). Después, la imagen se rompe en pequeñas zonas a las que se aplica la transformada discreta del coseno (DCT). Después de esta transformación, no existen "pixels" individuales como tales, sino que se representa por serie de tramas que indican cómo de rápido los valores de los "pixels" pueden variar. Dado que el concepto de pixel individual se pierde en DCT, el algoritmo JPEG se denomina algoritmo con pérdidas.

3.5 Estudio de los formatos "*raster*" más extendidos

En los siguientes subapartados se estudiarán los formatos para almacenar imágenes de mapas de "bits" que se utilizan con mayor frecuencia.

Antes de entrar en detalles, se harán algunas recomendaciones a título personal para utilizar un formato u otro dependiendo de las características de las imágenes que queramos almacenar. Estas recomendaciones o recetas están encaminadas a utilizar un formato adecuado a los propósitos que tengamos, sin necesidad de conocer las peculiaridades de cada uno.

3.5.1 Formatos recomendados para almacenar imágenes en tonos de gris o en color de alta densidad

Para imágenes que han de ser tratadas, recortadas o ajustadas, el formato TIFF con compresión LZW proporciona una buena compresión y está ampliamente soportado. Además, pocos formatos aparte de TIFF permiten ajuste fotométrico, corrección de color, ajuste de la resolución y son tan flexibles. Almacenar planos de color separado aveces aumenta la relación de compresión, pero no está soportado por todas las aplicaciones. Por compatibilidad con aplicaciones que no soportan LZW TIFF, se puede utilizar TIFF con compresión PackBits o el formato Targa.

Para la versión final de la imagen que ha de ser difundida, y que no se va a editar posteriormente, el formato JPEG es una elección razonable, ya que ofrece una compresión excelente a costa de una pérdida de precisión asociada con el almacenamiento. Para el almacenamiento, el valor del factor de calidad recomendable está entre el 70 y el 90 %. (El formato JPEG, aunque incluye un algoritmo de almacenamiento que no presenta pérdidas, no se suele utilizar y pocos programas soportan esta extensión ya que presenta una compresión muy pobre).

3.5.2 Formatos recomendados para el almacenamiento de imágenes con 256 colores o menos

Los formatos GIF o TIFF ofrecen buenas prestaciones y relaciones de compresión. El formato BMP es sencillo, rápido de manejar y está muy extendido en las plataformas PC, pero tiene una baja relación de compresión. También es común utilizar los formatos Targa y PCX.

Dentro de esta categoría se pueden incluir las imágnes con tonos de gris, ya que muchos formatos guardan una imagen con una escala de grises con una tabla de color en la cual las entradas son tonos de grises.

3.5.3 Formatos recomendados para el almacenamiento de imágenes bicolores o en blanco y negro

El formato TIFF con la compresión CCITT Grupo 4 ofrece muy buena compresión. Los algoritmos de compresión CITT Grupo 3 y Grupo 4 se crearon para crear un estándar de compresión para el envío y recepción de facsímiles que redujera el tiempo de comunicación y por tanto presentan muy alta tasa de compresión.

3.5.4 Formato IFF de Amiga

El formato "*Amiga Interchange File Format, IFF*" se utiliza para transferir documentos entre ordenadores Commodore Amiga. Además, este formato está soportado por bastantes programas gráficos en ordenadores compatibles PC.

Una variedad de este formato es el formato "*Amiga Hold and Modify, HAM*". HAM es un algoritmo de codificación que empaqueta aproximadamente el equivalente de 12 bits por pixel en 6 bits por pixel. Este formato se diseño para transferir imágenes de Windows al Amiga, aunque su difusión es reducida incluso en el sistema Windows. Originalmente, las imágenes HAM podían ser de dos tamaños, 320 x 200 y 320 x 400.

Ahora, muchos programas gráficos pueden manejar imágenes HAM no estándares con "pixels" rectangulares.

3.5.5 Formato GIF de CompuServe

El formato gráfico de intercambio de gráficos de CompuServe© "*CompuServe Graphics Interchange Format, GIF*" se utiliza ampliamente para transferir documentos en el servicio de información de CompuServe y otros servicios en línea, y para transferir documentos entre tipos distintos de ordenadores. GIF es un formato comprimido que fue diseñado para minimizar el tiempo de transferencia del fichero en líneas telefónicas.

Este formato se puede utilizar para imágenes en tonos de gris y con mapa de colores con una resolución de la imagen desde 1 bit hasta 8 (hasta 256 colores). Otra capacidad de este formato es la de poder contener varias imágenes de diferentes tamaños en un mismo fichero, contener comentarios, utilizar colores transparentes, utilizar un formato entrelazado que posibilita ver una presentación preliminar de la imagen mientras se descarga de una red lenta, etc.

Existen dos versiones de este formato, la llamada *gif87a* y la denominada *gif89*. Las especificaciones de este formato se pueden obtener a través de repetidores *SimTel*, como *cak.oakland.edu* /msdos/gif/gif89a.zip.

El formato *gif* es de uso público, pero el algoritmo de compresión que utiliza –LWZ– está actualmente patentado por Unisys. El uso de este formato en aplicaciones comerciales debe ser registrado en la compañía UNISIS. Para más información, véase las paginas WWW de Unisys (*www.unisys.com*).

3.5.6 Formato PCX de ZSoft

El formato PCX fué creado ZSoft para utilizarlo en su programa PaintShop. Este formato almacena las imágenes en planos y puede utilizar el algoritmo de compresión RLE. Utiliza un encabezado de 128 bytes para indicar el tipo de fichero y las características de la imagen.

3.5.7 Formato BMP de Microsoft

El formato Windows bitmap es utilizado por los programas que utilizan el sistema operativo Microsoft Windows e IBM OS/2, y el formato en disco BMP y memoria DIB (mapa de Bits Independiente del Dispositivo) coinciden salvo un pequeño encabezamiento que se añade al formato DIB al escribirlo en un fichero.

Este formato permite guardar de forma eficiente imágenes monocromas y con un mapa de colores. El algoritmo de compresión que utilizan los modos de 4 y 8 bits por pixel es sencillo de implementar y es rápido, pero no ahorra mucho espacio. En general, es un formato sencillo que permite cargar rápidamente los ficheros, pero que no tiene un gran coeficiente de compresión.
El fichero consiste en un encabezamiento en el cual se especifica la versión del fichero, dimensiones, tipo de compresión (RLE), y formato de color(1/4/8/24 bits por pixel) . A continuación le sigue una tabla de colores excepto si el formato es 24 bits por pixel. Finalmente, se almacenan los “pixels” de la imagen. La imagen se almacena por líneas, normalmente empezando por la esquina inferior izquierda. En los modos de 1, 4 u 8 bits por “pixels”, cada valor representa un índice de la tabla de colores. En el modo 24 bits por pixel, cada trío de bytes representan las componentes de color RGB de cada pixel.

3.5.8 Formato Scitex CT

El formato “*Scitex Continous Tone, CT*” está disponible para imágenes en color del tipoRGB y CMYK y escala de grises. Los ordenadores Scitex se utilizan para procesamiento de imágenes de alta calidad.

3.5.9 Formato Targa

El formato TGA (Targa) se diseñó para su uso en sistemas con tarjetas de vídeo Truevision© y está ampliamente difundido en aplicaciones gráficas para ordenadores compatibles PC. Permite 8, 16, 24 y 32 bits por pixel.

3.5.10 Formato TIFF

Una descripción completa del formato TIFF "*Tag Image Format*", incluyendo herramientas, documentación, etc. se puede obtener vía *ftp* anónimo en *sgi.com* en el directorio *graphics/tiff*.

Los ficheros TIFF contienen registros que indican el orden de los bytes (primer o último byte más representativo, al estilo de los procesadores Intel o Motorola), el tamaño de la imagen, resolución, número de "pixels" utilizado, tipo etc.

El formato TIFF, versión 6.0, soporta imágenes bicolor sin compresión, con compresión *CCITT 1D*, o PackBits; imágenes tipo facsímil que incluye el formato bicolor más los argoritmos de compresión CCITT Grupo 3 y CCITT Grupo 4; imágenes en tonos de gris con cuatro u ocho bits por pixel, sin compresión o con algoritmo LZW.

Las imágenes en color se pueden almacenar con una paleta de colores desde 1 hasta 8 bits por pixel, sin compresión o con algoritmo LZW; color RGB un byte por color (en total 24 bits por pixel), pudiéndose utilizar planos de color y algoritmo de compresión LZW o JPEG; color CMYK con cuatro bytes por pixel, pudiendo utilizar el algoritmo de compresión LZW o JPEG; color YCbCr con tres bytes por pixel con compresión ninguna, LWZ o JPEG; e información de colorimetría.

El algoritmo de compresión JPEG (*Huffman*) e información adicional de colorimetría se añadieron en la versión 6.0 del formato TIFF, por lo que todavía no todos los programas los incluyen. Además hay otros algoritmos opcionales de compresión como la codificación *NeXT* de dos bits, y la codificación *ThunderScan* de 4 bits, pero que apenas tienen uso.

Además proporciona un interfaz que proporciona un almacenamiento basado en filas *"scanline–oriented"* o basados en zonas rectangulares denominadas en la documentación anglosajona *"tiles"*, de igual dimensión –múltiplo de 16– y ordenadas de izquierda a derecha y de la parte superior a la inferior. Las filas y las zonas rectangulares se pueden extender en la dimensión z, para formar matrices tridimensionales de datos.

3.5.11 Formato JPEG

El formato JPEG, pronunciado *"yai-peg"*, es un método estándar de compresión para imágenes a todo color y en escalas de grises. Ha sido desarrollado por la unión de un grupo de expertos fotográficos "*Joint Photograph Expert Group*". Para obtener los detalles de las últimas versiones de este formato, se puede contactar con el *Grupo Independiente JPEG* en *jpeg-info@uunet.uu.net*.

Este formato está pensado para manejar imágenes reales, por ejemplo, fotografías digitalizadas. Dibujos lineales, caricaturas y otras imágenes no realistas producen pequeña calidad de imagen o compresión. El formato JPEG tiene pérdidas, es decir, la imagen almacenada es no necesariamente igual a la imagen inicial. Por tanto, no se debe usar el formato JPEG si es necesario conservar todos los detalles de la imagen, como por ejemplo para el tratamiento digital de la imagen. Como contrapartida se obtienen muy buenos niveles de compresión en imágenes fotográficas sin cambio perceptible, e incluso aumentar más la compresión si toleramos imágenes de baja calidad (ratios entre 50 y 150 son normales). El compromiso entre calidad y compresión se puede ajustar en la mayoría de programas. El formato JPEG tiene una variante o subformato que soporta compresión sin pérdidas, pero esta opción no suele estar implementada en los programas actuales.

Un subconjunto del formato JPEG que recibe el nombre de formato JFIF y que contiene la mayor parte de las características del formato JPEG que son de dominio público, intenta imponerse como estándar JPEG y es una de las razones de la creación del *grupo independiente JPEG*. El algoritmo de compresión más ampliamente utilizado es el *Huffman*, ya que está libre de royaltys. La codificación aritmética, patentada por IBM, AT&T, y Mitsubishi, ha de ser usada con licencia y apenas aporta mejora sobre el método Huffman, por lo que su utilización es más reducida. El formato soporta guardar las imágenes almacenadas por filas o dividiendo la imagen en bloques, o en formato progresivo, que no aporta mejoras de la compresión pero que facilita una visualización rápida mientras la imagen se transfiere a través de una red lenta.

Una buena introducción a los algoritmos de compresión JPEG se encuentra en [WAL91]. Una versión de este artículo se puede encontrar vía ftp en *ftp.uu.net,*

graphics/jpeg/wallace.ps.gz. Una introducción menos técnica pero más amena se puede encontrar en [NEL91]. La referencia más completa del formato JPEG se encuentra en [PEN93]. Las librerías y documentos del grupo independiente JPEG se pueden hallar en *ftp.uu.net* directorio graphics/jpeg (dirección de Internet 192.48.96.9). El archivo con las preguntas más usuales sobre el formato JPEG se encuentra en *ftp.uu.net* directorio /pub/usenet/news.answers/jpeg-faq/part1 y .../part2. Además, se puede encontrar información y software adicional del grupo *PVRG* de Standford en *havefun.standford.edu /pub/edu*.

3.5.12 Formato PBM

El formato *pbm*, "*portable bitmap file format*", es la especificación para ficheros de imágenes monocromas de la cual también forman parte el formato *pgm* para imágenes en tonos de gris y *ppm* para imágenes en color. Estos formatos se utilizan ampliamente en plataformas UNIX, aunque en entornos de ordenadores personales no están tan difundidos ya que en éstos se prefieren formatos comprimidos (GIF, TIFF, JPEG).

Este formato fue originalmente diseñado para intercambiar mapas de bits vía correo electrónico y por tanto su estructura es la de un fichero de texto ASCII. En la actualidad se utiliza como base de una gran familia de filtros de conversión. Dado que su definición es muy simple, se incluye a continuación (se ha tomado como fuente la página *man* pbm que acompaña a muchos paquetes gráficos como *xv*, *pbmplus*, *netpbm*, ...).

- Un "número mágico" para identificar el tipo de fichero. Para un fichero pbm, son dos caracteres: "P1".
- Separador (espacios, tabuladores, saltos de línea -LF- y retornos de carro -CR-)
- La anchura del bitmap, en formato decimal ASCII.
- Separador
- Una altura, en formato decimal ASCII.

- Anchura x altura bits, cada uno representados por ‘1’ o ‘0’, empezando por la esquina superior izquierda, en el sentido normal de lectura. El carácter ‘1’ representa un pixel negro, y ‘0’ uno blanco.
- Los separadores en la sección con los bits se ignoran.
- Los caracteres comprendidos entre “#” y el siguiente fin de línea son ignorados ya que son comentarios.
- Ninguna línea debe ser superior a 70 caracteres.

Un ejemplo de un pequeño bitmap es el siguiente:

```
P1
# feep.pbm
24 7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0
0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Una variante de este formato, es escribir el formato en formato RAW en vez de ASCII. Se diferencia en el anterior en:

- El “*número mágico*” es “P4” en vez de “P1”.
- Se almacenan ocho bits por byte, siendo el primer bit el más significativo y el último el menos significativo. No se permiten separadores en esta sección, y sólo debe haber un carácter separador (normalmente un salto de línea) después de la altura.
- Los ficheros ocupan la octava parte y se leen y escriben más rápidos.

3.5.13 Formato PGM

El formato *pgm*, "*portable graymap file format*", es la especificación para ficheros de imágenes en tonos de gris. Dado que su definición es muy simple, se incluye a continuación.

- Un "*número mágico*" para identificar el tipo de fichero. Para un fichero pgm, son dos caracteres: "P2".
- Separador (espacios, tabuladores, saltos de línea -LF- y retornos de carro -CR-)
- La anchura del bitmap, en formato decimal ASCII.
- Separador.
- Una altura, en formato decimal ASCII.
- El máximo valor de la escala de grises, en formato decimal ASCII.
- Separador.
- Anchura x altura valores de tonos de gris, representados en formato decimal ASCII, empezando por la esquina superior izquierda, en el sentido normal de lectura. Un valor '0' representa el negro, y el máximo valor de la escala representa el negro.
- Los separadores en la sección con los bits se ignoran.
- Los caracteres comprendidos entre "#" y el siguiente fin de línea son ignorados ya que son comentarios.
- Ninguna línea debe ser superior a 70 caracteres.

Un ejemplo de un pequeño bitmap es el siguiente:

```
P2
# feep.pgm
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0 15 0
```

```
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11  0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0  0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15  0  0  0 0
0 0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0 0
```

Una variante de este formato, es escribir el formato en formato RAW en vez de ASCII. Se diferencia en el anterior en:

- El "*número mágico*" es "P5" en vez de "P2".
- Los valores de los tonos de gris se almacenan como bytes, en vez de números decimales ASCII. En esta sección no se permiten separadores en la sección de los *bits*, y sólo debe haber un carácter separador (normalmente un salto de línea) después de la altura. El mayor valor que puede utilizar la variante RAW para representar un tono de gris es 255.
- Los ficheros son más pequeños y se leen y escriben más rápidos.

3.5.14 Formato PPM

El formato *pgm*, "*portable pixmap file format*", es la especificación para ficheros de imágenes en color RGB. Dado que su definición es muy simple, se incluye a continuación.

- Un "*número mágico*" para identificar el tipo de fichero. Para un fichero ppm, son dos caracteres: "P3".
- Separador (espacios, tabuladores, saltos de línea -LF- y retornos de carro -CR-)
- La anchura del bitmap, en formato decimal ASCII.
- Separador.
- Una altura, en formato decimal ASCII.
- El máximo valor de las componentes de los colores, en formato decimal ASCII.
- Separador.
- Anchura x altura "pixels", representados cada uno con tres números en formato decimal ASCII, empezando por la esquina superior izquierda, en el sentido normal de lectura. Los tres números para cada pixel representan la componente roja, verde y azul, respectivamente. Un valor cero significa la ausencia de ese color, y el máximo significa que el pixel está saturado de ese color.
- Los separadores en la sección con los bits se ignoran.

- Los caracteres comprendidos entre "#" y el siguiente fin de línea son ignorados ya que son comentarios.
- Ninguna línea debe ser superior a 70 caracteres.

Un ejemplo de un pequeño bitmap es el siguiente:

```
P3
# feep.ppm..
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```

Una variante de este formato, es escribir en formato RAW en vez de ASCII. Se diferencia en el anterior en:

- El "*número mágico*" es "P6" en vez de "P3".
- Los valores de los componentes del color se almacenan como bytes, en vez de números decimales ASCII. En esta sección no se permiten separadores en la sección de los *bits*, y sólo debe haber un carácter separador (normalmente un salto de línea) después de la altura. El mayor valor que puede utilizar la variante RAW para representar una componente de color es 255.
- Los ficheros son más pequeños y se leen y escriben más rápidos.

3.5.15 Formato XWD

Xwd es un formato de imágenes exclusivo del sistema *X Window* y que se utiliza para volcar imágenes de ventanas a un archivo. Este formato es casi exclusivo de esta plataforma. La definición de este formato se puede obtener en el fichero *XWDFile.h* que se puede encontrar en el directorio de ficheros *include* de sistemas UNIX.

3.5.16 Formato XBM

El formato *"X BitMap"* proporciona una forma fácil de incluir pequeñas imágenes monocromas en programas escritos en C o C++, ya que el fichero en sí se puede incluir en tiempo de compilación al programa. Es el más utilizado para incluir *bitmaps* en programas en plataformas X window pero en ordenadores personales no está muy difundido.

La estructura del formato es como sigue:

- la primera línea que empieza con "`#define`", un identificador "`imagen_width`" y un número decimal, representa la anchura del dibujo
- la segunda línea que empieza por "`#define`", un identificador"`imagen_height`" y un número decimal, representa la altura del dibujo
- Una definición de un vector de bytes (representados por un vector de caracteres al estilo C, definido mediante números hexadecimales de los bytes). Los bits están almacenados empezando por la esquina superior izquierda, en el sentido de lectura normal, en grupos de ocho bits por byte, empezando a rellenar el byte por el bit menos significativo. La presencia del bit correspondiente indica que el pixel es negro. Si el bit no está presente, indica el color blanco. Si la imagen no tiene una anchura múltiplo de 8, los bits se alinean en una frontera de tamaño byte, de forma que cada línea empiece en un byte.

Como resultado, el código es compatible con el lenguaje C. Un ejemplo de una imagen de 24x24 bits es el siguiente:

```
#define wait4_width 24
#define wait4_height 24
static unsigned char wait4_bits[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3e, 0x00,
  0x80, 0xc1, 0x00, 0x40, 0x00, 0x01, 0x60, 0x00, 0x03, 0xf0, 0x80, 0x07,
  0xf0, 0xc1, 0x07, 0xf8, 0xe3, 0x0f, 0xf8, 0xf7, 0x0f, 0xf8, 0xff, 0x0f,
  0xf8, 0xf7, 0x0f, 0xf8, 0xe3, 0x0f, 0xf0, 0xc1, 0x07, 0xf0, 0x80, 0x07,
  0x60, 0x00, 0x03, 0x40, 0x00, 0x01, 0x80, 0xc1, 0x00, 0x00, 0x3e, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  };
```

3.5.17 Formato XPM

El origen del formato *"X PixMap"* estuvo en la necesidad de tener un formato adecuado para pequeños iconos, como los que se utilizan en aplicaciones *X Window* ya que los formatos existentes como *gif, tiff, iff,* etc... se habían diseñado para grandes imágenes. El formato XPM fue diseñado para:

- poder ser modificado con un editor de texto;
- ser incluido en un programa en lenguaje C;
- ser portable, poder ser enviado por correo electrónico y codificado en ASCII;
- poder manejar imágenes monocromas, en color y en tonos de gris;
- poder sustituir colores;
- poder incluir comentarios en el fichero;
- sin compresión. La compresión debe ser proporcionada por otros medios externos, como un compresor de ficheros.

El formato se presenta como un vector de seis cadenas de caracteres.

```
/* XPM */
static char* <variable name>[] = {
<Valores>
<Colores>
<Pixels>
<Extensiones>
};
```

Las palabras están separadas por espacios y tabuladores.

La sección `Valores` es una cadena que contiene cuatro o seis números decimales que corresponden a: la anchura y altura de la imagen, el número de colores, el número de caracteres por pixel (`<cpp>`, de esta forma no hay limite de número de colores), y opcionalmente las coordenadas del *hotspot* y la etiqueta *XPMEXT* si hay cualquier

extensión después de la sección <Pixels> –*hotspot* es útil para utilizar la imagen como botón de un programa–.

```
<Anchura> <Altura> <nº colores> <cpp> [<x_hotspot> <y_hotspot>]
                                [XPMEXT]
```

La sección Colores contiene tanta cadenas como los colores, y cada cadena es como sigue:

```
<caracteres> {<clave> <color>} +
```

donde <caracteres> es la cadena (de longitud <cpp>) con la que se representa el color <color> y <clave> describe en qué contexto este color debe ser usado. Actualmente las claves son:

```
m    visualización en dispositivos monocromos
s    nombre simbólico
g4      visualización con cuatro niveles de gris
g    escala de gris con más de cuatro niveles
c    visualización en color
```

Los colores pueden ser especificados dando el nombre del color en inglés, con el código hexadecimal RGB precedido del carácter # o con el código hexadecimal HSV precedido del carácter %. Además se puede utilizar un color “transparente” especificando la cadena None .

La sección <Pixels> está compuesta por <altura> cadenas con <anchura> * <cpp> caracteres. Cada uno de las grupos de caracteres de longitud <cpp> debe estar previamente definido en la sección <Colores>.

La sección <Extensiones> debe estar etiquetada si no está vacía en la sección <Valores> como se ha descrito anteriormente. Esta sección puede estar formada por varias subsecciones, que pueden ser de dos tipos:

– Un único grupo

  `XPMEXT <nombre_extensión> <datos_extensión>`

– Un bloque formado por varias cadenas

  `XPMEXT <nombre_extensión>`
  `<datos relacionados con la extensión dispuestos en varias cadenas>`

Finalmente, si la extensión no está vacía, debe terminar con la cadena

  `XPMEXTEXT`

Para evitar posibles conflictos con las extensiones de ficheros, debe estar precedido por el nombre de la compañía (de esta forma se asegura la unicidad).

Un ejemplo de un archivo XPM es el siguiente pixmap, de tamaño 14x17, con cuatro colores y dos caracteres por pixel. Las coordenadas *hotspot* son (0,0). Hay símbolos y colores por defecto para visualización monocroma y en color.

```
/* XPM */
static char *plaid[] = {
/* plaid pixmap
* width height ncolors chars_per_pixel */
"17 14 4 2 0 0 XPMEXT",
/* colors */
"  c red    m white s light_color ",
"Y c green  m black s lines_in_mix ",
"+ c yellow m white s lines_in_dark ",
"x m black  s dark_color ",
/* pixels */
"x   x   x x x   x   x x x x x x + ",
"  x   x   x   x   x   x x x x x x ",
"x   x   x x x   x   x x x x x x + ",
"Y Y Y Y Y x Y Y Y Y Y + x + x + x ",
"x   x   x x x   x   x x x x x x + ",
"  x   x   x   x   x   x x x x x x ",
"x   x   x x x   x   x x x x x x + ",
"          x           x   x   x Y ",
"          x             x   x   Y ",
"          x           x   x   x Y ",
"x x x x x x x x x x x x x x x x x ",
"          x           x   x   x Y ",
"          x             x   x   Y ",
"          x           x   x   x Y ",
"XPMEXT ext1 data1",
"XPMEXT ext2",
"data2_1",
"data2_2",
"XPMENDEXT"
};
```

3.5.18 Formato RAW

El formato Raw produce un tipo flexible de ficheros para transferir documentos entre aplicaciones y plataformas de ordenadores. El formato Raw consiste en un flujo de *bytes* describiendo la información de color en el fichero. Cada pixel es descrito en formato binario, en el cual 0 equivale negro y 255 equivale a blanco. Algunas aplicaciones designan el número de canales o planos necesarios para describir la imagen. Se pueden especificar extensiones del archivo así como información de cabecera.

El parámetro de la cabecera es el número de bytes de información que aparecerá al principio del documento, antes de los datos de la imagen propiamente dicha. Este valor define el número de ceros que se situarán al inicio del documento como relleno. Por defecto, no hay cabecera (el tamaño de la cabecera es 0). La cabecera se puede modificar con un editor de archivos. La especificación de este formato se puede obtener en Microsoft (www.microsoft.com).

La imagen se puede guardar con una codificación intercalada "*interleaved*" o no. En la codificación intercalada, los valores de color (como rojo, verde y azul) se almacenan secuencialmente. En la codificación no inercalada, cada componente del color se almacena en un plano diferente. La elección final depende de los requerimientos de la aplicación que se vaya a utilizar para abrir el documento.

3.5.19 Formato Vista

Este formato es el utilizado en el programa de visión artificial *Vista* y su utilización se reduce casi exclusivamente a este programa. La definición de este formato se puede encontrar en las páginas de ayuda *man Vfile* y *man VImage*.

Las características que lo diferencian de los demás formatos gráficos es la posibilidad de contener objetos distintos de imágenes, sus grandes posibilidades de expansión y poder guardar imágenes codificadas no sólo con valores enteros como utilizan el resto

de formatos, sino como números de coma flotante. Permite guardar las imágenes por planos, diferentes profundidades de color e incluso varias imágenes en un único archivo.

3.5.20 Formato PNG

Este nuevo formato intenta convertirse en un nuevo estándar y reemplazar el formato GIF, que tiene problemas legales debido a la patente del algoritmo que utiliza (Lempel-Zip & Welch). En enero de 1995 se crea un consorcio de desarrolladores gráficos y en mayo de 1995 se aununcia la primera revisión del formato de archivo de imágenes PNG (ping). Las definición de este formato se puede obtener en ftp.uu.net /graphics/png/documents y en la página de Web www.group42.com, además de una API portable, que permite leer y grabar PNG y una librería de compresión. Utilidades para la conversión de este formato se pueden obtener también en ftp.uni-erlangen.de.

El formato PNG no está limitado a las imágenes, y debido a su particular estructura puede contener textos y otros componentes. Presenta una compresión del 100% sin pérdida de información, mayor ratio de compresión que la realizada con el método GIF/LZW, soporta imágenes de color real de hasta 48 bits por píxel y de 16 bits para los tonos de gris, y con una paleta de 256 colores, y la posibilidad de mostrar la imagen de forma progresiva

3.5.21 Otros formatos.

Las estaciones de trabajo Sun utilizan además el formato SUN para imágenes *raster* (para más información www.sun.com). Las estaciones Irix pueden utilizar el formato SGI para imágenes en color RGB (la información se puede obtener en la dirección internet. El paquete de herramientes Magick Image utiliza el formato MIFF, el paquete para tratamiento gráfico Khoros utiliza el formato VIFF y el paquete Utah utiliza en formato RLE (Utah Run Leng encoded image file). En plataformas Apple Macintosh es común el formato PICT, utilizado en programas MacPaint y QuickDraw. Además de los formatos nombrados, muchos programas gráficos definen su propio formato.

3.6 Estudio de los formatos *vectoriales* y *meta–ficheros* más extendidos

Los meta–ficheros y los ficheros de vectores pueden contener información geométrica de figuras, segmentos, objetos, texto, etc. para representar una imagen. La imagen se guarda como una colección de fórmulas matemáticas. Las figuras vectoriales no pueden reproducir imagen foto–realistas, pero cuentan con dos ventajas sobre los formatos "*raster*": se pueden escalar sin distorsión y producen ficheros más pequenos.

En el sentido estricto, un formato vectorial sólo puede contener información sobre segmentos. En la práctica común, bastantes formatos considerados vectoriales permiten al usuario incluir datos no vectoriales, como imágenes de mapas de bits y texto.

Los meta–ficheros explícitamente permiten más que datos vectoriales. Por ejemplo, un meta–fichero de *Windows* puede contener un mapa de bits, información vectorial, y texto.

En la siguiente tabla se muestran los formatos más usuales:

Formato	Fuente / Estándar
CDR	CorelDRAW!
CGM	Computer Graphics Metafile
DRW	Micrografx Draw
DXF	Autodesk
GEM	Ventura / GEM
HGL	Heweltt-Packard Graphics Language
PIC	Lotus development Corp.
WMF	Microsoft Windows Metafile
WPG	WordPerfect
PS	PostScript
EPS	Encapsulated PostScript

3.6.1 Formato PostScript

Una imagen PostScript es el conjunto de instrucciones que se envían a una impresora de tipo PostScrip para imprimir una imagen. Puede contener texto, imágenes vectoriales e imágenes *raster*. Una descripción más detallada del lenguaje PostScrip se encuentra en [ADO90].

El archivo PostScript se puede guardar en formato binario o ASCII. El primero ocupa aproximadamente el doble de espacio, pero presenta problemas de compatibilidad con algunos programas. El formato ASCII, por otra parte, puede ser modificado con un editor de texto.

Adobe ha definido una extensión de las convenciones de los comentarios incluidos en los ficheros PostScrip, que proporcionan información adicional sobre la estructura de la página y los requerimientos de recursos del fichero y que se denominan "*Document Structuring Convention (DSC)*". Estos ficheros empiezan con el siguiente comentario:

```
%!PS-Adobe-3.0
```

donde el número 3.0 puede cambiar y el número de versión DSC.

Los ficheros PostScript encapsulado (EPSF) son documentos con una simple página y con la dimensión que se indica en el fichero, que contienen un subconjunto de los comentarios DSC y de los comandos PostScrip y comienzan con el comentario:

```
%!PS-Adobe-3.0 EPSF-3.0
```

Algunos ficheros EPS contienen una vista preliminar del documento, que puede ser un Meta–archivo de Windows, un fichero TIFF o un fichero para intercambio con formato EPSI. De todas formas, cuando se imprimen, utilizan la imagen original PostScrip, no la vista preliminar.

3.6.2 Formato EPS

El lenguaje PostScrip Encapsulado "*Encapsulated PostScript, EPS*" está ampliamente difundido en la mayoría de los programas de ilustración y composición de páginas, y en la mayoría de los casos, es el formato preferido por estas aplicaciones. Además EPS es el único formato que admite blanco transparente en los mapas de bits.

Básicamente es un subconjunto del lenguaje PostScript, que se limita a gráficos contenidos en una única página, y que contiene los siguientes dos comentarios en el DSC que lo diferencia de un fichero PostScrip convencional:

```
%!PS-Adobe-3.0 EPSF-3.0
```

y la dimensión del documento

```
%%BoundingBox: Xlowleft Ylowleft Xupright Yupright
```

Este formato no es un formato de imágenes *raster*, sino imágenes tipo *metafile* que puede contener gráficos vectoriales, texto e imágenes raster en su interior. Además, puede contener una imagen de presentación "*preview*" guardada en formato TIFF o JPEG, que da una idea de como va a quedar la imagen cuando sea impresa sin necesidad de decodificar el documento y facilita el emplazamiento de la imagen en la página.

Hay dos tipos de codificación: ficheros EPS binarios o en formato ASCII. La codificación binaria reduce el tamaño del fichero a la mitad aproximadamente y se tarda la mitad de tiempo en transferirlo a la impresora, pero hay software comercial de cola de impresión e impresión en red que no soporta la codificación binaria. El fichero ASCII es más transportable y está más difundido.

Una extensión estándar del EPS es el formato DCS, llamado "*Desktop Color Separations*", desarrollado por Quark. Este formato es utilizado por ciertas aplicaciones como QuarkXPress. Utiliza el formato de color CMYK o tonos de gris y contiene información sobre el color de alta resolución.

3.6.3 Microsoft Windows Metafile (WMF)

Este formato independiente del dispositivo produce ficheros estructurados y compactos, y está ampliamente soportado en plataformas PC. Un fichero WMF almacena la lista de las llamadas a funciones gráficas de Microsoft Windows, de un gran subconjunto de las que dispone el sistema Microsoft Windows.

Aunque originariamente el metafichero fue diseñado como una macroinstrucción gráfica, se volvió de uso general y ahora se utiliza a menudo como formato de intercambio de imágenes entre aplicaciones Windows.

Este metafichero consiste en una cabecera seguida por algunos registros. Una cabecera adicional de localización puede preceder el inicio del archivo. Cada registro corresponde a una llamada del interfaz Windows de dispositivo gráfico (GDI), que contiene el tamaño, el número de función, y algunos datos de los argumentos. En la mayoría de los casos, los argumentos son simplemente los valores que se pasarán a la rutina correspondiente del dispositivo gráfico, aunque para las rutinas más complicadas, la codificación es más compleja.

3.6.4 Computer Graphics Metafile (CGM)

El formato CGM tiene la aspiración de ser un formato para el almacenamiento e intercambio de ficheros independiente del hardware y del sistema operativo, y se limita a dos dimensiones. Se ha inspirado en el Núcleo del Sistema Gráfico GKS (ISO 7942), un conjunto de subrutinas gráficas, y hay una relación íntima entre las rutinas del Núcleo del Sistema Gráfico y los elemntos de CGM. El nuevo sistema de jerarquía interactiva del programador PHIGS es una librería 3-D, y por tanto se espera que aparezca la versión tridimensional de CGM correspondiente.

CGM es un formato muy rico, con soporte para líneas, círculos, elipses, polígonos, cadenas de texto y bitmaps (una descripción completa ocuparía cientos de páginas). Este archivo contiene una secuencia de elementos correspondientes a los comandos definidos en el lenguaje CGM. Un único metafichero puede contener varias imágenes, las cuales pueden ser imágenes separadas o pertenecientes a una secuencia de animación.

Aunque los ficheros CGM aspiran a ser portátiles e independientes de los dispositivos, en la teoría una implementación del formato a menudo no puede leer con éxito un fichero generado por otra implementación, debido a que el estándar es tan grande que muchas realizaciones no pueden manejar cada elemento definido; el formato de imagen es más general que el que pueden representar muchos dispositivos; la interpretación de algunos elementos se ha dejado deliberadamente ambigua y el nombre de las fuentes de texto no está estandarizado.

3.6.5 Hewlett-Packard Graphics Language (HPGL)

Este formato para el dibujo de líneas, está muy extendido, es fácil de producir e independiente del tamaño del papel, aunque puede ser necesario dibujar piezas muy grandes en varios trozos. Como desventaja está que el formato es de bajo nivel y que está optimizado a dibujar líneas.

El formato HPGL es el lenguaje de comandos de los trazadores de plumilla de Hewlett-Packard. Debido a la amplia difusión de los trazados de HP, HP-GL se ha convertido en un estándar y está ampliamente implementado en trazadores de plumilla e impresoras láser (es más, el lenguaje PCL de las impresoras Hewlett-Packard incluye las instrucciones del formato HPGL). Al contrario que la mayoría de los formatos de dibujo, HPGL consiste casi por completo en caracteres ASCII leíbles, lo cual hace fácil producir y depurar los ficheros.

HPGL fue originalmente implementado en 1976 en las trazadoras HP de la serie 7400, y a partir de entonces se ha vuelto a implementar muchas veces, ya sea en la empresa Hewlett-Packard o en otras. En 1990 HP lanzó HP-GL/2, un formato casi totalmente compatible con el anterior HP-GL.

3.6.6 Drawing Interchange Format (DXF)

Debido a la popularidad de AutoCAD de Autodesk, el formato de intercambio DXF es soportado por otros programas de CAD, incluso en plataformas distintas del compatible PC. Es un formato estándar público que proporciona un medio para exportar dibujos de CAD a aplicaciones de otro tipo. Tiene todas las ventajas del formato vectorial, más la ventaja de ser un formato vectorial tridimensional y por tanto, capaz de comunicar figuras tridimensionales, incluyendo estructuras de alambres y planos rellenos.

Se puede asignar códigos de color de una tabla de 256 a las figuras, pero no es necesario que la tabla esté relacionada con unos colores determinados.

La codifiación ASCII del formato DXF es la más usada, aunque su lectura es extremadamente lenta. Las aplicadciones que utilicen DXF deben poder realizar cálculos en al menos dos dimensiones, tratar texto, ser capaces de ajustar curvas, etc. El formato binario es aproximadamente un 25 por ciento menor y es leído cinco veces más rápido.

BIBLIOGRAFÍA:

A parte de las direcciones URL nombradas en cada subsección, y de la bibliografía allí señalada, a continuación se muestran otras referencias útiles, agrupadas por formatos.

- Bibliografía básica

[GFF95] D.C. Kay J.R. Levine: "Graphics File Formats". 1995 McGraw-Hill.

- Algoritmos de compresión

[ABR89] Abrahamson, D.M. "An adaptative Dependency Source Model for Data Compression". Communications of the ACM, vol. 32, nº. 1. Enero 1989.

[HUF52] Huffman, D.A. "A method for the construction of minimum-redundancy codes." Proc. Inst. Electr. Radio Eng. vol. 40, nº. 9. Septiembre 1952

[LAN84] Langdon, G.G. Jr. "An Introduction to Arithmetic Coding." IBM Journal of Research and Development, vol. 28, nº 2. March 1984.

[RIS79] Rissanen, J.J., and G.G. Langdon, Jr. "Aritmetic Coding." IBM Journal of Research and Development. December 1979.

[WEL84] Welch, T.A. "A Technique for High Perfomance Data Compression." IEEE Computer, vol. 17, nº. 6. Junio 1984.

[WIT87] Witten, I.H., R.M. Neal, and J.G. Cleary. "Arithmetic coding for data compression." Communications of the ACM, vol. 30, nº. 6. Junio 1987

- PCX

[PCX91] Zsoft Corporation. Technical Reference Manual. 1991.

[RIM90] Rimmer, Steve. Bit-Mapped Grapics. 1990. Blue Ridge Summit, PA: Windcrest Books, an imprint of TAB BOOKS.

[WIL87] Wilton, Richard. Programmer's Guide to PC& PS/2 Video Systems, 1987. Redmond, WA: Microsoft Press.

- MacPaint

[APP84] "MacPaint Document Format." Macintosh Technical Note #86. Apple Computer Developer Technical Support.

[CAM 90] Birse, Cameron, Guillermo Ortiz, an Jon Zap. "Things You Wanted to Know About _PackBits", Macintosh Technical Note #171. 1987, 1990. Apple Computer Developer Technical Support.

[APP88] Apple Computer, Inc. Inside Macintosh. 1988. Reading, MA: Addison-Wesley Publishing Co.

- TIFF

[TIF92] Aldus Corporation Developer's Desk. TIFF Revision 6.0, Final. June 3, 1992.

[CIT85] International Telegraph an Telephone Consultativa Committee (CCITT). Terminal Equipement and Protocols for Relematic Services, Vol. VII, Fascicle VII.3, pages 16-31 and 40-48, respectively, for Group 3 and Group 4 facsimile. 1985. Geneva.

[CIE86] Commision Internationales de l'Eclarage. CIE Colorimetry: official Recommendations of the International Commision on Illumination, Publication 15-2. 1986.

- GIF

[GIF89] Compuserve Incorporated. Grafics Interchange Format. Version 89a. 1989. Columbus, Ohio

[GIF87] Compuserve Incorporated. Grafics Interchange Format. Version 87a. 1987. Columbus, Ohio.

- IFF/ILBM

[IFF85] Morrison, Jerry. EA IFF 85 Standard for Interchange Format Files. January 14, 1985. Electronic Arts.

- TARGA

[TGA91] Truevision Targa File Format Specification Version 2.0. Technical Manual Version 2.2. January 1991. Indianapolis, IN: Truevision Inc.

- MICROSOFT WINDOWS DEVICE INDEPENDENT BITMAP (BMP)

[MIC90] Microsoft Corp. Microsoft Windows Programmer's Reference. 1990. Microsoft Press.

[VC+95] Microsoft Corp. Microsoft Visual C++ 4.0 Programmer's Reference. 1995. Microsoft Press.

- SUN RASTERFILE

[SUN84] Sun Mcrosystems. "Rasterfile (5)". Sun OS 4.0 Programmer's Manual.

[GEE84] McGee, Pat. "Format For Byte-Encoded Rasterfiles," Sun-Spots Digest, Volume 6, Issue 84.

- X Window bitmaps

[XLI89] Nye, Adrian. Xlig Programming Manual, pp. 167-68. 1989. O'Reailly and Associates, Inc.

[MIT91a] Gettys, James, Robert W. Scheifler, et al. "Aplication Utitlity Functions" (Chapter 16), Xlig-C Language X Interface, MIT X Consortium Standard, Xversion 11, Release 5, Firs Revision. August 1991.

- X Window Dump

[XLI89] Nye, Adrian. "Xlig Programming Manual", pp. 167-68. 1989. O'Reailly and Associates, Inc.

[MIT91b] MIT X Window Consortium. "xwd.c and X11/XWDFile.h," X Window System Version 11 Release 5. 1991. Network Distribution.

- JPEG

[JPG91] International organization for Standards, Digital Compression and Coding of Continuous-Tone Still Images. Part 1: Requiremets and guidelines. 1991. New York: ANSI.

[HAM92] Eric Hamilton. "JPEG File Interchange Format". Version 1.02. September 1, 1992. San Jose CA: C-Cube Microsystems, Inc.

[IJG94] Thomas G. Lane. "IJG Group Library". Independent JPEG Group's software, 1994.

[WAL91] Wallace, Gregory K. "The JPEG Still Picture Compression Standard", Communications of the Association for Computing Machinery, April 1991 (vol. 34 no. 4), pp. 30-44.

[NEL91] Mark Nelson. "The Data Compression Book" M&T Books (Redwood City, CA), 1991, ISBN 1-55851-216-0.

[PEN93] William B. Pennebaker y Joan L. Mitchell "JPEG Still Image Data Compression Standard". Van Nostrand Reinhold, 1993. (incluye los estándares ISO DIS 10918-1 y el borradorDIS 10918-2).

- FITS

[FIT81] Wells, D.C. and E.W. Greisen. FITS: A Flexible Image Transport System. Astron. Astrophys. Suppl. Ser. 44: 363-370. 1981.

[FIT93] NASA/OSSA Office of Standards and Technology. Definition of the Flexible Image Transport System. Standard NOST 100-1.0. June 18, 1993. Greenbelt MD: NASA Goddard Space Flight Center.

- DXF Drawing Interchange Format

[DXF90] Autodesk, Inc. AutoCAD Reference Manual, 1990

[DXF91] Bourke, Paul. Minimum Requirements for Creating a DXF File of a 3D Model. 1991. Auckland, NZ: School of Architecture, Property and Planning, Auckland University, Private Bag.

- HP-GL

[HPG01] Hewlett-Packard. The HP-GL/2 Reference Guide: A Handbook for Program Developers. 1990. Reading, MA: Addison-Wesley.

[HPG02] Hewlett-Packard. HP-GL vs. HP-GL/2 Language Comparison Guide, Rev. 1.0. 1991. San Diego: Hewlett-Packard.

[HPG03] Hewlett-Packard. The HP-GL/2 Product Comparison Guide, Pub. No. 5959-9734. 1990. San Diego: Hewlett-Packard.

[HPG04] Hewlett-Packard. HP PCL 5 Printer Language Technical Reference Manual, Pub. No. 33459-90903. 1990. Boise, ID: Hewlett-Packard

[HPG05] Hewlett-Packard. PCL 5 Developer’s Guide, Third Edition, Part No 5002-1847. 1991.

[HPG06] Hewlett-Packard. PCL 5 Comparison Guide, Third Edition, Part No 33481-90968.1991.

[HPG07] Hewlett-Packard. LaserJet Series II, LaserJet IIP, LaserJet IIID and LaserJet 4L Printer Technical Rerefence Manuals.

- PostScript

[ADO90] Adobe Corporation. "PostScript Language Reference Manual", Second Edition. 1990 Addison-Wesley.

- WMF

[WMF90] Microsoft Corp. Microsoft Windows Programmer's Reference. 1990. Redmond. WA: Microsoft Press.

- CGM

[CGM01] American National Standards Institute/International Organization for Standards. ANSI/ISO 8632-1987.1. Information Processing Systems– Computer Graphics–Metafile for the Storage and Transfer of Picture Description Information–Part 1: "Functional Description." 1987. New York: ANSI. (Formerly ANSI X3.122-1986.)

[CGM02] American National Standards Institute/International Organization for Standards. ANSI/ISO 8632-1987.2. Information Processing Systems– Computer Graphics–Metafile for the Storage and Transfer of Picture Description Information–Part 2: "Character Encoding." 1987. New York: ANSI. (Formerly ANSI X3.122-1986.)

[CGM03] American National Standards Institute/International Organization for Standards. ANSI/ISO 8632-1987.3. Information Processing Systems– Computer Graphics–Metafile for the Storage and Transfer of Picture Description Information–Part 3: "Binary Encoding." 1987. New York: ANSI. (Formerly ANSI X3.122-1986.)

[CGM04] American National Standards Institute/International Organization for Standards. ANSI/ISO 8632-1987.4. Information Processing Systems– Computer Graphics–Metafile for the Storage and Transfer of Picture Description Information–Part 4: "Clear Text Encoding." 1987. New York: ANSI. (Formerly ANSI X3.122-1986.)

[CGM05] American National Standards Institute/International Organization for Standards. ANSI/ISO 8632-1987.5. Information Processing Systems– Computer Graphics–"Programmer's Hierarchical Interactive Graphics Systems (PHIGS)." 1989. New York: ANSI. (Formerly ANSI X3.144-1986.)

Capítulo 4:

Generalidades del diseño del entorno de visión

4.1 Introducción

A lo largo de los capítulos anteriores los entornos de visión disponibles actualmente han sido estudiados y se han examinado los formatos gráficos para almacenar imágenes en ficheros. Ahora que ya sabemos qué características son deseables para el entorno de visión y qué tipos de formatos gráficos debe ser capaz de leer, es el momento de planificar la estructura interna del entorno de visión.

En este capítulo se intentará dar una visión general del entorno, de cómo están hechas las cosas y por qué se han hecho así. En la primera parte del capítulo se justificará la elección del lenguaje de programación con el cual se va a realizar el entorno, que de ahora en adelante lo denominaré "Entorno de visión articial `Visio`" o simplemente "`Visio`". A continuación se presentará la estructura de las clases utilizadas en el caso más sencillo: cuando construimos el entorno en modo texto (dado que en modo texto sólo se utilizan las bibliotecas estándar de *C* y no existen las complejidades asociadas al entorno de ventanas, este caso es muy ilustrativo). La visión del entorno en modo texto será suficiente para que cualquier desarrollador de algoritmos tenga la información necesaria para realizar su cometido.

Una vez que se ha estudiado y comprendido la disposición del entorno en modo texto, se procederá a comentar la organización del programa en una interfaz de ventanas. Dada la complejidad de las aplicaciones que utilizan ventanas, este apartado sólo será necesario para las personas que quieran ampliar las capacidades gráficas del entorno de visión. Aunque intentaré simplificar y esquematizar el entramado de clases que forman una aplicación gráfica, sería conveniente que quien leyera este apartado estuviera familiarizado con la arquitectura de programación "*documento–vista"* y con las clases `Microsoft Fundation Classes`.

4.2 Programación orientada a objetos.

El diseño de software tiene una complejidad inherente, que aumenta proporcionalmente con la dificultad de la tarea que desarrolla. El entorno de visión se va a desarrollar en lenguaje C++, que aúna la rapidez y la potencia del lenguaje C con la metodología de la programación orientada a objetos. Esto nos permite diseñar un entorno de visión con las siguientes propiedades:

• Encapsulamiento u ocultación de la complejidad interna de las entidades: Combinando un registro o estructura de datos con los procedimientos y funciones que lo manipulan se obtiene un objeto.

• Herencia: Definiendo un objeto y utilizandolo para construir una jerarquía descendente de objetos, se permite el acceso a todos los datos y algoritmos inherentes de los antecesores.

• Polimorfismo: Se puede dar el mismo nombre a una acción que es compartida por la jerarquía de objetos, pero que se realiza de forma distinta según las características particulares de cada objeto.

Además, proporciona ocultación de datos, garantiza la inicialización de los datos, la conversión implícita y un mecanismo para añadir nuevas capacidades a los operadores. Los datos y los algoritmos se asocian en objetos. Se puede acceder a la información interna del objeto a través de métodos, que eviten tener que recordar la estructura interna de los datos.

4.3 Jerarquía de objetos utilizada en el entorno de visión en modo texto

El entorno de visión se ha dividido en tres capas. La inferior contiene la estructura de los datos en memoria y los procedimientos para leer y escribir en ficheros las imágenes contenidas en memoria. A continuación se le añaden los algoritmos, y en la última capa se incluye la interfaz de usuario. Para facilitar el mantenimiento y la ampliación del entorno, cada capa está implementada como una clase.

El esquema de clases utilizado para trabajar en entorno texto (tipo terminal de texto) es el siguiente:

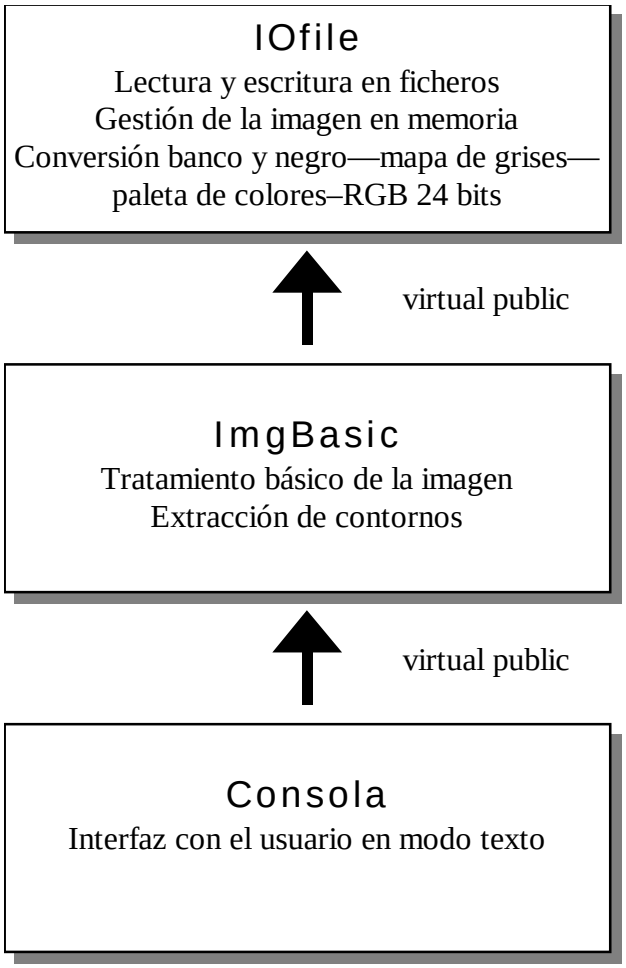


Fig. 1: Proceso de derivación de las clases utilizadas en modo texto.

Aunque en los siguientes capítulos se concretan los contenidos de cada clase, para poder entender las relaciones entre las clases es necesario conocer los contenidos básicos de cada clase.

La clase `IOfile` contiene en memoria la imagen que se lee y guarda en ficheros, así como el código necesario para poder deshacer cambios y para convertir el tipo de color (por ejemplo, para representar una imagen en color como tonos de gris). También incluye la declaración de las funciones necesarias para el manejo de errores y el interfaz con el usuario. Más adelante se verá que la clase `IOfile` sólo define las

funciones del interfaz de usuario y de errores porque el código que realmente se ejecuta al llamar a esas funciones es el contenido en las clases derivadas de `IOfile`.

Los algoritmos para el tratamiento básico de la imagen están contenidos en la clase `ImgBasic`. Aunque los algoritmos se podrían haber dispuesto en una biblioteca estilo "C", se ha preferido utilizar una clase derivada de `IOfile` para utilizar la gestión de errores y la interfaz de usuario que proporciona `IOfile`. La clase `ImgBasic` se deriva virtualmente de la clase `IOfile` ya que de esta forma se asegura que sólo existe un objeto `IOfile` cuando se utilice la herencia múltiple (la justificación del tipo de relación entre `ImgBasic` e `IOfile` que se ha escogido se puede obtener en el tercer apartado del capítulo sexto).

En la clase `Consola`, derivada de `ImgBasic`, se implementan todas las funciones de tratamiento de errores (por ejemplo, para que den un mensaje, cierren la aplicación, etc.) y las funciones de la interfaz con el usuario. En esta clase está contenido el código que se ejecuta cuando `IOfile` o `ImgBasic` llaman, en la versión en modo texto del entorno, a alguna función de la interfaz de usuario o de gestión de errores.

Una característica que se ha utilizado en la clase `IOfile` es que contiene funciones virtuales puras[1], que están declaradas inicialmente en estas clases, pero que no se le asocia un código hasta que no se sobrecargan en alguna clase derivada tal como `Consola`. Por ejemplo, la clase `ImgBasic` llama a la función *Error*, que está declarada en la clase `IOfile` para poder ser utilizada, pero que no se codifica hasta que la clase `Consola` proporciona el código que se ejecuta cada vez que se produce un error. De esta forma, es sencillo tener un núcleo con las funciones básicas formado por las clases `IOfile` e `ImgBasic`, y una capa que sirva de intercomunicación entre el núcleo y el usuario, que en el caso de utilizar un entorno en modo texto es `Consola`.

El uso de clases facilita la reutilización de código, permite la clara separación de las capas del entorno y las responsabilidades de cada clase. Otra ventaja de la estructura del entorno nada desdeñable es que de esta forma se acelera el proceso de compilación.

[1] Una función virtual pura está declarada o definida en la clase pero no tiene ningún código asociado. Si se hace una analogía en lenguaje “C”, sería una función de tipo *`extern`*, que puede ser utilizada dentro del programa porque está declarada, y que en algún momento del enlazamiento ha de ser asociada a un

Dado que la estructura interna de la imagen y las funciones de E/S contenidas en `IOfile` son definitivas y no van a cambiar, la clase `IOfile` se compila formando la librería "*IOfile.lib*". Esta librería, formada a partir de más de un megabyte de código fuente, no debe ser recompilada aunque cambien las clases derivadas de `IOfile`.

A su vez, los algoritmos contenidos en la clase `ImgBasic` se compilan y se genera la librería "*Image.lib*", acelerándose todo el proceso de construcción.

Por último, la clase `Consola` añade la codificación de las funciones que servirán como interfaz de comunicación del núcleo `ImgBasic` con el usuario. La clase `Consola`, se enlaza (aunque quizás sería más claro utilizar el término anglicista "linkar") con las librerías "*Image.lib*" para dar el fichero ejecutable "*Visio.exe*". De esta forma, se produce algo parecido al enlazamiento incremental, acelerando los tiempos de construcción de la aplicación.

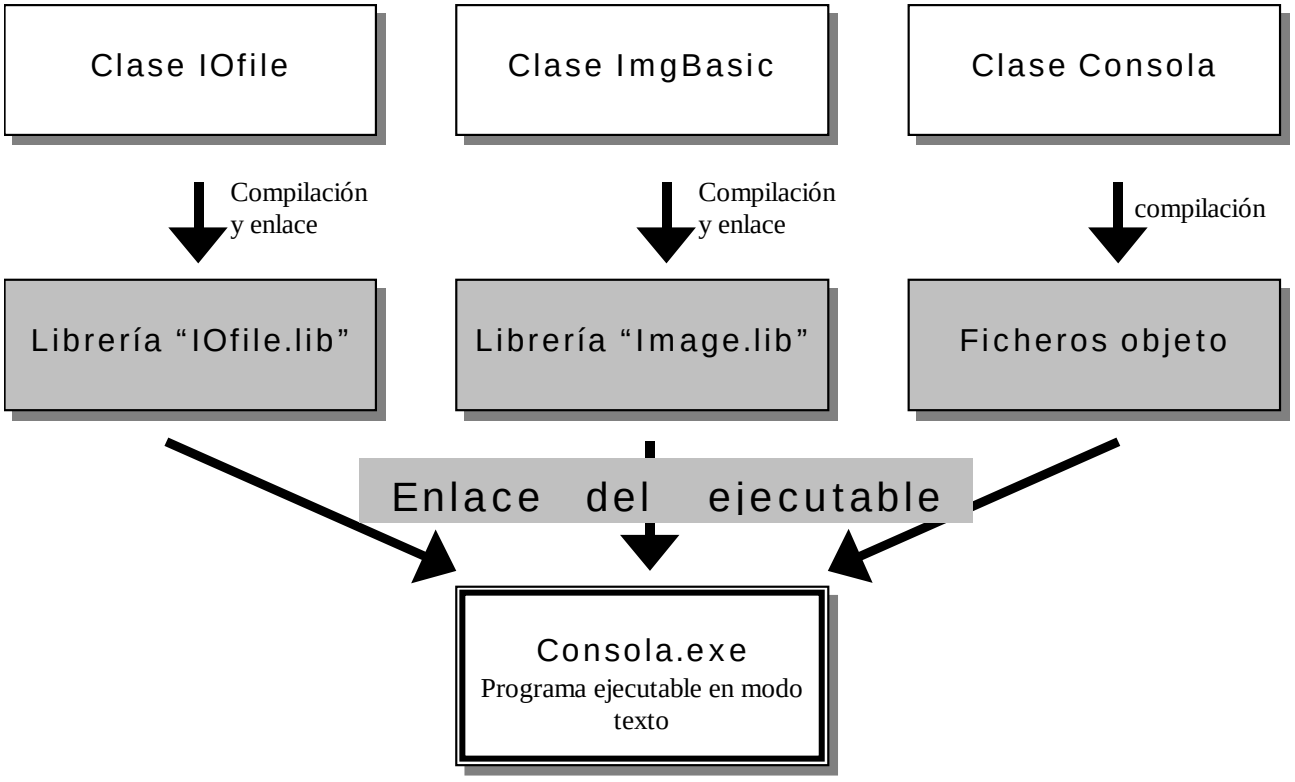


Fig. 2: Esquema del proceso de construcción de la aplicación en modo texto.

código ejecutable. Análogamente, las funciones virtuales puras se pueden utilizar en las clases base, pero el código que se les asocia pertenece a alguna clase derivada.

4.4 Jerarquía de objetos utilizada en el entorno de visión en Microsoft Windows

En el caso de que utilicemos la interfaz gráfica Microsoft Windows, la estructura de la imagen en memoria y los algoritmos no cambian. Las aplicaciones en entorno MS-Windows son fáciles de usar y tienen una rica interfaz gráfica, compuesto por más de 600 funciones. La facilidad de uso para un usuario novel tiene como contrapartida un trabajo extra para el programador, que debe respetar muchos detalles para que la aplicación se comporte apropiadamente. Por tanto, la capa gráfica es mucho más compleja que la utilizada en el caso de la aplicación en modo texto.

Para facilitar la labor del programador Microsoft ha puesto a pública disposición las clases *Microsoft Fundation Classes (MFC).* Estas clases proporcionan un esqueleto que oculta (hasta cierto nivel) la complejidad inherente a la programación en cualquier entorno gráfico de ventanas, y proporcionan una garantía de compatibilidad con las futuras versiones del sistema operativo MS-Windows.

La forma de trabajar con las *MFC* es sobrecargar las funciones virtuales que efectúan ciertas operaciones y sustituir el comportamiento por defecto de las clases MFC. De esta forma no se tiene que programar las características comunes a la mayoría de las aplicaciones de *Windows* y sólo se sobrecargan la funciones que efectúan tareas específicas a la aplicación que se está diseñando.

Las clases *MFC* se basan en la arquitectura del *modelo-vista-controlador* (MVC). La arquitectura de *modelo-vista-controlador*, separa una aplicación en tres capas:

• El *modelo* se refiere a la capa de la aplicación donde todos los objetos dependientes de la aplicación residen.

• La *vista* es la capa de representación que muestra la información al usuario.

• El *controlador* es la capa de interacción, que hace posible la intercomunicación entre los dispositivos de entrada (como el teclado y el ratón) y la *vista* y el *modelo*.

Las clases *MFC* soportan la arquitectura *modelo-vista-controlador* a través del paradigma *documento-vista*, en donde el *controlador* y la *vista*

se funden en las clases derivadas de *CView* y el *documento* se representa a través de las clases derivadas de *CDocument*.

Dado que el presente Proyecto Fin de Carrera no tiene como objetivo explicar la estructura subyacente a las clases *MFC* ni la programación en el sistema operativo Windows, a partir de este punto se asumirá que el lector tiene un cierto conocimiento del tema. Para obtener más información sobre el tema puede ser útil la ayuda *on-line* del entorno de programación [VC+95], el manual de referencia del compilador [VC+97] y la bibliografía que acompaña al capítulo.

En el entorno de visión artifical, la clase *documento* básica se denomina `CDibDoc`[2] y tiene como padres a la clase *CDocument* y a la clase `IOfile`. La clase *CDocument* (que pertenece a las clases MFC) aporta la funcionalidad de un *documento* Windows y la clase `IOfile` proporciona las herramientas necesarias para leer y guardar imágenes. La clase `CImgBasicDoc` hereda de la clase `CDibDoc` la funcionalidad básica y de la clase `ImgBasic` algunos algoritmos de tratamiento de imagen. A su vez, la clase C`ImgBasic` y la clase `Inspection` se derivan para formar la clase `CInspectionDoc`, en donde se puede acceder a los algoritmos de inspección contenidos en la clase `Inspection` y se retiene la funcionalidad Windows que proporciona la clase base `CDibDoc`.

En nuestro entorno, las funciones de la clase *CDibView* (y las clases que se derivan a partir de ella) realizan la presentación en pantalla y actualización de las imágenes contenidas en memoria. Para formar la clase *CDibView*, se añaden a la clase *CView* (que pertenece a las clases MFC) las funciones que muestran la imagen en pantalla y actualizan la paleta de colores del sistema. Si bien es cierto que la mayoría de las misiones encomendadas a la *vista* las realiza la clase *CDibView*, en el entorno MS-Windows no existe una única clase que se encargue de esta tarea en exclusiva. Por

[2] En el resto del capítulo se van a nombrar muchas clases. Para facilitar en alguna medida la comprensión de la estructura, se ha utilizado un tipo distinto de letra para las clases que son propias del entorno de visión artifical (como "`CDibDoc`" o "`IOfile`") y las clases propias de la biblioteca *Microsoft Fundation Clases* o *MFC*, como es el caso de "*CDocument*".

ejemplo la plantilla *CMultiDocTemplate* gestiona las relaciones entre la clase `modelo` y la clase `vista`, y la clase *CVisioApp* (derivada de la clase MFC *CWinApp*) se encarga de gestionar los mensajes del sistema, las barras de herramientas, la ayuda y contiene el equivalente de la función *main* de un programa "ANSI C".

Se ha derivado la clase CMDIChildFrame (incluida en MFC) para soportar múltiples `documentos`, cada uno con su barra de menús personalizada.

La relación entre las clases se puede expresar de forma más clara a través del correspondiente gráfico.

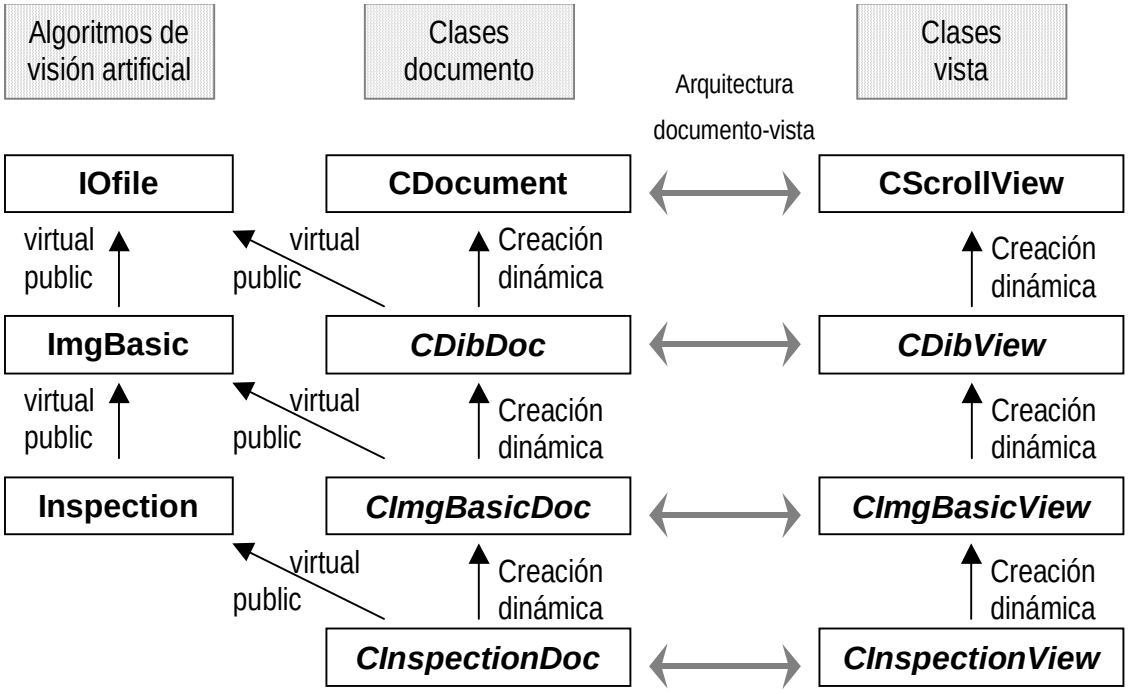


Fig. 3: Diagrama de clases `documento`–`vista` utilizado en el entorno de visión.

La clase `CDibDoc` tiene dos padres, es decir, utiliza la característica de C++ de herencia múltiple: contiene la clase `IOfile` y la clase *CDocument*. Dado que la arquitectura `documento-vista` se implementa en *MFC* a través de plantillas y se añade la información dinámica a través del método *CRuntimeClass::CreateObject*, es necesario crear las clases `documento` y `vista` dinámicamente. La creación dinámica de las clases se realiza de forma sencilla incluyendo la macro IMPLEMENT_SERIAL (*NombreClase*) en la declaración de la clase y la macro IMPLEMENT_DYNCREATE

(*NombreClase, NombreClaseBase*) . Para obtener más información sobre este tema, consúltese en [VC+95] y [VC+97] la creación dinámica de clases, la clase *CRuntimeClass*, las macros *DECLARE_DIYNCREATE* e *IMPLEMENT_DYNCREATE*, la clase *CObject* y cómo obtener información en tiempo de ejecución sobre la jerarquía de las clases *–Run Time Tipe Information o RTTI–*.

Las clases `IOfile`, `ImgBasic` e `Inspection` deben compartir una sóla imagen en memoria. Una forma de tener una sóla imagen en memoria sería haber utilizado la contención de clases. En vez de haber derivado las clases `ImgBasic` y `CDibDoc` de la clase `IOfile`, se habría utilizado en las clases `ImgBasic` y `CDibDoc` un puntero a un objeto `IOfile`. De la misma forma sería necesario incluir un puntero a un objeto `ImgBasic` en las clases `Inspection` y `CImgBasicDoc`. Los principales inconvenientes de utilizar esta técnica son las precauciónes con que se han de utilizar los punteros para no comprometer la estabilidad de la aplicación (la utilización de punteros *const* a objetos sólo es posible si se pueden fijar antes de iniciar el constructor de la clase) y dificultar la sobrecarga de funciones en las clases básicas (por ejemplo, la clase `IOfile` tal como se ha definido no se puede utilizar directamente sino que se han de sobrecargar las funciones virtuales puras). En la figura 4, se indica el esquema básico de clases que se hubiera tenido utilizar si se hubiera implementado la jerarquía utilizando la contención. Se puede observar que un objeto de la clase `CInspectionDoc` debe mantener la consistencia de cinco punteros cuando con tres punteros sería suficiente (puntero a `IOfile`, `ImgBasic` e `Inspection`).

El mecanismo de herencia virtual provee de un mecanismo más natural y efectivo: la clase `ImgBasic` se deriva virtualmente de la clase `IOfile` y dinámicamente de *CDocument*. Si no se hubiera utilizado la herencia virtual en la clases `ImgBasic` e `CDibDoc`, al crear un objeto `CImgBasicDoc` se tendría en memoria un objeto `IOfile` heredado a través de `ImgBasic` y otro heredado de `CDibDoc`.

Al utilizar la herencia virtual se asegura que sólo exista un objeto `IOfile` cuando se utilice la herencia múltiple en la clase `CImgBasicDoc`. A su vez la clase `Inspection` se ha de derivar virtualmente de `ImgBasic` para que al crearse la clase

`CInspectionDoc` no se tengan en memoria dos objetos `ImgBasic` (uno heredado a través de `CImgBasicDoc` y el otro a través de `Inspection`).

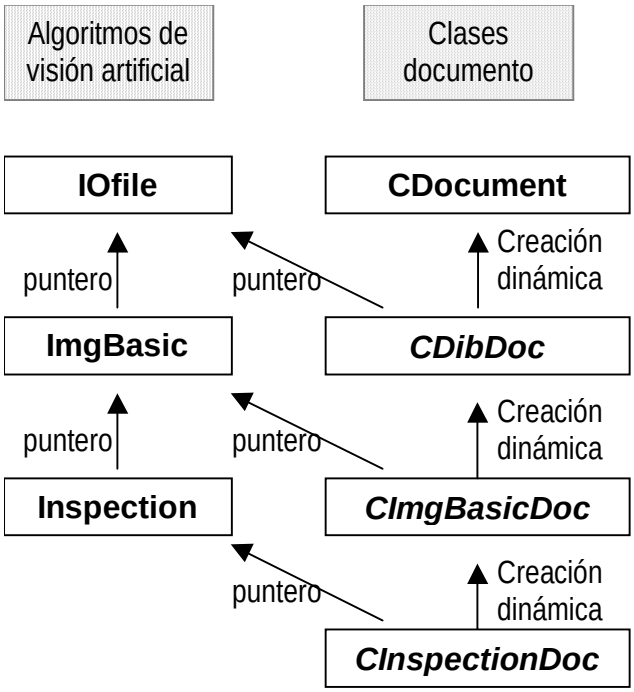


Fig. 4: Diagrama de clases *`documento`* en caso de haber utilizado la técnica de contención.

La aplicación elegida para el entorno de visión es *`multidocumento`* (en terminología inglesa se utiliza el término *Multiple Document Interface* o *MDI*) ya que podemos tener varias imagenes en memoria, y a la vez podemos tener varias ventanas abiertas para cada imagen. Para cada *`documento`* (objetos `CDibDoc`, `CImgBasicDoc` y `CInspectionDoc`) tendremos una o varias *`vistas`* (objetos `CDibView`, `CImgBasicView` y `CInspectionView`, respectivamente).

El manejo de las ventanas se realiza a través de las clases `CMainFrame`, `CDibFrame`, `CImgFrame` y `CInspectionFrame`. `CMainFrame` está derivada de la clase *MFC CMDIFrameWnd*, que proporciona una ventana *MDI* principal. Las clases `CDibFrame`, `CImgFrame` y `CInspectionFrame` están derivadas de la clase `CChildFrame`, que a su vez se deriva de la clase *MFC CMDIChildWnd*.

La clase ventana `CMainFrame` da soporte, entre otros cometidos, a las barras de herramientas, de estado y de menú, al mensaje que se muestra mientras se está cargando la aplicación, a los botones de minimizar, maximizar y salir, etc. La clase `CChildFrame` da soporte genérico a las ventanas MDI hijas dentro de una ventana padre, y sus hijas `CDibFrame`, `CImgFrame` y `CInspectionFrame` están ajustadas para gestionar las ventanas hijas de forma específica al tipo de documento al que están asociadas. La clases `CDibFrame`, `CImgFrame` *y* `CInspectionFrame` se encargan, entre otras cosas, de mostrar la *`vista`* en una ventana y de las barras de desplazamiento de la ventana, de que se muestre el menú correspondiente a la clase de visualización, procesamiento de imágenes o inspección, etc. cuando se cambia de una ventana hija a otra.

Para cada tipo de *`documento`*, se necesita especificar un identificador de los recursos –menú, combinaciones de teclas atajo, iconos, extensiones y denominación del tipo de *`documento`*, etc.– asociados a dicho tipo de *`documento`*, una clase *`documento`*, una clase *`vista`* y una clase ventana *MDI* hija. Esto se realiza a través de la clase *CMultiDocTemplate* y la función *CWinApp::AddDocTemplate*, que añade a la aplicación un objeto *CDocTemplate* para manipular cada tipo de *`documento`* en la función *CVisioApp*.

El esquema de todas las clases de tipo ventana utilizadas en el entorno se muestran a continuación en la figura 5. En el diagrama no se ha incluido la clase `CSplash`, derivada directamente de *CWnd* y que muestra una imagen de mapa de bits cuando se inicializa la aplicación ya que no tiene apenas interés.

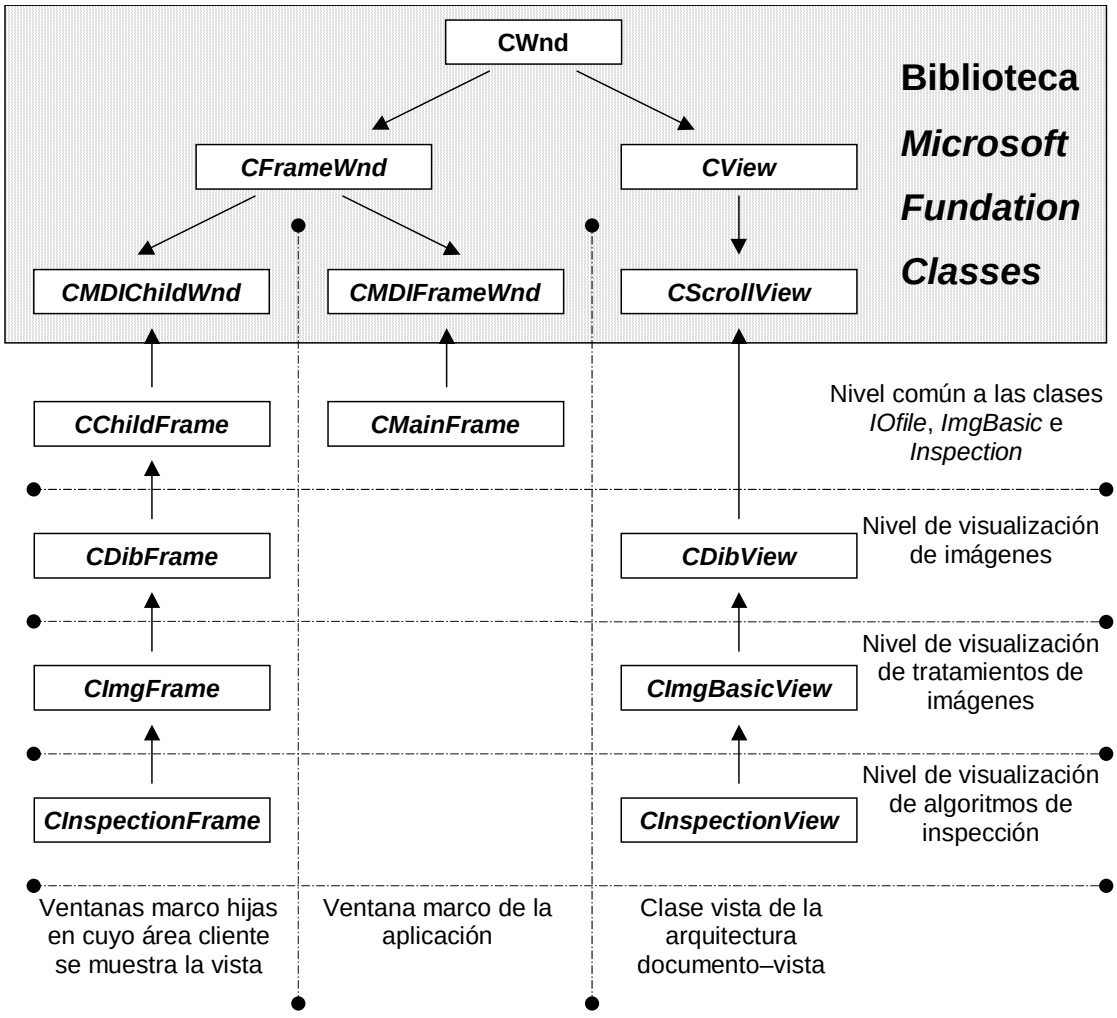


Fig. 5:Esquema de las clases *vista* utilizadas en el entorno.

Fig. 6: Derivación de la clase aplicación.

La clase aplicación `CVisioApp`, derivada de la clase MFC *CWinApp*, constituye lo que más se puede asimilar a una función `main` de un programa "ANSI C" o a la función `WinMain` de un programa realizado con la librería "Microsoft Windows SDK". En la figura 6 se puede contemplar el proceso de derivación de la clase aplicación personalizada, `CVisioApp`, a partir de la clase aplicación genérica *CWinApp*.

En el fichero "VisioApp.cpp" se define el único objeto de la clase aplicación que al construirse dará lugar a la aplicación. El proceso de inicialización puede resumirse el la figura 7.

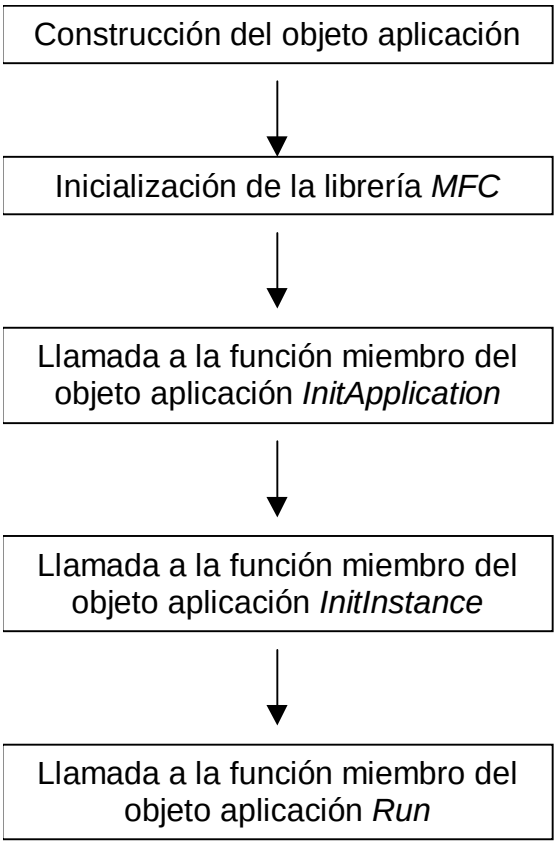


Fig. 7: Líneas principales de la inicialización de la clase aplicación.

El esquema típico de las relaciones entre el objeto aplicación y la plantilla `documento–vista` se muestran en la figura 8.

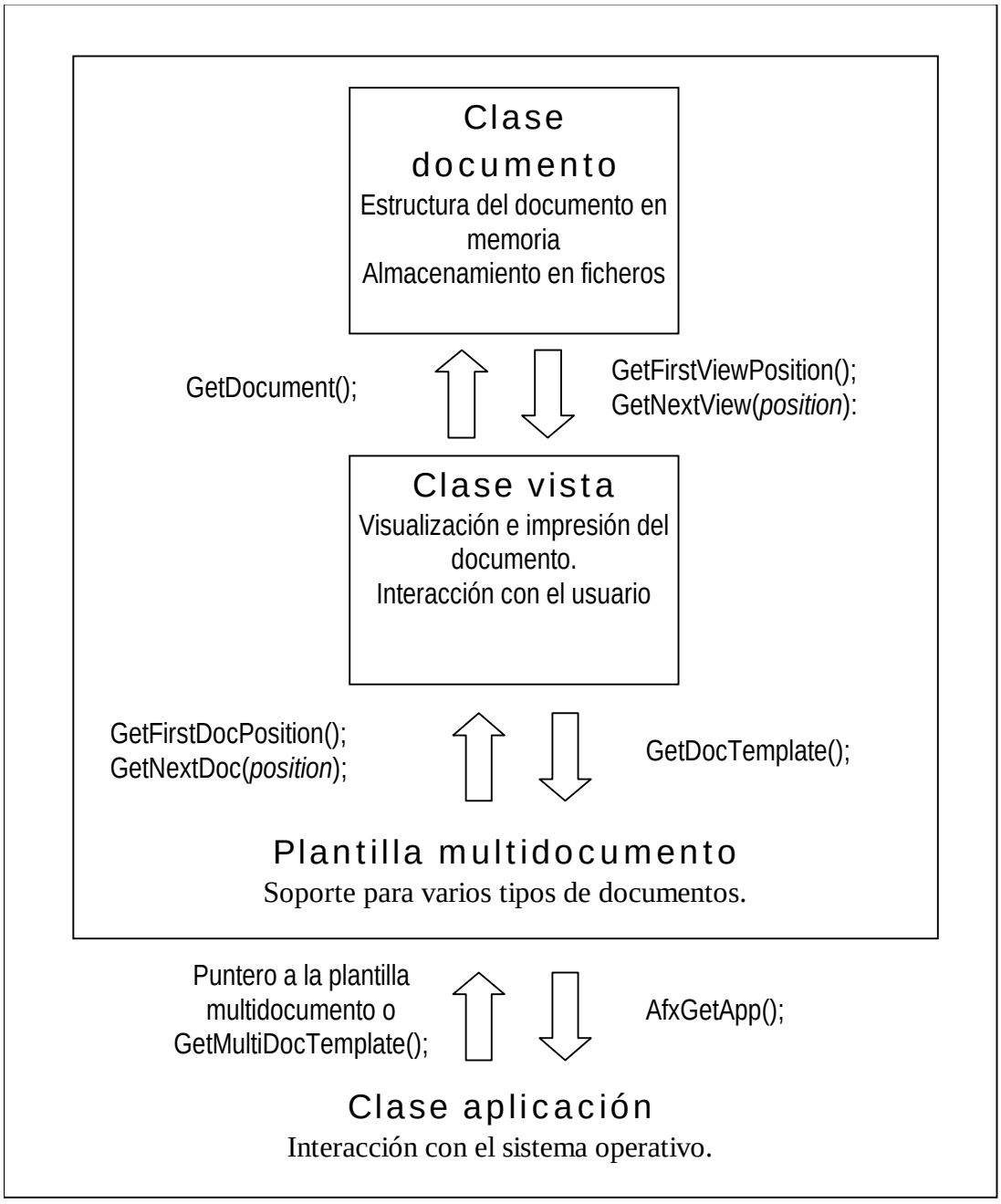


Fig. 8: Relaciones entre el objeto aplicación, el *`documento`* y la *`vista`*.

BIBLIOGRAFÍA:

[ATT91] AT&T and UNIX System Laboratories, Inc. "C++ Language System Release 3.0.2 Libray Manual" 1991.

[ATT92a] AT&T and UNIX System Laboratories, Inc. "C++ Language System Release 3.0.2 Machine–Readable Documentation Release Notes" 1991,1992.

[ATT92b] AT&T and UNIX System Laboratories, Inc. "C++ Language System Release 3.0.2 Release Notes" 1991,1992.

[ATT92c] AT&T and UNIX System Laboratories, Inc. "C++ Language System Release 3.0.2 Selected Readings" 1991,1992.

[BOO94] G. Booch. "Object-Oriented Analysis and Design with Aplications". The Benjamin/Cummings Series in Object–Oriented Software Engineering. The Benjamin/Cummings Publishing Company, Inc., Reading, Massachusetts, second edition, 1994.

[ELL90] M.A. Ellis and B. Stroustrup. "The Annotated C++ Reference Manual". Addison–Wesley, Reading, M.A., 1990.

[STR91] B. Stroustrup. "The C++ Programming Language Second Edition". Addison-Wesley, Reading, MA, 1991.

[USL92] USL. "C++ Language System Release 3.0.2 Internals Manual". 1991, 1992.

[UNL96] Viktor Toth. "Visual C++ 4 Unleashed". Sams Publishing, 1996.

[PAV96] David J. Kruglingnski. "Programación Avanzada con Visual C++". Microsoft Press, 1996.

[COM97] Scott Stanfield. "Cómo se hace con Visual C++ 4". INFORBOOK'S, 1997.

[VC+97] Microsoft Corp. "Microsoft Visual C++ 5.0 Reference Manual". Microsoft Press, 1997.

[GAM95] Microsoft Corporation. "Windows 95 Game SDK", Microsoft Press, 1995.

[VC+95] Microsoft Corp. "Visual C++ Books Online 4.0". Microsoft Press, 1995.

[MDG95] Nabajyoti Barkakati "Visual C++ 2 Master Developpers Guide". Second Edition. Sams Publishing, 1995.

[GLV97] Jens y Jörg Herbers "El Gran Libro de Visual C++". Ed. Marcombo, 1997.

[AVC97] Mark Adrews. "Aprenda Visual C++ ya". Ed. MacGraw-Hill, 1996.

[CEB95] Fco. Javier Ceballos Sierra. "Visual C++. Aplicaciones para Windows". Ed. RA-MA, 1995.

Capítulo 5:

La clase `IOfile`

5.1 Introducción

En este capítulo se comentará la clase `IOfile` ya que es la base de todo el entorno de visión. Todas las clases que contienen imágenes se derivan de ella porque contiene la estructura de la imagen en memoria, las rutinas para leer y escribir en ficheros, para gestionar la memoria y para interactuar con el usuario.

Si bien este capítulo es superfluo para un usuario final del programa, que sólo van utilizar el programa ya compilado, es imprescindible para los futuros desarrolladores que quieran ampliar los algoritmos contenidos en el entorno.

El capítulo está dividido en varios apartados. Primeramente, indicaré la finalidad de la clase y sus características generales, y a continuación presentaré el mapa de la clase. Una vez que se tenga una visión global de la clase, procederé a explicar en cierto detalle sus atributos y en menor detalle, sus funciones. Para que facilitar la comprensión del funcionamiento de la clase, he comentado las funciones agrupadas por su cometido: funciones para grabar y leer imágenes desde fichero, para gestionar la memoria, para cambiar el formato de las imágenes y funciones de la interfaz de usuario. En el anexo A se podrá obtener una información más detallada de las funciones.

5.2 Funcionalidad de la clase `IOfile`

La clase `IOfile` contiene la estructura de las imágenes en memoria, así como los procedimientos necesarios para leer o grabar una imagen en un fichero. Además, en la clase `IOfile` se definen las funciones virtuales puras necesarias para la interacción con el usuario (mostrar mensajes de error, preguntar parámetros al usuario...). Las funciones relacionadas con la interfaz se implementan en las clases derivadas de `IOfile` para poder mantener la independencia de la clase `IOfile` con el tipo de plataforma utilizada para generar el programa (entorno tipo terminal de texto o entorno de ventanas).

La clase `IOfile` contiene las rutinas necesarias para las operaciones de lectura/grabación de imágenes, así como la estructura en memoria y los procedimientos para la gestión de memoria. También proporciona las funciones que cambian propiedades de la imagen como el tipo de color (RGB, mapa de colores, imagen en tonos de gris, imagen monocroma...) y las dimensiones de la imagen.

Una de las razones que me llevó a estructurar el entorno de visión a partir de esta clase es poder tener una librería que contenga todas las funciones básicas para agilizar los procesos de compilación y enlazado. La librería que contiene la clase `IOfile`, de 1,8Mb de tamaño aproximadamente, no se ha de volver a compilar cada vez que se cambia la declaración de las clases derivadas de ella como `ImgBasic` o `Inspection`. Otra ventaja es que se agrupan y simplifican los procesos para la lectura y escritura en disco, proporcionando una única función sencilla para la lectura y otra para la escritura de imágenes en vez de tener varias y complicadas de utilizar para cada formato de fichero.

Los formatos para guardar imágenes en ficheros actualmente soportados en esta clase son MS-Windows BMP, FITS, Compuserve GIF, RGB IRIS, JPEG, Utah Raster Toolkit, SUN Rasterfile, RAW, TARGA, TIFF, ZSoft PCX, PDS/VICAR, PM, PBM, XBM y XWD. Además, las librerías `LIBTIFF` de Silicon Graphics y `JPEGLIB` del

grupo independiente JPEG se incluyen en la librería `IOfile.lib` ya que internamente se utilizan durante la lectura/escritura en los formatos TIFF y JPEG.

5.3 Estructura de la clase `IOfile`

A continuación se muestra un gráfico con todas las funciones y variables de la clase `IOfile`. Las entradas señaladas por ◆ indican una función pública, 🔒◆ indican una función privada que sólo se puede llamar por otras funciones de la clase `IOfile` y que sólo han de tenerse en cuenta si se quiere modificar la clase. Las funciones con la clave 🔑◆ son de tipo "*protected*", es decir, sólo pueden ser usadas dentro de su propia clase o sus hijas. Las variables de la clase se indican con la marca ◆.

RGBtoPIC24()
re_read()
reError()
SaveMemory(int status)
SaveMemory()
SavePic()
scan()
SetFullName()
SetISTR()
SetJPEGquality()
SetJPEGsmooth()
SetTIFF-compression()
ShowImage()
slow_map_pixels()
SortColormap()
splitfits()
start24bitAlg()
startGrayAlg()
StoreNewPic()
StoreNewPic(byte * NewPic
sunRasError()
TextView()
TIFFDialog()
UncompressFile()
WaitCursor()
Warning()
WriteEMP()
WriteITS()
WriteGIF()
WriteIRIS()
writeJRF()
writeJPEG()
WritePBM()
WritePM()
WriteRAW()
WriteSunRas()
WriteTarga()
WriteTIFF()
WriteXBM()
WriteXPM()
zbmError()
XpmLoadError()
zwdError()
zwdWarning()
YesNo()
YesNoCancel()
bMap
borg
conv24
curPage
DEBUG
filename
formatStr
gMap
gorg
istrs
mono
numcols
numPages
origHIGH
origMono
origNumcols
origPic
origPicType
origWIDE
pageBaseName
pHIGH
pic
picComments
picType
pWIDE
rMap
rorg
SaveMem

5.4 Datos de la clase `IOfile`

En el apartado anterior se exponía un gráfico de la estructura contenida en la clase `IOfile`. Para poder trabajar con la clase `IOfile` es necesario conocer la funcionalidad de sus elementos más importantes, y unos de los más importantes son las variables miembro. Las variables miembro de `IOfile` almacenan en memoria la imagen en sí más toda la información necesaria para determinar sus propiedades.

Es conveniente comprender la forma en que se almacena una imagen en la clase `IOfile`, según sea una imagen en color o monocroma. En la figura 1 se representa el esquema de los tipos de imágenes que se pueden contener en memoria sin ninguna transformación adicional. Si la imagen es monocroma, puede tratarse de una imagen bicolor (sólo contienen el color negro o el blanco) o en tonos de gris, con una gradación de 256 tonalidades desde el negro hasta el blanco. Una imagen en color puede estar almacenada en memoria en formato RGB, en el cual las componentes roja, verde y azul de la imagen se guardan independientemente, o utilizando una paleta[1] de colores de hasta 256 entradas.

La paleta de colores se basa en que el ojo humano sólo puede discernir entre un número limitado de colores. Este formato se ha extendido porque guardar para cada punto un índice que indique una entrada en la paleta de colores, ocupa la tercera parte de las componentes RGB (el índice de la entrada se almacena como un byte y cada componente RGB se almacena como un byte –en total tres bytes–). Sin embargo es muy inusual trabajar con paletas de colores en algoritmos de visión y únicamente se ha incluido este formato en el entorno de visión por su gran difusión.

[1] A continuación, me referiré indistintamente a las imágenes con paleta de colores o con tabla de colores. El término tabla de colores recuerda que internamente se utiliza una tabla para codificar los colores, mientras que el término paleta de colores alude intuitivamente a cómo un artista pinta un cuadro con un número reducido de pigmentos colocados en su paleta. Ambos términos son equivalentes y su uso depende de la bibliografía consultada.

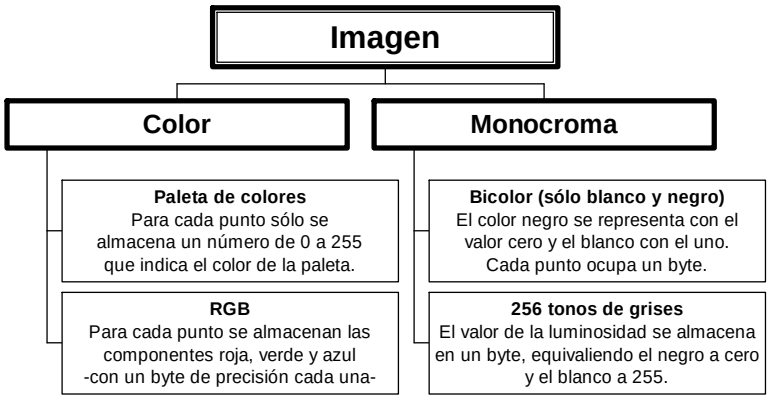


Fig. 5.1: Tipos de imágenes que se pueden almacenar en `IOfile`.

Cuando una imagen en color se representa mediante una paleta de colores, se calculan los colores que más aparecen en la imagen, agrupándose los colores indistinguibles al ojo humano en un solo tono. De esta forma se calculan los colores más representativos y se toman tantos como entradas distintas se quieran tener en la paleta. Finalmente, cada pixel cambia su color al color al que más se aproxima dentro de los contenidos en la paleta, produciéndose una pérdida de información en la imagen. La utilización de paletas de colores se ha habilitado en la clase `IOfile` porque hay muchos formatos de fichero que soportan esta característica, e incluso algunos tan populares como GIF sólo soportan imágenes de este tipo, y porque en muchas pantallas de ordenador se utilizan paletas de color para representar imágenes.

A continuación se puede observar la funcionalidad de todas las variables miembro de la clase `IOfile`, que en este caso son todas variables públicas. No he utilizado en la presente exposición el orden alfabético para hacer más comprensible la estructura de datos. En general, las variables están dispuestas por mayor a menor de importancia.

- El puntero ***`pic`*** a una zona de memoria que contiene la imagen almacenada por filas, de izquierda a derecha, empezando desde arriba. Cada punto está representado por un byte en el caso de imágenes monocromas, en tonos de gris o con paleta de colores y tres bytes consecutivos en el caso de imágenes en color RGB.

• Los números enteros ***pWIDE*** y ***pHIGH*** representan el ancho por el alto de la imagen en pixels.

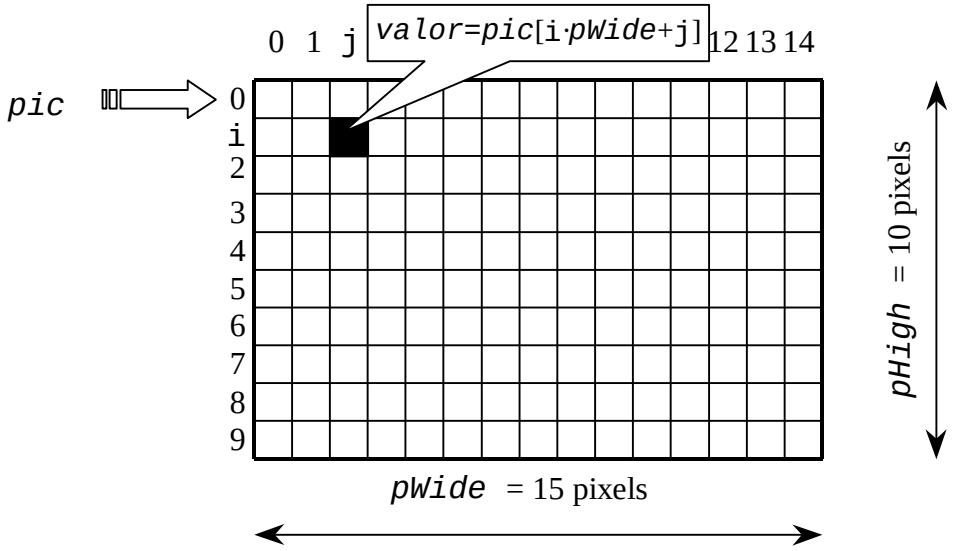


Fig. 5.2: Formato de una imagen en memoria en formato no-RGB.

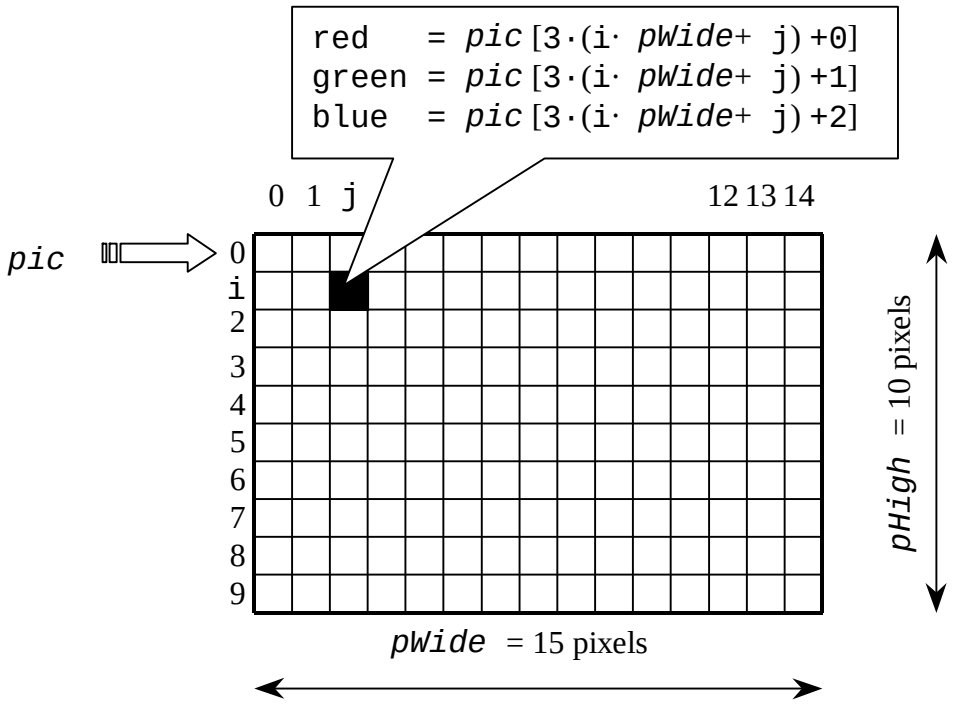


Fig. 5.3: Formato de una imagen en memoria en formato RGB.

• ***mono*** tiene el valor cero si la imagen contiene colores y distinto de cero si la imagen es monocroma bicolor –sólo blanco y negro– o de tonos de grises. Esta variable es útil para determinar si se ha de aplicar un algoritmo para imágenes en color o monocromas.

• El entero ***picType*** indica el número de bits por pixel de imagen apuntada por *pic*. Puede tomar el valor PIC8 y PIC24.

— Si su valor es PIC24, la imagen está representada por tres bytes por pixel correspondientes a la intensidad del color rojo, verde y azul –es decir, formato RGB–. Los bytes se almacenan en la región de memoria apuntado por *pic* en orden consecutivo: R1 G1 B1 R2 G2 B2 R3 G3 B3 R4 G4 B4... , donde R1, G1 y B1 representan las intensidades del color rojo, verde y azul del primer pixel, en el rango 0 ... 255.

— Si su valor es PIC8, la imagen está almacenada en memoria con un byte por pixel. El valor de ese byte (rango de 0 a *numcols*-1) indica o una tonalidad gris o el color correspondiente a una entrada de la tabla de colores. Hay hasta 256 colores simultáneos distintos, correspondientes a una gradación de grises o a las entradas en la tabla tal como se explicará a continuación.

• ***numcols*** contiene el número de colores usados del mapa de colores. Para una imagen del tipo monocroma bicolor –sólo blanco y negro– vale 2. Para una imagen en tonos de gris vale 256 (256 tonos de gris). Para una imagen en color con mapa de colores, puede tomar un valor entre 2 y 256, ya que en ese caso se pueden representar hasta 256 colores simultáneamente de una gama de 16.777.216 colores. Para una imagen en color tipo RGB, con una gama de 16.777.216 colores simultáneos, *numcols* vale 0 pues no se utiliza el mapa de colores –se almacena directamente las componentes rojas, verdes y azul para cada punto–.

• Las tablas ***rMap[]***, ***gMap[]*** y ***bMap[]***, de hasta 256 elementos o entradas, contienen el mapa de colores de las imágenes de tipo distinto al RGB. En *rMap* se almacena la componente roja para cada entrada, en *gMap* la componente verde y en *bMap* la componente azul. Cada componente se guarda en vectores independientes y el número de colores usados se almacena en *numcols* (sólo las entradas [0] hasta [*numcols*-1] de la tabla de colores son válidas).

En la tabla 1, se ha representado una tabla genérica para imágenes en color. Los colores contenidos en la tabla son arbitrarios para imágenes en color, es decir, pueden tomar cualquier valor y en cualquier orden.

Índice	rMap[]	gMap[]	bMap[]	Descripción
0	x_0	y_0	z_0	color 0 (por ej. azul claro)
1	x_1	y_1	z_1	color 1 (por ej. rojo intenso)
2	x_2	y_2	z_2	color 2 (otra tonalidad cualquiera)
...	...	...	...	...
numcols-2	$x_{numcols-2}$	$y_{numcols-2}$	$z_{numcols-2}$	color *numcols*-2
numcols-1	$x_{numcols-1}$	$y_{numcols-1}$	$z_{numcols-1}$	color *numcols*-1

Tabla 5.1: Paleta de *numcols* entradas, genérica, para imágenes en color.

Para facilitar y acelerar el tratamiento de imágenes monocromas, se han predefinido dos mapas de colores de colores, uno para imágenes bicolores y otro para tonos de gris:

— Si la imagen es de tonos de gris, entonces la imagen está almacenada con un byte por pixel, "*picType* = PIC8", "*mono* = TRUE" y "*numcols* = *256*". El valor cero representa el color negro, el valor 255 representa el color blanco y los valores intermedios se corresponden con una escala lineal de grises, tal como se muestra en la tabla 2. Por tanto el valor de cada punto de la imagen corresponde con su luminosidad.

Índice	rMap[]	gMap[]	bMap[]	Descripción
0	0	0	0	Negro
1	1	1	1	Gris muy oscuro (luminosidad 0.4 %)
2	2	2	2	Gris muy oscuro (luminosidad 0.8 %)
...	...	...	...	...
n	*n*	*n*	*n*	Gris (luminosidad $\frac{n}{255}\cdot 100\,\%$)

253	253	253	253	Gris muy claro (luminosidad 99.2%)
254	254	254	254	Gris muy claro (luminosidad 99.6%)
255	255	255	255	Blanco

Tabla 5.2: Paleta de colores predefinida para tratar imágenes de 256 tonos de gris.

— Si la imagen es monocroma bicolor, entonces la imagen está almacenada con un byte por pixel, "*picType* = PIC8", "*mono* = TRUE" y "*numcols* = 2". El valor cero representa el color negro y uno representa el color blanco de acuedo con criterio "*min is black*" adoptado. El resto de valores están indeterminados.

Índice	rMap[]	gMap[]	bMap[]	Descripción
0	0	0	0	Negro
1	255	255	255	Blanco

Tabla 5.3: Paleta de colores predefinida para imágenes bicolores (sólo blanco y negro).

• ***filename*** es una cadena de caracteres que contiene el nombre del fichero de la imagen. Inicialmente es una cadena vacía, pero al leer una imagen de disco a memoria se almacena en ella el nombre del fichero. Cada vez que se graba la imagen con un nombre distinto o se lee una nueva imagen de fichero, se actualiza *fullname*. Para acceder a esta variable fuera de la clase IOfile, se deben utilizar las funciones miembro *GetFullName* y *SetFullName*.

• ***picComments*** es un puntero a una cadena de caracteres que contiene información sobre la imagen. Cuando se carga una imagen de un fichero con un formato que permite insertar comentarios dentro de la imagen, estos comentarios se guardan en la zona de memoria apuntada por *picComents* para poder restituirlos cuando se vuelve a guardar la imagen (por ejemplo en los formatos GIF, PM, PBM, XPM, FITS).

• ***formatStr*** es una cadena corta en donde se almacena la dimensión de la imagen y el tipo de formato de fichero cada vez que se lee una imagen de fichero. Por ejemplo, "200x150 TIFF", "50x75 Sun Rasterfile", "100x25 PCX", etc. Su contenido es sólo informativo.

• ***istrs*** es un vector de cadenas que contienen información de la imagen. Para mostrar la información que contiene por pantalla, hay que llamar a la función *drawStrings*. Su contenido se obtiene mediante la función *GetISTR* y se con *SetISTR*. Por orden, las cadenas contienen información sobre: el último tratamiento aplicado a la imagen, avisos mandados por procedimientos, el nombre del fichero asociado, el formato del último fichero leído/guardado y la resolución. El resto de entradas no se utilizan actualmente.

• ***numPages*** y ***curPage*** contienen el número de imágenes en un fichero multi-página (tal como el GIF o TIFF), y el número de la página actual. Cuando se abre un fichero multi-página, cada página almacena temporalmente en un fichero cuyo nombre está formado por el prefijo contenido en la cadena ***pageBaseName*** y el número de la página en formato decimal. Finalmente se carga en memoria la primera página o imagen.

En este proyecto la lectura y escritura de ficheros multi-página o multi-imagen todavía no está implementada. De todas formas, se han previsto en la clase ya que puede ser una buena solución para tratar escenas captadas con varias cámaras (por ejemplo, en visión tridimensional).

♦ Además, para hacer posible deshacer cambios en una imagen cuando se aplica un algoritmo, es necesario guardar un duplicado de la imagen en las variables análogas a *pic*, *picType*, *pWIDE*, *pHIGH*, *rMap*, *gMap*, *bMap*, *mono* y *numcols* en las variables *origPic*, *origPicType*, *origWIDE*, *origHIGH*, *rorg*, *gorg*, *borg*, *origMono* y *origNumcols*, respectivamente. Estas variables son de uso interno y el usuario nunca las deberá modificar directamente[2].

[2] Estas variables sólo se modifican en las funciones *EndAlg24bitAlg*, *EndAlgGreyAlg* y *StoreNewPic* para guardar el original y *RestorePic* para volver a la imagen guardada

♦ Por último, he de señalar que existen unas variables miembro estáticas, cuyo valor es común a todas las imágenes. Por ejemplo, ***DEBUG*** indica si se ha de mostrar mensajes para la depuración. El algoritmo utilizado para reducir una imagen RGB a otra con mapa de colores se selecciona con la variable *conv24*. Si ha de ahorrarse memoria, *SaveMem* es distinta de cero. Los parámetros por defecto para guardan imágenes TIFF y JPEG se almacenan *TIFFcomp*, *JPEGquality* y *JPEGsmooth*. Estos valores pueden son ajustados por el usuario y tienen poca utilidad para el desarrollador.

• ***DEBUG*** : esta variable miembro estática se utiliza para indicar si se ha de mostrar información adicional en los algoritmos para facilitar su depurado. Por defecto, el valor DEBUG es cro. Para el primer nivel de depuración, debug debe valer 1, para el segundo 2, etc.

• ***conv24*** indica el tipo de algoritmo utilizado para la reducción de colores. Cuando se muestran imágenes en pantallas con paleta de color, o cuando se guardan imágenes en tipos de fichero que sólo admiten imágenes con paleta de colores (por ejemplo, el formato GIF), se ha de calcular la reducción de colores. La variable conv24 especifica el tipo de algoritmo que se utilizará en estos casos:

- `CONV24_BEST`: La mejor conversión es poco recomendable pues consume mucho tiempo de cálculo y sólo da unos resultados ligeramente superiores a `CONV24_SLOW`.
- `CONV24_SLOW`: Es el algoritmo que se utiliza por defecto y el que ofrece la mejor relación tiempo de cálculo/calidad final. En un ordenador 486 o superior, el consumo de tiempo es reducido y los resultados son muy buenos. Es el valor por defecto
- `CONV24_FAST`: Este algoritmo se ha diseñado para aumentar la velocidad de visualización en pantallas con 256 colores en ordenadores lentos como los 386.

anteriormente (en el anexo correspondiente a las funciones contenidas en la clase `IOfile` se puede obtener información detallada de estas funciones).

El incremento de velocidad se consigue a costa de reducir la calidad final de las imágenes.

5.5 Funciones públicas de la clase `IOfile`

En la clase `IOfile` existen bastantes funciones miembro de tipo privado o protegido. Estas funciones no tienen ningún interés para los usuarios de la clase `IOfile` y sólo pueden ser llamadas dentro de la misma clase.

Sin embargo, hay otras funciones declaradas como públicas que probablemente el desarrollador necesitará llamar. Por ello, las funciones públicas se han intentado diseñar para que su utilización sea lo más sencilla posible.

Las funciones públicas que se proporcionan se pueden dividir en funciones para lectura y escritura de imágenes en fichero, funciones para la gestión de memoria, funciones relacionadas con el formato interno y el color de las imágenes y funciones para interactuar con el usuario. En este capítulo se comentarán –sin entrar en detalles– las funciones más importantes, haciendo hincapié en la misión para la cual han sido diseñadas. La información detallada de todas las funciones públicas incluidas en la clase `IOfile`, se encuentra en el anexo A.

5.5.1 Lectura y escritura de una imagen en un fichero.

Normalmente no es necesario llamar directamente a las funciones que leen o graban imágenes en disco, ya que es el usuario el que indirectamente las llama a través de las opciones del menú Archivo —abrir, guardar, guardar como, etc.—. Además, cuando se efectúan modificaciones sobre la imagen, antes de cerrar la ventana el programa pide confirmación para desechar los cambios.

Para cargar una imagen desde un fichero a un objeto `IOfile`, se debe utilizar la función *`OpenPic`* y para guardar una imagen en un fichero, se debe utilizar la función *`SavePic`*. Al llamar a estas funciones se debe especificar si el nombre del fichero es el que actualmente contiene el objeto `IOfile` o se debe preguntar al usuario por un nuevo fichero.

Estas funciones detectan automáticamente el formato de las imágenes: al inicio de los ficheros existen unas claves que identifican de forma inequívoca la imagen, independientemente de la extensión del fichero. La excepción a la regla es el formato RAW, que no contiene esta clave y que por tanto se ha de identificar por la `raw`. La función *`SavePic`* graba la imagen en el formato indicado por la extensión del fichero y, en caso de no existir la extensión o no ser soportada, utiliza la extensión por defecto `bmp` —formato Windows BMP—.

Las extensiones que actualmente se soportan son BMP, FTS, FITS, GIF, JFIF, JPG, JPEG, PBM, PGM, PM, PNM, PPM, RAS, RGB, TGA, TIF, TIFF, XBM y XPM —aunque en alguno de los formatos más complejos no se soportan todos los subformatos introducidos más recientemente—.

5.5.2 Gestión de memoria

En este subapartado he incluido las funciones más usuales para gestionar las imágenes. Las imágenes ocupan cantidades muy importantes de memoria, por lo que es fundamental para la estabilidad del entorno controlar su ubicación y liberación. El presente entorno de visión habilita deshacer el último cambio de la imagen, lo cual supone una cierta complicación en la gestión de memoria. Afortunadamente para el futuro desarrollador, se han previsto las funciones adecuadas para que la gestión de la imagen se realice a alto nivel.

Además de la inicialización y liberación de recursos que se realizan en el constructor y destructor de la clase `IOfile`, existen una serie de funciones encaminadas a controlar los cambios de la imagen. Aunque es posible modificar las imágenes sin utilizar estas funciones, es más robusto y sencillo utilizar la función *`StoreNewPic`*[3]. Al utilizar esta función se tiene soporte automático para deshacer.

[3] *`StoreNewPic`* almacena la imagen en IOfile pero no actualiza la presentación en pantalla. Por ello, casi siempre después de almacenar la nueva imagen es necesario llamar a la función *`RedrawPic`* que actualiza la imagen mostrada en la pantalla.

En la llamada a *StoreNewPic* se ha de proporcionar al menos un puntero a la zona de memoria donde está almacenada la imagen y el nuevo formato de imagen —existen además otros parámetros opcionales para cambiar el tamaño de la imagen y el mapa de color—. Si el puntero de la nueva imagen es NULL, se asigna a IOfile una imagen vacía de dimensión 0x0 y se libera la imagen anterior. Para simplificar el caso habitual de almacenar el resultado de un algoritmo, se proporcionan las funciones *end24bitAlg* o *endGreyAlg*, como se verá en el apartado siguiente.

Para volver a la imagen anterior al último algoritmo, se puede utilizar la función *RestorePic*, que equivale a la opción del menú Edición\Deshacer cambios. Cuando se activa el ahorro de memoria, no se guarda la imagen anterior a la aplicación del algoritmo y por tanto el comando no tendrá efecto.

5.5.3 Acceso a la imagen

Tal como se ha visto en el apartado 5.4, la imagen se puede guardar en memoria en cuatro formatos, según sea una imagen en color —RGB y paleta de colores—, blanco y negro y tonos de grises. Sin embargo, a veces es necesario disponer la imagen en un formato determinado, independientemente del formato original. Por ejemplo, algunos algoritmos pueden no trabajar con el color o al revés, otros quizás sólo funcionen con imágenes en color.

En un caso típico, al inicio algoritmo se llama a la función *start24bitAlg* si se necesita obtener la imagen en el formato RGB o a *startGreyAlg* si se requiere la imagen en tonos de gris. Estas funciones además de devolver la imagen original en el formato adecuado, proporcionan una imagen vacía (inicializada a cero) de las mismas dimensiones que la original para guardar allí la imagen transformada. Al final del algoritmo, se llama a la función *end24bitAlg* o *endGreyAlg* para devolver la imagen original y actualizar la imagen contenida en IOfile con la que se ha calculado en el algoritmo.

El procedimiento recomendado para acceder a la imagen en los algoritmos es:

— Al inicio del algoritmo, llamar a las funciones *start24bitAlg* o *startGreyAlg* para obtener la imagen en color o escala de grises, respectivamente.

— Realizar el algoritmo.
— Una vez concluido el algoritmo, llamar a las funciones *end24bitAlg* o *endGreyAlg* para devolver la nueva imagen.

Si el formato modifica las dimensiones de la imagen o necesita tener mayor control sobre la imagen, debe utilizar la función *GetImage*. La función *GetImage* devuelve la imagen contenida en IOfile en el formato que especifican sus parámetros. Un posible esquema de un algoritmo que utilizase *GetImage* sería:

— Crear una variable para almacenar si se ha de liberar la imagen devuelta por *GetImage* después de utilizarla.
— Obtener la imagen original con *GetImage*, especificando en la llamada el formato deseado. Si el formato pedido es el utilizado internamente por IOfile se devuelve directamente el atributo *pic*. En otro caso, se devuelve una copia en el formato pedido, que se ha de liberar al final del proceso.
— Una vez comprobado que se ha devuelto la imagen no nula, ubicar la memoria necesaria para el algoritmo.
— Realizar el procesamiento.
— Una vez concluido el algoritmo, liberar con la función *free* la imagen devuelta por *GetImage*, si es necesario.
— Llamar la función *StoreNewPic* para almacenar la nueva imagen. En esta función se puede especificar completamente las dimensiones y el formato de la nueva imagen.
— Llamar a la función *RedrawPic* para que actualice la imagen mostrada en pantalla.

Cuando se trabaja con color, puede ser deseable separar las componentes RGB de la imagen en tres matrices independientes, teniendo una matriz[4] con la componentes roja, otra con la verde y otra con la azul. Un caso típico en el cual esta forma de trabajar puede ser ventajosa es cuando se quiere aplicar un algoritmo para imágenes en tonos de

[4] Estrictamente, no se puede utilizar matrices para acceder a los datos de la imagen, ya que su tamaño no se conoce en tiempo de compilación. En el programa, la matriz se almacena –por filas– como un vector y la posición del elemeto (*i,j*) se ha de calcular como *vector*[*i**anchura + *j*].

gris a cada componente. Las funciones *RedComponent*, *GreenComponent* y *BlueComponent* calculan cada una de las componentes de la imagen. Si no especifica el segundo parámetro de estas funciones, se toma por defecto la imagen contenida en IOfile.

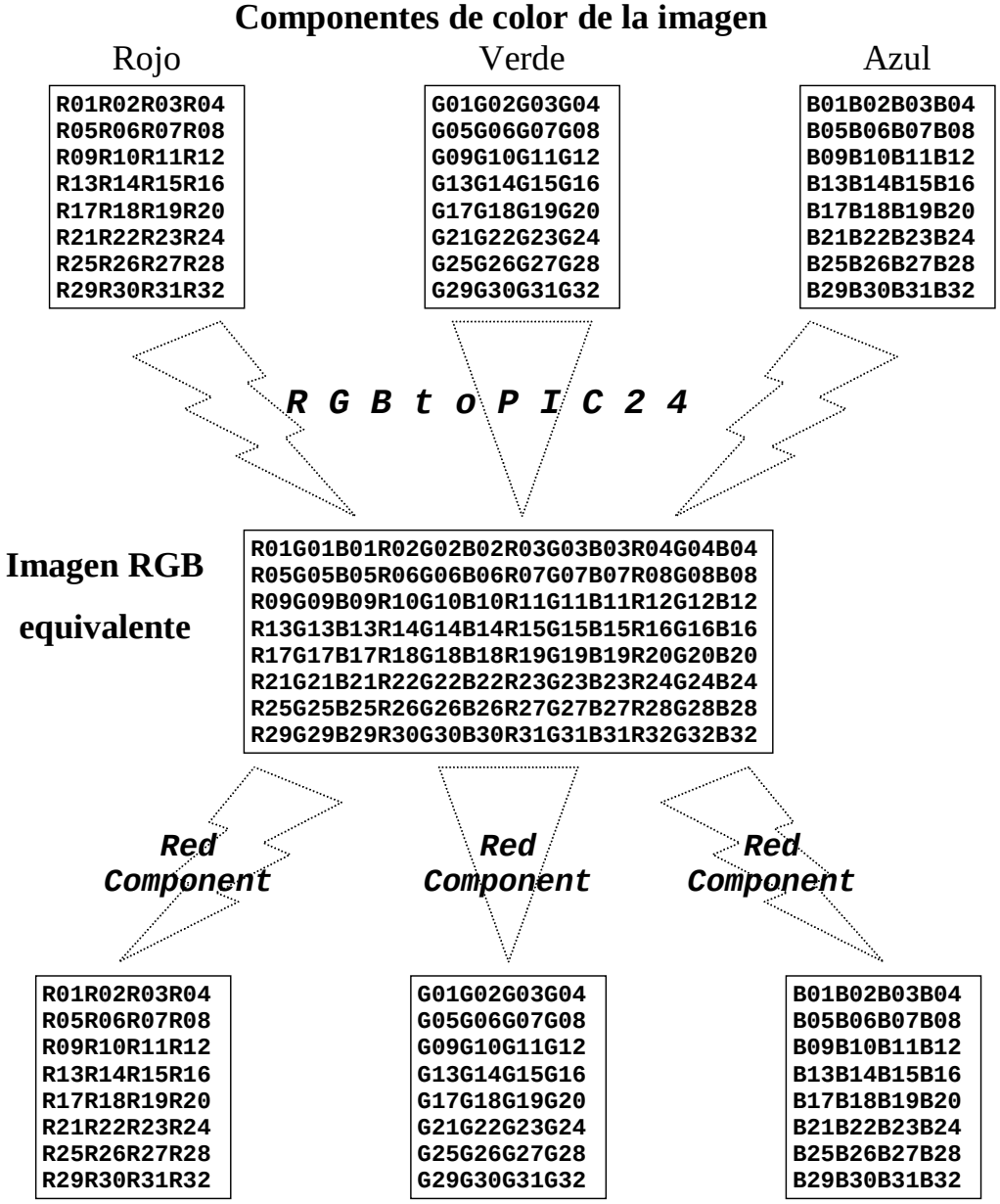


Fig. 4: Descomposición de una imagen RGB en sus componentes.

A partir de las componentes roja, verde y azul de una imagen, expresadas en tres matrices independientes, la función *RGBtoPIC24* guarda la imagen equivalente en el formato PIC24 o F_FULLCOLOR en la clase IOfile. Las funciones *RedComponent*, *GreenComponent*, *BlueComponent* y *RGBtoPIC24*

proporcionan una forma alternativa para tratar imágenes de color a *start24bitAlg* y *end24bitAlg*.

Otras funciones contenidas en la clase IOfile permiten forzar a utilizar un formato determinado dentro de la clase IOfile. La función *GetColorType* devuelve el tipo de imagen de los cuatro posibles contenido actualmente en IOfile: color RGB, color con paleta, 256 tonos de grises y blanco y negro. El tipo de formato utilizado se puede cambiar con la función *ChangeImageFormat*. Normalmente, esto no es necesario, ya que la propia clase utiliza el formato más adecuado dependiendo de las propiedades de la imagen. Cada vez que se lee una imagen desde fichero o se pega una imagen desde el portapapeles, IOfile determina el formato más adecuado para la imagen. En el apéndice A se encontrará información sobre estas funciones.

También se proporcionan las funciones *hsv2rgb* y *rgb2hsv* que convierten un punto expresado en el formato HSV (tono, saturación e intensidad) en el formato RGB (componentes roja, verde y azul del color) y viceversa. Por último, la macro *MONO* devuelve la luminosidad de un punto a partir de sus componentes RGB.

5.5.4 Funciones de la interfaz del usuario

En la clase IOfile se han incluido las funciones necesarias para la interacción del usuario con el programa. De esta forma se puede mantener la independencia de la clase IOfile respecto de la interfaz del usuario, ya que en las clases derivadas de IOfile se sobrecargarán las funciones de la interfaz según la plataforma para la cual se compilará el entorno de visión (terminal de texto, *MS-Windows*, *X-Windows* o *Motif*, etc.).

Debido a que la forma de interactuar con el usuario depende del tipo de plataforma escogido, el código de estas funciones no se puede concretar en la clase IOfile. La solución que se ha adoptado es crear unas funciones virtuales puras para interaccionar con el usuario. Puesto que estas funciones forman parte de la clase IOfile, se tiene acceso a ellas. Además, dado que son funciones virtuales puras no

tienen ningún código asociado en la clase `IOfile` pero sí se debe dar código al menos en una clase derivada de `IOfile` (por ejemplo, en las clases derivadas de `IOfile` para la plataforma de texto o *MS-Windows*). Al ser funciones virtuales, el código que ejecutan dichas funciones es el correspondiente a la última clase derivada que ha sobrecargado la función. Hay una serie de funciones que sólo tienen sentido en un entorno de ventanas, tales como las que muestran una imagen por pantalla, cambian el cursor, etc. Estas funciones, cuando se utilizan en un entorno basado en carácter, no producen ningún efecto –a lo sumo, mostrar un mensaje indicando que se ha llamado a esa función–.

Un ejemplo de esto es la función *`RedrawPic`* que realiza las operaciones necesarias para que la imagen mostrada en pantalla se actualice según el contenido de `IOfile` (en la versión modo texto, dada la imposibilidad de mostrar imágenes, sólo presenta un mensaje por pantalla para indicar que la imagen ha cambiado). Esta función siempre es llamada, directa o indirectamente, después de cualquier algoritmo, después de leer una imagen, etc.

También se ha diseñado la función *`ShowImage`* que muestra una imagen temporal por pantalla, sin afectar a las imágenes contenidas en la clase `IOfile`. Esto es útil para ver el proceso en algunos algoritmos paso a paso y ayuda a depurar los algoritmos. Si se especifica la cadena *message*, se muestra un cuadro de diálogo con ese texto y hasta que no se aprieta el botón aceptar no se restablece la imagen.

Otro grupo de funciones muy importantes son las que presentan texto. La función *`TextView`* muestra un fichero de texto por la pantalla. Las funciones *`Warning`*, *`Error`* y *`FatalError`* muestran por pantalla una cadena de caracteres, en un cuadro de diálogo con un botón para cerrar el mensaje. La función *`OpenAlert`* realiza el mismo cometido salvo que no tiene botón para cerrar (se debe cerrar la ventana del diálogo con la función *`CloseAlert`*).

Algunas veces es muy deseable poder utilizar *`printf`* para mostrar información de un proceso, pero no se quiere que el usuario tenga que aceptar la información como ocurre en los cuadros de diálogo. El usuario debe tener la opción de

analizar más tarde los mensajes mostrados. Esto se realiza típicamente en entornos UNIX desviando el flujo <stdout>. Sin embargo, esto no funciona correctamente en MS-Windows. Por ello se ha incluido en la clase `IOfile` las funciones *`printf`* y *`scanf`*, que realizan una salida y entrada en la ventana de texto de la aplicación.

Las funciones que se han habilitado para que el usuario introduzca datos al programa son *`scanf`* y *`InputDialog`*. Mientras que *`scanf`* hace activa la ventana de texto, de forma convencional, *`InputDialog`* muestra un cuadro de diálogo con un control de edición en donde se introduce el dato pedido. Otra diferencia es que *`scanf`* admite los parámetros de formato, mientras *`InputDialog`* devuelve la cadena introducida en el control de edición, de donde se han de leer los valores.

A veces simplemente se requiere del usuario que confirme o cancele un mensaje —típicamente mediante un cuadro de mensaje con dos o tres botones—. Para este tipo de mensajes se han dispuesto las funciones *`YesNo`*, *`YesNoCancel`*, *`OkCancel`*, *`RetryCancel`* y *`AbortRetryIgnore`*, que devuelven el valor del botón pulsado.

Por último, las funciones *`WaitCursor`* y *`ArrowCursor`* permiten cambiar el estilo del puntero del ratón. *`ProgressMeter`* muestra una barra de progreso en la base de la ventana en procesos que duren más de cuatro segundos.

BIBLIOGRAFÍA

[MON89] Montano L., Neira J. Sagüés C., Silva M., Tardós J.D. "Introducción a la robótica Industrial". C.P.S.U.Z, 1989.

[MAR82] D. Marr. "Vision". W.H. Freeman, San Francisco, 1982.

[ROB88] K.S. Fu, R.C. González, C.S.G. Lee. "Robótica: Control, detección, visión e inteligencia". Mc. Graw Hill, 1988.

[BRA82] Brady M. "Computational Approaches to Image Understanding". ACM Computing Surveys, Vl 14, No. 1, pp 3-71, March 1982.

[DUD73] Duda R.O., Hart P.E. "Pattern Classification and Scene Analysis". Wiley, New York, 1973.

[FU82] Fu K.S. "Sintactic Pattern Recognition and Applications". Prentice Hall, Englewood Cliffs, N.J., 1982.

[GON77] Gonzalez R.C., Wintz P. "Digital Image Processing". Addison-Wesley, Reading, Mass. 1977.

[HOR86] Horn B.K.P. "Robot Vision". The MIT Press, Cambridge, Mass., 1986.

[PAV77] Pavlidis T. "Structural Pattern Recognition. Srpinger, Berlin, 1977.

[SHI85] Shimon Y.N. (ed.) "Handbook of Industrial Robotics". John Wiley & Sons Inc., New York, 1985.

Capítulo 6:

La clase `ImgBasic`

6.1 Introducción

En el capítulo quinto se comentó la clase `IOfile`, que contenía la estructura en memoria de la imagen. Pero si se quiere diseñar un entorno de visión artificial productivo, no es suficiente con poder leer, escribir, procesar y mostrar imágenes: es necesario tener una biblioteca de funciones que permitan realizar todos los procesamientos más habituales sin tener que programarlos.

El objetivo principal del proyecto era construir el entorno, tal como se recoge en la clase `IOfile`, y futuros proyectos completarían hasta crear una amplia biblioteca de funciones para visión artificial. La construcción de la librería ya se ha iniciado en este proyecto por dos motivos: proporcionar algunos algoritmos de tratamiento de imágenes para ser utilizados por los usuarios y servir de ejemplo y de base para aquellas personas que desarrollen el entorno de visión. En esta línea se ha encaminado los proyectos fin de carrera de Dña. Begoña Zamora Catevilla y Dña. Gemma Benito Navaridas, que están realizando una labor muy notable ampliando el entorno con algoritmos de inspección industrial.

La librería se ha dispuesto en la clase `ImgBasic`, derivada de `IOfile`. Para facilitar una visión global de la clase `ImgBasic`, el manual de referencia se ha dispuesto en el anexo B. También se han añadido en el anexo C algunos ejemplos de utilización de la clase.

6.2 Funcionalidad de la clase `ImgBasic`

La clase `ImgBasic` añade a la clase `IOfile` algoritmos básicos de procesamiento de imágenes. La clase `ImgBasic` se ha derivado virtualmente de `IOfile` para poder utilizar las imágenes cargadas en memoria y los procedimientos de la interfaz de usuario.

Los algoritmos se pueden utilizar de forma que el usuario introduzca los parámetros del mismo o llamar directamente a la función que realiza el algoritmo, especificando los parámetros en la llamada de la función.

Los algoritmos incluidos en `ImgBasic` son:

- Funciones para recortar imágenes y redimensionarlas.
- Manipulaciones básicas, como invertir las imágenes, voltearlas, girarlas 90º y calcular la imagen espejo.
- Ajustes del color, tono, saturación, luminosidad, etc.
- Filtros de suavizado con máscara gausiana y máscara unidad.
- Filtro de la mediana.
- Filtro para aumentar los contrastes de la imagen.
- Algoritmos para añadir ruido a la imagen (ruido gausiano y desplazamiento aleatorio de pixels).
- Algoritmos para la detección de contornos: cálculo del gradiente de una imagen utilizando ventanas Sobel y Roberts, el algoritmo Canny y varios algoritmos multirresolución para la detección de contornos[1].
- Efectos sobre las imágenes, como el efecto bajorrelieve, explosión, rotaciones de un ángulo cualquiera, efecto cuadrícula y pintura al óleo.

[1] Realmente los algoritmos multirresolución utilizan técnicas avanzadas, pero se han incluido en la clase `ImgBasic` porque presentan una gran mejora frente al resto de los algoritmos de extracción de contornos.

6.3 Jerarquía de la clase `ImgBasic`

En el capítulo 4 ya se comentó la ventaja que presentaba la jerarquía de clases adoptada en el entorno. `IOfile` es la clase base virtual `ImgBasic` para evitar duplicidad de datos al utilizar herencia múltiple. En caso de que se cree una clase con herencia múltiple de varias clases que se hayan derivado virtualmente de `IOfile`, sólo se tiene una copia de la imagen en memoria que será compartida por todas las clases, al contrario que sucedería con una derivación no virtual.

Se han probado otros tipos de relaciones de la clase `ImgBasic` con `IOfile`, como es la herencia no virtual, pero con la desventaja de que si se utiliza herencia múltiple de varias clases derivadas convencionalmente de `IOfile`, se tendrán varias copias de la imagen en memoria. Otra posibilidad probada fue la utilización de un puntero a una clase `IOfile` (la opción de que la clase `ImgBasic` contuviera un objeto `IOfile` tiene todavía más desventajas que la herencia no virtual de `IOfile`, por lo cual tenemos que utilizar un puntero a la clase si no queremos duplicar la imagen en memoria). Si bien la utilización de un puntero a la clase `IOfile` resolvía el problema, complicaba la clase encargada de la interfaz de usuario. Por tanto, al final se decidió utilizar la herencia virtual, que mantiene una simplicidad elevada junto con la característica de mantener una única copia de la imagen en memoria si se crean clases con herencia múltiple.

6.4 Estructura de la clase `ImgBasic`

A continuación se muestra un gráfico con todas las funciones y variables de la clase `ImgBasic`. Las entradas señaladas por indican una función pública que está diseñada para ser llamada desde otras clases; indican una función privada que sólo se puede ser llamada por otras funciones dentro de la clase `ImgBasic` y que sólo ha de tenerse en cuenta si se quiere modificar la clase. Las funciones con la clave son de tipo protegido y son rutinas específicas de filtros o algoritmos, que a su vez son llamadas por las funciones públicas. La clase no tiene ninguna variable miembro, sólo procedimientos.

6.5 Estructura general de la clase

En este apartado se quiere dar una visión global de la clase. Si quiere obtener las especificaciones y los parámetros de los algoritmos, éstos se encuentran en el anexo B.

La clase `IOfile` contiene algoritmos y manipulaciones básicas. Las manipulaciones básicas sirven para recortar, invertir, y girar 90º las imágenes y calcular la imagen especular. Los algoritmos son, en general, más complejos y se pueden llamar de forma simplificada a través de una función genérica llamada *`DoAlg`*. Los algoritmos que se han incluido en la clase son aumentar o reducir las imágenes (zoom), filtros de suavizado, de la mediana, un filtro para aumentar los contrastes de la imagen, algoritmos para añadir ruido a la imagen (ruido gausiano y desplazamiento aleatorio de pixels), algoritmos para la detección de contornos y varios efectos fotográficos.

Tanto los algoritmos como las manipulaciones se pueden llamar sin especificar ningún parámetro, con lo que la imagen se toma de la clase padre `IOfile` y los parámetros los introduce el usuario, o se puede llamar especificando la imagen y los parámetros[2].

Dentro de las manipulaciones, se puede utilizar la función *`CropPic`* para recortar una imagen. Si se quiere redimensionar la imagen con escalado, es decir, reduciéndola o ampliándola sin recortarla, se puede utilizar *`ResizePic`*. *`InvertPic`* invierte una imagen, es decir, el color blanco lo convierte en negro y el negro en blanco, como si fuera un negativo de fotografía. Por último *`RotatePic`* gira la imagen 90 grados y *`FlipPic`* voltea la imagen.

En este apartado no voy a explicar detalladamente los algoritmos, sino que comentaré la función `DoAlg,` que se ideó para poder llamar de forma sencilla a los

[2] En algunas manipulaciones, se ha optado por proporcionar dos funciones con el mismo nombre y distintos argumentos. En los algoritmos se ha preferido utilizar nombres distintos para separar la función que interactúa con el usuario con la que realiza el algoritmo o procesado de la imagen.

algoritmos con una sola función —en el anexo B se encontrará información más detallada sobre todas las funciones contenidas en la clase `ImgBasic`—.

La función `DoAlg` sólo tiene como parámetro un identificador con el cual se selecciona el algoritmo que se va a realizar. En realidad esta función actúa como pasarela, ya que solamente llama a otra función según su parámetro. A continuación se enumeran los identificadores que tiene asociados cada identificador, con una descripción breve:

–"ALG_BLUR": Realiza un suavizado de la imagen pasando una máscara unidad, es decir, todos los pixels dentro de la ventana se ponderan de igual forma. El valor del pixel filtrado es la media de los pixels que forman la ventana. Equivale a la función *Blur*().

–"ALG_BLEND": Realiza el efecto explosión sobre la imagen. Equivale a la función *Blend*().

–"ALG_CANNY": Realiza el algoritmo de extracción de contornos de Canny. Equivale a la función *AlgCanny*().

–"ALG_CANYMRES": Aplica un análisis multirresolución para la detección de contornos tal como se indica en el capítulo siete de la memoria. Equivale a la función *CannyMRes*().

–"ALG_CROP": . Toma una porción de la imagen original como nueva función, es decir, crea una subimagen de la original recortando las dimensiones. Equivale a la función *CropPic*().

–"ALG_EDGE": Calcula el gradiente de una imagen utilizando el filtro Sobel. Equivale a la función *SobelGradient*().

–"ALG_FASTMRES": Este algoritmo se diferencia del anterior en el uso de filtros iterativos y optimizaciones para reducir el tiempo de proceso. Equivale a la función *Multires*(ALG_FASTMRES).

–"ALG_GAMMA": Corrige el valor de una imagen (por ejemplo el factor gamma) a partir de tablas de corrección de las componentes de color rojo, verde y azul, y/o tono, saturación y luminosidad. Equivale a la función

–"ALG_GRAD_ROB": Calcula el gradiente de la imagen según la fórmula de Roberts. Equivale a la función *GradientRoberts*().

–"ALG_GSBLUR": Efectúa un filtrado gausiano de la imagen. Se diferencia de "ALG_BLUR" en que la ventana de filtrado tiene los valores correspondientes a la distribución gausiana, en vez de ser una máscara con todos sus componentes la unidad Equivale a la función *GaussianBlur*().

–"ALG_GSNOISE": Superpone un ruido gausiano a la imagen. Equivale a la función *GaussianNoise*().

–"ALG_MEDIAN": Aplica el filtro de la mediana a la imagen contenida en memoria. Equivale a la función *MedianFilter*().

–"ALG_MULTIRES": Efectúa una extracción de contornos y caracterización de los mismos basada en técnicas multirresolución según [MAM95] y [JRB94][3].

–"ALG_NONE": Deshace el último algoritmo realizado a la imagen. Equivale a la función *RestorePic*().

–"ALG_OIL": Da el efecto de pintura al óleo a las imágenes. Equivale a la función *OilPaint*().

–"ALG_PIXEL": Crea el efecto cuadrícula de la imagen, es decir, agrupa los pixels en celdas rectangulares y da un color promedio a cada celda de la imagen. Equivale a la función *Pixelize*().

–"ALG_RESIZE": Escala la imagen, sin recortar la imagen. Equivale a la función *ResizePic*().

–"ALG_ROTATE": Realiza una rotación de la imagen sobre la original como fondo. Equivale a la función *FineRotate*(0).

–"ALG_ROTATECLR": Realiza una rotación de la imagen sobre un fondo negro. Equivale a la función *FineRotate*(1).

–"ALG_SATURATE": Esta función es un caso particular del ajuste de la luminosidad de una imagen. Multiplica la luminosidad de cada pixel por un factor, lo cual es apropiado para imágenes con poca iluminación, oscuras, como los resultados de los gradientes. Equivale a la función *Saturate*().

–"ALG_SHARPEN": Aumenta el contraste en los contornos, restando al valor de la luminosidad del pixel analizado la media de la luminosidad de los 9 pixeles de una

[3] En el capítulo 7 se comentarán los algoritmos multirresolución, y es allí donde se encuentran las referencias de los autores de este tipo de técnicas.

ventana 3x3 centrada en el punto. La media de la luminosidad se pondera con el valor de la luminosidad del punto analizado. Equivale a la función *Sharpen*().

–"ALG_SPREAD": Realiza un desplazamiento aleatorio sobre cada pixel de la imagen, a otro punto de la ventana considerada. Este efecto se puede considerar un tipo de ruido o distorsión de la imagen. Equivale a la función *Spread*().

–"ALG_TINF": Crea el efecto de bajorrelieve a partir de una imagen. Equivale a la función *TinFoil*().

BIBLIOGRAFÍA

[MAM95] Miguel Angel Martínez. Análisis Multirresolución por gausianas para la detección y caracterización de contornos en imágenes 2-D. Centro Politécnico Universidad de Zaragoza. Mayo de 1995.

[JRB94] J.R. Beltrán. Detección y clasificación de contornos, en imágenes digitalizadas, utilizando la transformada de Wavelet. Centro Politécnico Universidad de Zaragoza. Octubre 1994.

[CAN83] J.F. Canny. Finding edges and lines. Technical Report 720. M.I.T. 1983.

[CAN86] J.F. Canny. A computational approach to edge detection. IEEE Transations on Pattern Analysis and Machine Intelligence, 8:679-698, Noviembre 1988.

[XVJ94] John Bradley, 1989-94. "XV User manual". 1053 Floyd Terrace. Bryn Mawr, PA 19010

[IJG94] Thomas G. Lane. "IJG Group Library". Independent JPEG Group's software, 1994.

[TAR97] Varios autores. "TargetJunior User's manual" Robotics Research Group, Oxford University. Disponible en la dirección URL `robots.oxford.ac.uk.`

[WIR76] Niklaus Wirth. "Algoritmos + Estructuras de datos = Programas". Ediciones del Castillo.

[STR76] G. Strang. "Algebra lineal y sus aplicaciones". Ed. Addison-Wesley Iberoamericana.

[CAN76] George C. Canavos. "Probabilidad y Estadística. Aplicaciones y Métodos". Ed. McGraw-Hill.

[HOZ90] Holzmann. "Beyond Photography", capítulo 7.

[REC92] Press, William H., Saul A. Teukolsky, William T. Vettering, and Brian P. Flannery. "Numerical Recipes in C", Second Edition, Cambridge University Press, 1992.

Capítulo 7:

Algoritmo Canny Multirresolución

7.1 Introducción

En el capítulo anterior se nombraron los algoritmos incluidos en la clase `ImgBasic`. En este capítulo se expondrá una breve descripción de un algoritmo multirresolución para la extracción de contornos. Este algoritmo se basa en [MAM95] y dado que ha sufrido cambios apreciables, he creído conveniente documentar dichas modificaciones. El resto de algoritmos no están comentados, ya que son algoritmos clásicos cuya descripción se puede encontrar en la bibliografía sobre visión artificial y tratamiento de imágenes.

El algoritmo de Canny [CAN83] produce buenos resultados en imágenes sin ruido y contrastadas. Sin embargo presenta tres deficiencias: una dependencia del factor de suavizado σ escogido para el filtrado según las características de la imagen original, un error en la localización de los contornos dependiente de σ y una segmentación de los contornos cuando el suavizado de las imágenes es pequeño.

En [JRB94] se propuso un algoritmo multirresolución para la detección y clasificación de contornos en imágenes digitalizadas, utilizando la transformada wavelet. Este mismo algoritmo se implementó y amplió en [MAM95], utilizando la función gausiana para realizar el filtrado de las imágenes. Los resultados de ambos superan los resultados obtenidos con el algoritmo Canny. Sin embargo, el proceso final de recalificación de contornos, así como los algoritmos para calcular el gradiente de las imágenes presentaban aspectos que podían ser mejorados.

El algoritmo presentado en este capítulo surgió como una evolución del presentado en [MAM95] para incluirlo a modo de ejemplo en el entorno de visión que estaba realizando. Básicamente, se diferencia de éste en la utilización de un estimador por mínimos cuadrados del gradiente de la imagen sobre una ventana 2x2 en vez del

gradiente Roberts, la utilización de un filtro iterativo en vez del filtro gausiano clásico y un algoritmo de clasificación de contornos basado en el casamiento de contornos en distintos niveles utilizando una estructura de árbol en vez de utilizar únicamente el casamiento de puntos. El algoritmo de casamiento de contornos es la principal contribución de este algoritmo a la detección multirresolución de contornos, a la vez que proporciona una caracterización de los contornos continua y que hace innecesaria una recalificación posterior. A nivel de implementación del algoritmo, se han revisado, acelerado y recodificado todas las funciones tomadas de [MAM95] para poder tratar imágenes de cualquier dimensión, tanto en tonos de gris como en color.

7.2 Definición del algoritmo Canny Multirresolución.

El nombre del presente algoritmo se debe a que el esquema básico de extracción de contornos se basa en [CAN83] y [CAN86], pero en vez de limitarse a un solo nivel de filtrado de la imagen, se adapta para utilizarlo con el esquema de multirresolución. En los primeros niveles del análisis, el factor de suavizado σ del filtro gausiano es pequeño y se obtienen los contornos según los cambios de luminosidad o color –dependiendo si la imagen es en tonos de gris o es una imagen en color– en un entorno cercano del punto analizado. En los niveles más altos del análisis, el factor de suavizado σ es mayor, obtiénendose los contornos correspondientes a cambios de luminosidad o color entre zonas más alejadas de la imagen.

Este hecho es fundamental, ya que el ruido y los contornos poco significativos se detectan en los primeros niveles, pero desaparecen en las escalas más altas. Una vez que se relacionan todos los contornos que aparecen a los distintos niveles, se pueden descartar aquellos que sólo aparecen en las escalas más bajas, y mantener los que siguen apareciendo a las escalas más altas. La posición de los contornos es la detectada en la primera escala del análisis, que en el presente análisis se tomará la correspondiente a la imagen sin suavizar (σ=0), ya que en las escalas superiores aparece un desplazamiento de los contornos.

El diagrama de bloques del algoritmo Canny Multirresolución se muestra en la figura 7.1.

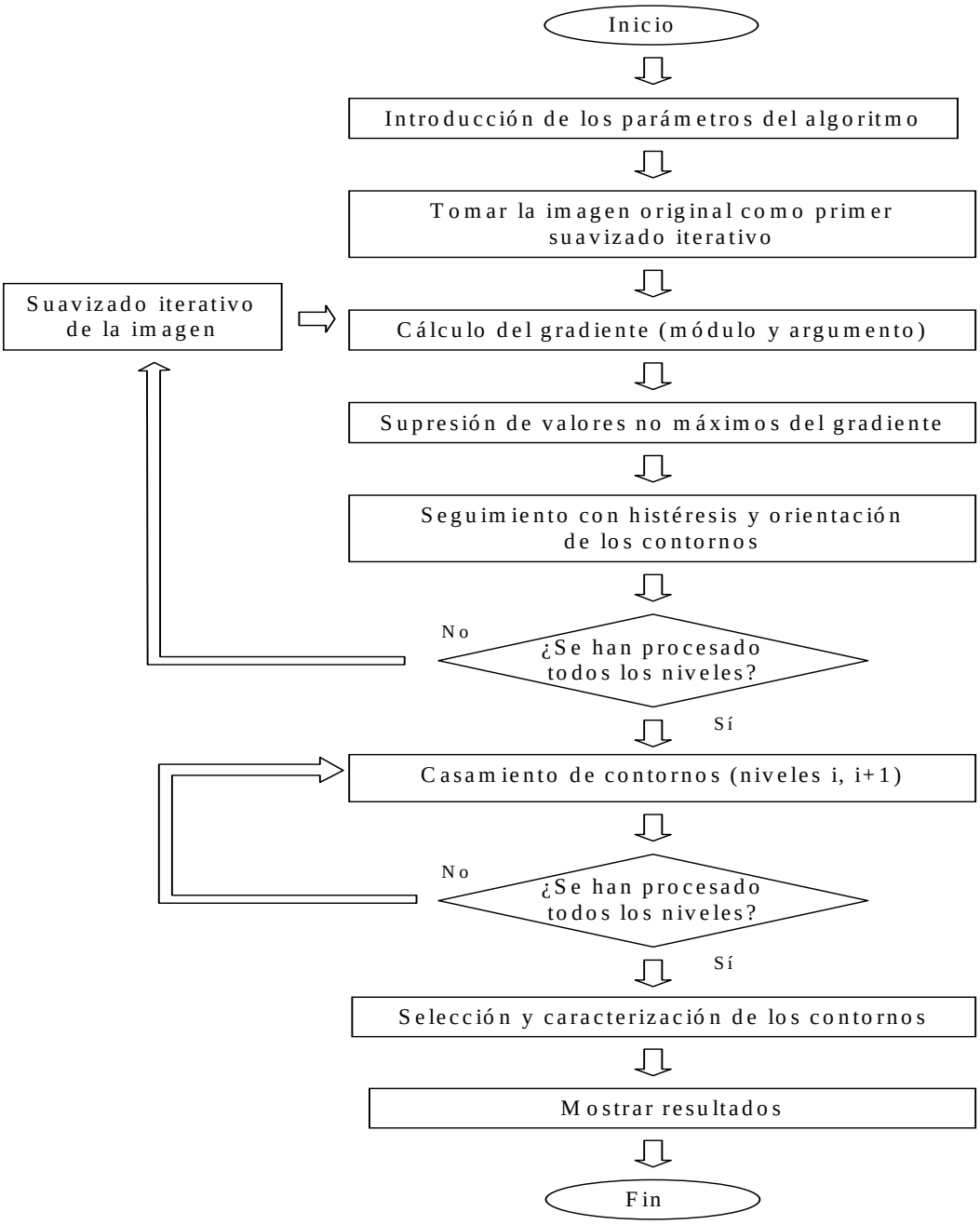


Fig. 1. Proceso usado en el algoritmo Canny Multirresolución

El casamiento de los contornos se puede realizar inmediatamente después del seguimiento de los contornos o realizarlo después de haber calculado los contornos de todos los niveles. En este proyecto se ha preferido separar obtención de contornos y casamiento en dos bucles distintos para minimizar el consumo de memoria y diferenciar claramente ambos procesos.

7.2.1 Filtrado iterativo de la imagen

En [JRB94] ya se había preferido utilizar filtrados con funciones *wavelets* al filtrado clásico para reducir los tiempos de cómputo. En [MAM95] se constataba la carga computacional que suponía el cálculo del filtrado gausiano clásico.

Para reducir el tiempo de procesado del algoritmo Canny Multirresolución se ha utilizado un filtro iterativo consistente en aplicar a la imagen un filtrado en el sentido longitudinal y otro en la dirección transversal a la imagen, con una ventana de longitud *2n-1*, donde *n* es el nivel del filtrado del nivel inferior (para el nivel uno, *n* vale uno), y cuyos elementos son todos la unidad.

En el primer nivel, únicamente se transforman los datos de la imagen guardados en la memoria como bytes (representación de 8 bits) a números enteros (representación de 16 o 32 bits). En los siguientes niveles, el filtro se aplica sobre la imagen suavizada del nivel anterior, siendo necesario para calcular cada punto de la imagen en el nivel siguiente dos sumas y dos restas, independiente del nivel de suavizado. Además las sumas y restas son en lógica entera, no de coma flotante, por lo que el filtrado es todavía más rápido. La imagen resultante tiene un factor de escala respecto la original, pero que no afecta a la supresión de puntos no máximos locales ni a la identificación de contornos.

El filtrado se realiza en dos pasos, realizándose primero un suavizado en la dirección horizontal sobre la imagen suavizada del nivel anterior y sobre este resultado, otro suavizado en la dirección vertical. De esta forma con dos filtrados 1-D obtenemos un filtrado 2-D.

Ventanas utilizadas en el algoritmo multirresolución rápida: 1, 2, 3, 5, 9 y 17
(Anchura de ventana = 2*Anchura de ventana del filtro del nivel inferior - 1)

Las ventanas se han escogido de forma que aproximadamente el factor σ equivalente se dobla en cada nivel, según se determinó en [JRB94]. En el filtrado iterativo σ aumenta entre un 60% y un 90% en cada nivel, ya que en las imágenes de las dos últimas escalas están tan difuminadas que los contornos obtenidos a partir de ellas no son significativos.

Otro factor importante para escoger la anchura de las ventanas es la derivada del filtro, ya que el objetivo del suavizado es ponderar el entorno del punto analizado al

calcular el gradiente, y de esta forma obtener contornos menos localizados. Para que la derivada del filtro sea lo más parecido a la derivada de la función gausiana, los pesos resultantes del filtrado iterativo tienen que ser también lo más parecido al filtro gausiano. En la figura 7.2. se puede ver las ventanas escogidas. En los primeros niveles, la anchura de la ventana es pequeña, y conforme aumenta el nivel, aumenta también la ventana. Por tanto, el filtrado es mayor en los niveles más altos.En la figura 7.3 se puede ver el resultado del filtrado aplicado a una dimensión.

Nivel	Anchura	Pesos	Sigma equivalente
1	1	1	0
2	2	1 2 1	0,7071
3	3	1 1 1	1,0801
4	5	1 1 1 1 1	1,7795
5	9	1 1 1 1 1 1 1 1 1	3,1358
6	17	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	5,8166

Fig. 2. Ventanas de filtrado utilizadas en el filtro iterativo

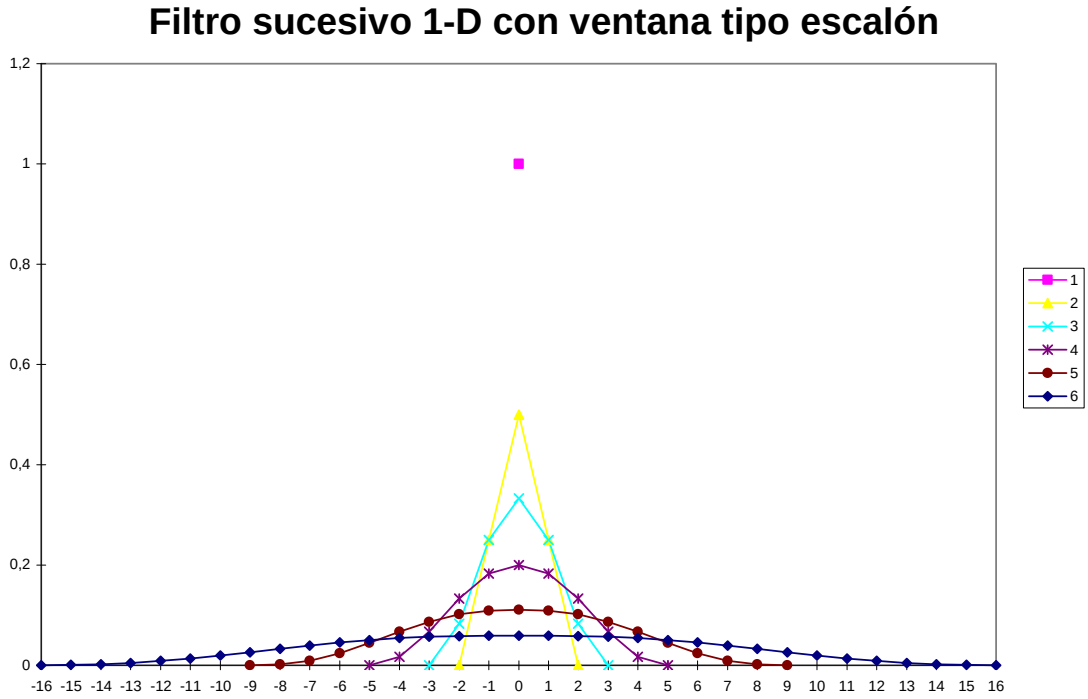


Fig. 3. Pesos resultantes de la aplicación iterativa del filtrado 1-D.

Principio de funcionamiento:

Se suma el punto que entra en la ventana y se resta el que sale. Primero se realiza el filtrado en la dirección horizontal y luego en la vertical (o a la inversa, ya que el resultado no cambia). Si la anchura de la ventana *w* es *impar*, (la imagen original es *orig* y la imagen filtrada es *final*), la iteración para un nivel se puede formular matemáticamente como:

$$final[i,\, j] = \sum_{l=i-\frac{n-1}{2}}^{i+\frac{n-1}{2}} \sum_{k=j-\frac{n-1}{2}}^{j+\frac{n-1}{2}} orig[l,k]$$

Si separamos el sumatorio correspondiente a las filas y las columnas y llamamos *inter* a la solución intermedia de aplicar el sumatorio de las filas, podemos representar el resultado en función del punto anterior, reduciendo el número de cálculos.

$$inter[i,\, j] = \sum_{k=j-\frac{n-1}{2}}^{j+\frac{n-1}{2}} orig[i,k] = inter[i,\, j-1] - orig\left[i,\, j-\frac{n+1}{2}\right] + orig\left[i,\, j+\frac{n-1}{2}\right]$$

$$final[i,\, j] = \sum_{l=i-\frac{n-1}{2}}^{i+\frac{n-1}{2}} inter[i,\, j] = final[i-1,\, j] - inter\left[i,\, j-\frac{n+1}{2}\right] + inter\left[i,\, j+\frac{n-1}{2}\right]$$

Casos especiales:

Filtro con anchura de ventana unidad: Se toma la imagen directamente.

Filtro con anchura de ventana dos: Si se toma la anchura de ventana 2, se produce un desplazamiento de medio pixel de la imagen filtrada respecto de la original. Dado que con suavizados pequeños interesa tener una localización precisa, se ha tomado una ventana equivalente de tres pixeles, sopesando el pixel central el doble que los que se encuentran a derecha e izquierda, que no tiene desplazamiento.

Filtro con anchura de ventana par (excepto 2). Se calcula normalmente, produciéndose un desplazamiento de medio pixel hacia arriba y a la derecha (con ventanas grandes no es un problema).

Coste computacional

Número de sumas:
2*(anchura imagen + anchura ventana)*(altura imagen + anchura ventana)
Número de restas:
2*(anchura imagen + anchura ventana)*(altura imagen + anchura ventana)

Si aplicamos el filtro a una imagen, el resultado sería semejante a la función gausiana bidimensional. A modo de ejemplo, en la figura 7.4. se muestra los pesos correspondientes al filtrado del cuarto nivel.

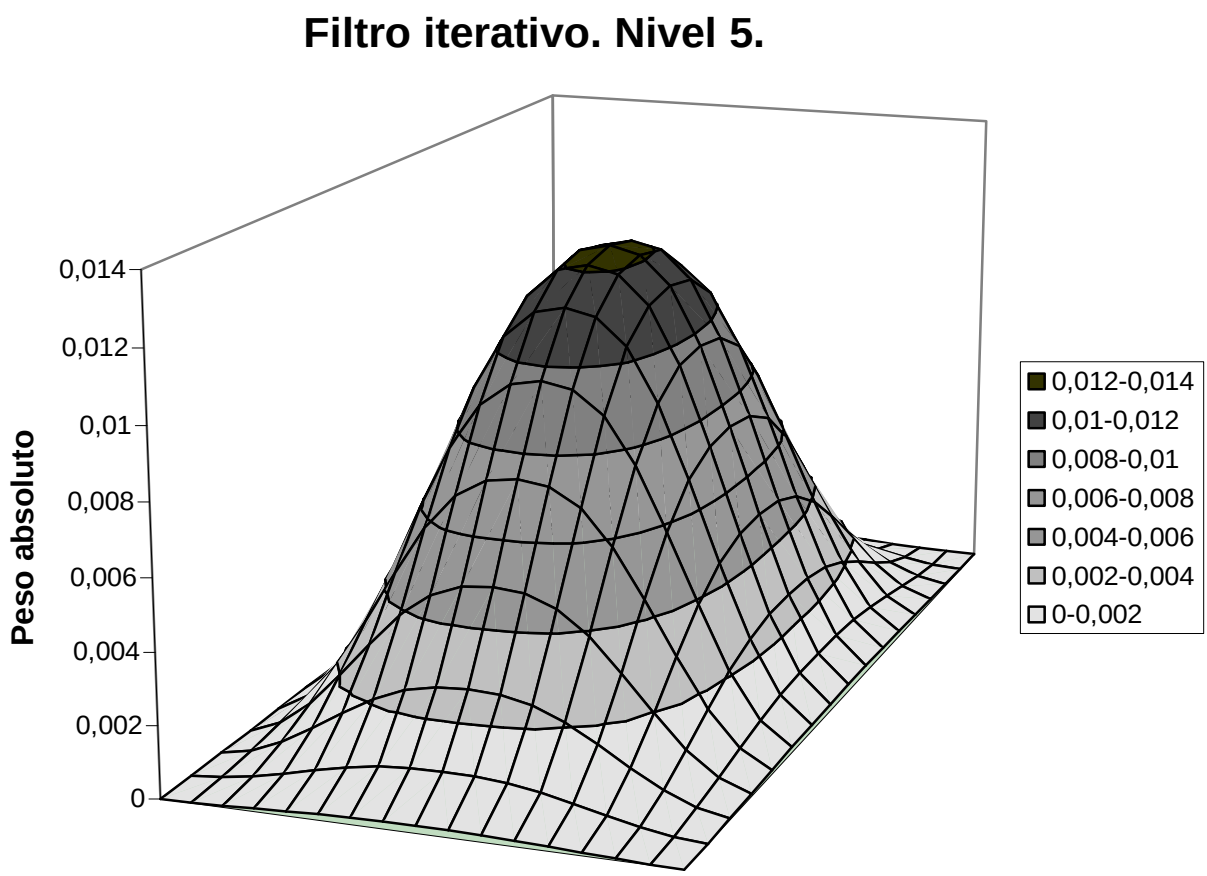


Fig. 4. Pesos resultantes de la aplicación iterativa del filtrado 2-D para el nivel 5.

El algoritmo se ha programado para que se produzca el orlado[1] la imagen con reflejos especulares de la original ya que éste es el orlado que se recomienda en [MAM95]. Se usan cada uno de los bordes de la imagen como eje de simetría para los cálculos antes descritos, como indica la figura 7.5.

Se necesita un bloque de memoria donde colocar la imagen suavizada y dos bloques adicionales del tamaño de una fila o una columna (lo que tenga mayor longitud). La necesidad de tener mayor precisión en la imagen filtrada, multiplica por

[1] Por orlado se entiende el criterio que se toma para calcular los puntos de la ventana de filtrado que se encuentran fuera de la imagen porque han sobrepasado los límites de la misma. En el orlado especular se toma el valor de los puntos reflejados respecto el borde de la imagen.

cuatro –depende del tipo de almacenamiento de la imagen suavizada– los requerimientos de memoria respecto la imagen sin filtrar.

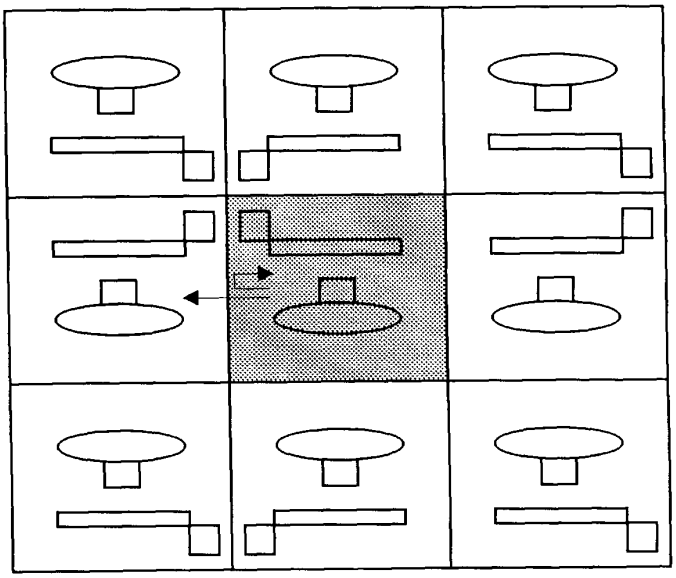


Fig. 5. Orlado de imágenes especular. Fuente: [MAM95].

Como conclusión de este apartado he de comentar que el filtrado arriba expuesto es muy rápido y se adapta muy bien a las necesidades del algoritmo multirresolución. Para comprobar la poca influencia sobre los resultados de utilizar el filtro gausiano o el iterativo, se ha sustituido el filtrado gausiano por el arriba expuesto en el algoritmo diseñado en [MAM95]. Los resultados son muy similares pero los tiempos de procesado son mucho menores (el coste computacional para calcular las imágenes suavizadas ha pasado de ser el más importante a uno de los menores, por debajo del coste del cálculo del gradiente y de la supresión de puntos no máximos).

La poca relevancia del filtro utilizado para el suavizado de la imagen también se puede comprobar en [JRB94], en donde los resultados obtenidos con el filtrado con funciones *wavelet* son también comparables.

7.2.2 Estimación del módulo y la dirección del gradiente

En [MAM95] se analizaron varios operadores para la detección de contornos, como los operadores diferencia, Sobel, Roberts, etc. Como resultado del citado estudio,

se concluyó que el mejor resultado se obtenía con el operador de Roberts. Sin embargo, para obtener resultados óptimos, se utilizaban varias expresiones que permitían calcular el módulo del gradiente a partir de las dos componentes del mismo, dependiendo del grado de suavizado de la imagen. Por tanto, se podía concluir que si bien se había llegado a una solución para el cálculo del gradiente, esta no era definitiva.

La dificultad encontrada para obtener una fórmula universal para el cálculo del módulo del gradiente, es decir, que de resultados óptimos independientemente del nivel de suavizado de la imagen, se debe a que el módulo resulta infra o sobreponderado dependiendo de la dirección del gradiente. Para calcular la influencia de la dirección del gradiente en el cómputo del módulo, se ha tomado una ventana 2x2 centrada en el origen de los ejes X e Y, cuyos valores son los correspondientes a una función lineal $z = A \cdot x + B \cdot y + C$, donde $A = -M \cdot sen(\theta)$, $B = M \cdot cos(\theta)$, y C es el nivel medio de la luminosidad o del color en la ventana 2x2. Esta función corresponde a un plano con pendiente M respecto al plano definido por los ejes X e Y , y con la orientación respecto a estos ejes θ.

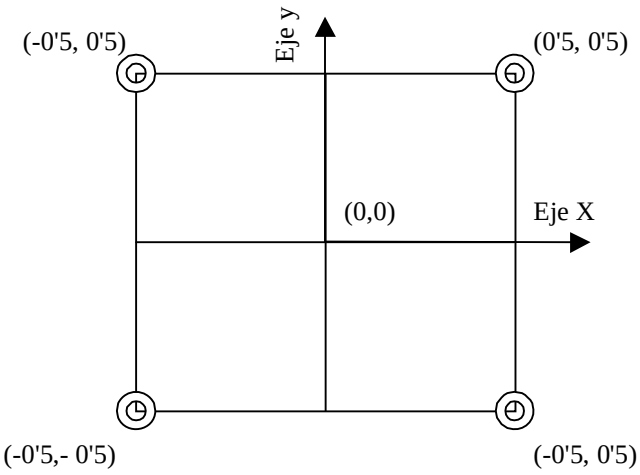


Fig. 6. Ventana 2x2 utilizada en el cálculo del gradiente.

Por tanto, los valores de la imagen en los cuatro puntos de la imagen son:

$z(-0'5, 0'5) = 0'5 \cdot M \cdot sen(\theta) + 0'5 \cdot M \cdot cos(\theta) + C$; $z(0'5, 0'5) = -0'5 \cdot M \cdot sen(\theta) + 0'5 \cdot M \cdot cos(\theta) + C$;

$z(-0'5,- 0'5) = 0'5 \cdot M \cdot sen(\theta) - 0'5 \cdot M \cdot cos(\theta) + C$; $z(0'5,-0'5) = -0'5 \cdot M \cdot sen(\theta) - 0'5 \cdot M \cdot cos(\theta) + C$;

El algoritmo óptimo debe devolver como módulo del gradiente M, y como dirección del gradiente θ. En la figura 7.7. se puede observar la desviación del módulo respecto al valor teórico.

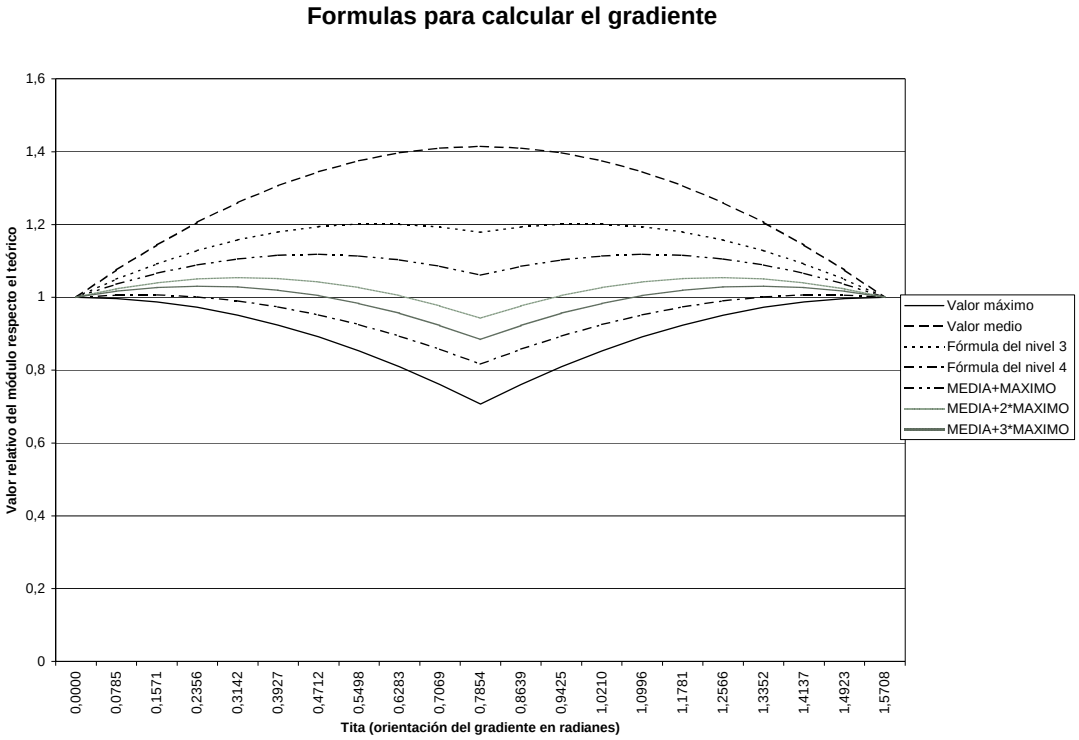


Fig. 7. Módulo del gradiente calculado según la referencia [MAM95].

El módulo y el argumento del gradiente se calculan en el algoritmo Canny Multirresolución a través de un estimador de mínimos cuadrados sobre una ventana 2x2. De esta forma, el valor teórico y el obtenido a través del estimador coincide, además de proporcionar un comportamiento más favorable frente al ruido. Si el estimador de mínimos cuadrados se aplicase a una ventana 3x3, el coste computacional se incrementa, además de producir un cierto suavizado en el cálculo del gradiente. Dado que ese suavizado ya se obtiene a través del filtrado de la imagen, la ventana más adecuada es 2x2.

Si consideramos la imagen como una superficie tridimensional, en donde x e y corresponden a las coordenadas del punto estudiado, y la altura corresponde a la luminosidad o color en ese punto, minimizar el error cuadrático respecto de la función $z = -M \cdot sen(\theta) \cdot x + M \cdot cos(\theta) \cdot y + C$ equivale a ajustar un plano a la superficie en el punto

donde se calcula el gradiente, y tomar como dirección del gradiente, la orientación del vector característico proyectado sobre el plano *XY*, y como módulo del gradiente la pendiente sobre el plano *XY*.

Las fórmulas obtenidas del ajuste por mínimos cuadrados, para un punto [i,j] son:

$$\mathrm{tg}(\theta) = \frac{imagen[i-1,\, j] - imagen[i,\, j-1] + imagen[i-1,\, j-1] - imagen[i,\, j]}{imagen[i-1,\, j] - imagen[i,\, j-1] - imagen[i-1,\, j-1] + imagen[i,\, j]}$$

máximo = max(abs(*imagen*[i,j]- *imagen*[i-1,j-1]), abs(*imagen*[i-1,j]- *imagen*[i,j-1]));

mínimo = min(abs(*imagen*[i,j]- *imagen*[i-1,j-1]), abs(*imagen*[i-1,j]- *imagen*[i,j-1]));

$$módulo \;=\; \frac{máximo}{\sqrt{2}}\sqrt{1+\left(\frac{mínimo}{máximo}\right)^2}$$

El valor del módulo puede ser aproximado por el polinomio de Taylor, tomando como variable *mínimo/máximo:*

$$módulo \;=\; \frac{máximo}{\sqrt{2}}\left[1+\frac{1}{2!}\left(\frac{mínimo}{máximo}\right)^2-\frac{3}{4!}\left(\frac{mínimo}{máximo}\right)^4+\frac{5}{6!}\left(\frac{mínimo}{máximo}\right)^6+\ldots\right]$$

Dado que el filtro iterativo genera la imagen suavizada con un factor de escala conocido, el factor $\frac{1}{\sqrt{2}}$ se puede incluir en dicho factor de escala. El resultado de la supresión de puntos no depende de este factor, y sin embargo se evita una multiplicación por cada punto de la imagen. Tampoco se tiene en cuenta que el cálculo del gradiente está desplazado medio pixel en el eje *X* e *Y* de la imagen, ya que este desplazamiento se da en todas las escalas. La fórmula final empleada en el cálculo es:

$$módulo\,[i,\, j] \;=\; \text{máximo} + \frac{mínimo \cdot mínimo}{2\,máximo}$$

Como nota técnica se ha de decir que el módulo del gradiente se guarda como una matriz de números reales de tipo *float* (4 bytes de precisión), mientras que la dirección del gradiente se guarda como una matriz de bytes, ya que no se necesita mayor precisión, correspondiendo un valor 0 a 0º y 256 a 360º. Para acelerar el cálculo del argumento se utiliza una tabla de arcotangentes precalculadas, evitanto de esta

forma el cómputo de la función *atan2*. El error que se produce en el cálculo de la dirección no afecta al algoritmo Canny Multirresolución, ya que la precisión de almacenamiento de la dirección es solamente un byte, ni al cálculo del módulo del gradiente, ya que no utiliza el valor de tita.

7.2.3 Tratamiento de las imágenes en color

En las imágenes en tonos de gris, la propiedad a la cual se le aplica el gradiente es la luminosidad. Sin embargo, si utilizamos la luminosidad para calcular los contornos en las imágenes en color se pierde información, ya que los cambios de color que no supongan un cambio de luminosidad (por ej., un cambio de color de rojo a azul) no se detectan. Por ello en las imágenes en color, para hallar el módulo del gradiente se calcula las diferencias de las componentes roja, verde y azul en cada par de puntos, y luego se calcula la diferencia equivalente sopesando cada componente con un coeficiente (este coeficiente puede ser el factor de luminosidad del color rojo, verde o azul, o directamente la unidad, para acelerar los cálculos). La ponderación de las componentes se realiza en la función *mono*.

máximo = max (
mono(abs(*rojo*[i,j]-*rojo*[i-1,j-1]), abs(*verde*[i,j]-*verde*[i-1,j-1]), abs(*azul*[i,j]- *azul*[i-1,j-1])),
mono(abs(*rojo*[i-1,j]-*rojo*[i,j-1]), abs(*verde*[i-1,j]-*verde*[i,j-1]), abs(*azul*[i-1,j]- *azul*[i,j-1]))
);

mínimo = min (
mono(abs(*rojo*[i,j]-*rojo*[i-1,j-1]), abs(*verde*[i,j]-*verde*[i-1,j-1]), abs(*azul*[i,j]- *azul*[i-1,j-1])),
mono(abs(*rojo*[i-1,j]-*rojo*[i,j-1]), abs(*verde*[i-1,j]-*verde*[i,j-1]), abs(*azul*[i-1,j]- *azul*[i,j-1]))
);

$$módulo\,[i,\ j]\ =\ \text{máximo}\ +\ \frac{mínimo\ \cdot mínimo}{2\,máximo}$$

En el cálculo de la dirección no se puede usar la estrategia de utilizar el valor absoluto de la diferencia porque solamente de esta forma se obtendrían argumentos en el primer cuadrante. Para tener resultados en los cuatro cuadrantes, se debe calcular el signo de las componentes.

diagonal1 = mono(abs(*rojo*[i,j]-*rojo*[i-1,j-1]), abs(*verde*[i,j]-*verde*[i-1,j-1]),

abs(*azul*[i,j]- *azul*[i-1,j-1]));

diagonal2 = mono(abs(*rojo*[i-1,j]-*rojo*[i,j-1]), abs(*verde*[i-1,j]-*verde*[i,j-1]),
abs(*azul*[i-1,j]- *azul*[i,j-1]));

if (mono(*rojo*[i-1,j-1],*verde*[i-1,j-1],*azul*[i-1,j-1]) < mono(*rojo*[i,j],*verde*[i,j],*azul*[i,j]))

diagonal1 = - diagonal1;

if (mono(*rojo*[i,j-1],*verde*[i,j-1],*azul*[i,j-1]) < mono(*rojo*[i-1,j],*verde*[i-1,j],*azul*[i-1,j]))

diagonal12= - diagonal2;

$$\mathrm{tg}(\theta) = \frac{diagonal1 - diagonal2}{diagonal1 + diagonal2}$$

7.2.4 Eliminación de puntos no máximos

Al igual que el algoritmo Canny [CAN83], una vez realizado el suavizado gausiano y el cálculo del gradiente, se eliminan los puntos que no son máximos locales.

Para cada uno de los puntos detectados como candidatos a contorno por la derivada primera se comprueba si son o no máximos locales en la dirección de la variación máxima indicada por el gradiente. Para ello, y según el esquema de la figura 7.8., se calcula el valor de los puntos G1 y G2. Estos valores se obtienen ponderando los valores de los dos pixels, para cada caso, más cercanos a los puntos G1 y G2.

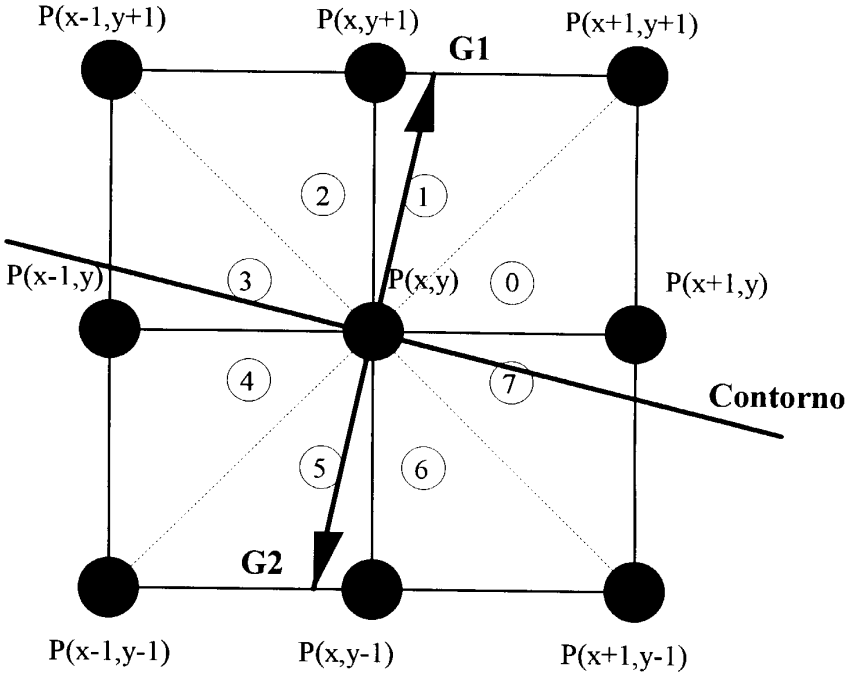


Fig. 8. Algoritmo de supresión de valores no máximos. Fuente: [MAM95].

En el caso de la figura 7.7. donde 45º<angulo<90º o 225º<angulo<270º, el cálculo de G1 y G2 se realiza según las siguientes expresiones:

G1 = *imagen*[i,j+1] + (*imagen*[i+1,j+1] - *imagen*[i,j+1])·cotan(angulo);

G2 = *imagen*[i,j-1] + (*imagen*[i-1,j-1] - *imagen*[i,j-1])·cotan(angulo);

Si los valores calculados G1 y G2 son inferiores al valor del pixel *imagen*[i,j], este punto se considera máximo local en vecindad 3x3 y es considerado como candidato a contorno. Si por el contrario G1 o G2 es superior o igual a *imagen*[i,j], este punto es desechado como borde de la imagen.

Como medida para acelerar el cálculo de la supresión de puntos no máximos, no se analizan los puntos cuyo módulo de gradiente no superan un umbral mínimo. Estos puntos directamente se desestiman como puntos contorno. Si el umbral se elige adecuadamente se favorece el comportamiento de los algoritmos de casamiento de segmentos, ya que se eliminan puntos irrelevantes que podrían ser marcardos como contornos.

7.2.5 Agrupamiento de puntos de contorno en segmentos

Una vez que se han calculado todos los puntos contorno, hace falta un algoritmo que los agrupe en segmentos[*]. Una opción es utilizar el algoritmo utilizado en [CAN83], buscando todos los puntos contorno en los ocho vecinos posibles. Este algoritmo fue el que se empleó en [MAM95].

Sin embargo, esta estrategia une segmentos que pertenecen a distintos contornos, ya que no tiene en cuenta que los puntos pertenecientes a un mismo contorno deben estar alineados en la dirección del contorno. Por tanto, los puntos detectados en dirección perpendicular al contorno se deben considerar como pertenecientes a segmentos distintos.

[*] Nota: a partir de este apartado se utilizará la palabra segmento para referirse a un conjunto de puntos agrupados, sin necesidad de que estos estén ordenados ni alineados respecto a una recta, y que pertenecen a lo que normalmente se entiende por contorno

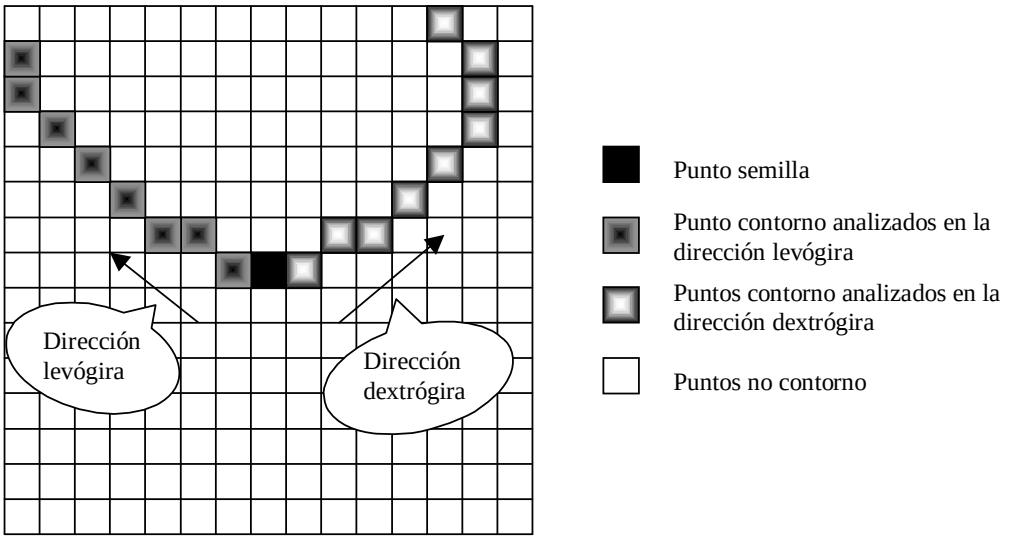


Fig 9. Crecimiento de un segmento a partir de un punto cortorno semilla.

El algoritmo funciona de la siguiente manera: cuando detecta un punto con módulo del gradiente superior a un umbral, se genera una semilla de contorno. Primero se buscan los puntos contorno en sentido levógiro, teniendo en cuenta que éstos tienen que estar alineados con el contorno. La dirección del contorno es la del último punto incluido en el segmento, y los candidatos a engrosar la lista de los puntos del segmento son los tres vecinos de los ocho posibles que están más cerca de la dirección del contorno, en el sentido levógiro (en el ejemplo de la figura 7.8, son los puntos P(x-1,y), P(x-1,y+1) y P(x-1,y-1). Una vez que ya no se encuentran puntos contorno en la dirección levógira, se pasa a buscar puntos contorno en dirección dextrógira a partir de la semilla. Los candidatos a ser incluidos en la lista de puntos del segmento son los tres vecinos en dirección del contorno, pero en dirección contraria a la levógira. En la figura 7.8., los puntos equivalentes son P(x+1,y), P(x+1,y-1) y P(x+1,y+1).

Como principal crítica del algoritmo de agrupamiento de contornos es que tiende a fragmentar los contornos más de lo que debiera en las primeras escalas. En la imagen original la orientación del gradiente presenta ruido superpuesto y en algunos puntos no coincide con la dirección del contorno. A partir del tercer nivel apenas se rompen los contornos. También se fragmentan segmentos cuando el radio de curvatura es pequeño, como ocurre en las esquinas, lo cual puede ser beneficioso si el algoritmo Canny Multirresolución se utiliza para ajustar curvas o rectas a los contornos.

7.2.6 Cálculo de la correspondencia entre segmentos

El principal problema del algoritmo Canny Multirresolución es realizar un casamiento de los puntos contorno que aparecen en los distintos niveles. Una primera aproximación se da en [JRB94], que posteriormente se utiliza en [MAM95]. En estas referencias se realiza una búsqueda de puntos correspondientes partiendo del punto contorno en el nivel superior, descendiéndose a las escalas inferiores, empezando a buscar en el mismo punto de la imagen, y desplazándose a los puntos adyacentes en los dos sentidos marcados por la dirección del gradiente de la escala superior[CAN83]. Se considera el punto superior casado con uno de la escala inferior cuando se encuentra un punto contorno cuyo gradiente tiene una dirección similar a la del punto superior (la diferencia absoluta en grados entre las orientaciones debe ser menor de una constante). Se considera el punto superior sin casar cuando se excede la distancia máxima entre punto analizado y el original para el nivel del punto analizado (Beltrán [JRB94] demuestra que el desplazamiento máximo que sufre un punto contorno es menor que 2σ). Si el punto superior ya ha sido casado, se continúa analizando el punto a la escala inmediatamente inferior, tal como se ha indicado. El proceso se finaliza cuando se llega al primer nivel, en tal caso se marca el punto del primer nivel como contorno, o cuando no se encuentra la correspondencia de un punto con otro de la escala inferior. Este proceso se repite tantas veces como puntos contorno se hayan detectado en las escalas superiores.

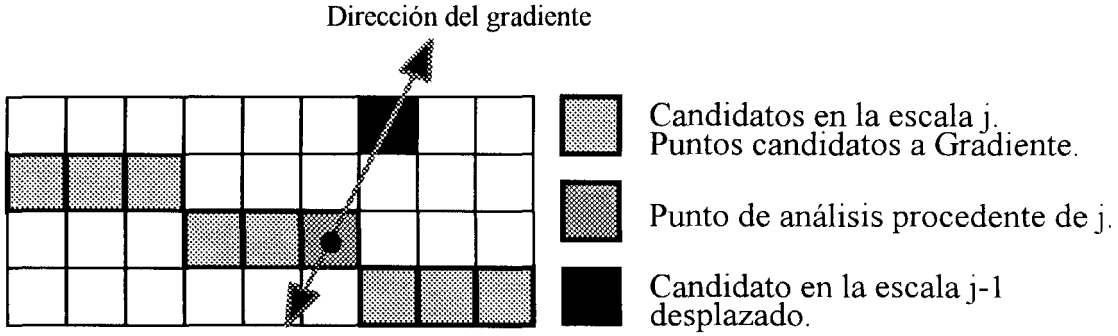


Fig. 11. Búsqueda de un punto desplazado en análisis descendente. Fuente: [MAM95].

El resultado obtenido con esta estrategia es claramente superior al obtenido con Canny, ya que no se consideran puntos contorno aquellos que no aparecen en las escalas superiores. Sin embargo los contornos se tienden a romper, ya que el desplazamiento de

los segmentos en los distintos niveles provocan que éstos se expandan o se contraigan, y sólo se puede marcar un punto en la escala inferior por cada punto en la escala superior. Si un contorno se expande, aumenta el número de puntos contorno con respecto al anterior y alguno de los puntos en la escala inferior no tienen correspondencia con un punto de la escala superior, con lo cual el contorno se fragmenta.

En el presente algoritmo se ha solucionado este problema casando los segmentos. De esta forma, aunque se contraiga o se expanda el contorno, no se produce su fragmentación. Además, este enfoque es más robusto, ya que el contorno sigue siendo válido aunque una pequeña parte de los puntos no se pueda casar, y es menos costoso tratar la información correspondiente a cada segmento en vez de cada punto.

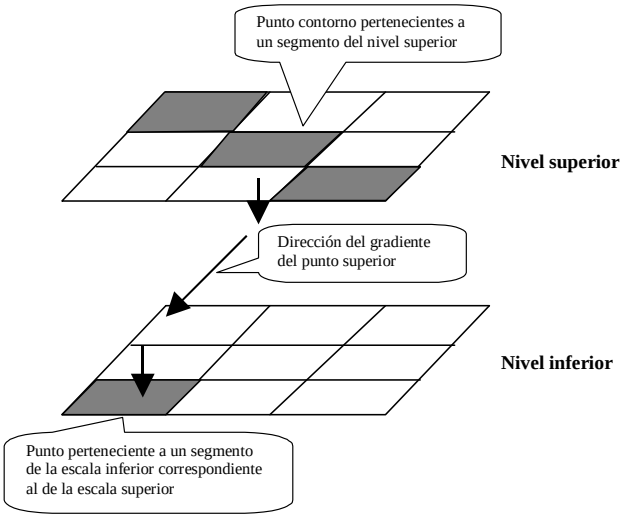


Fig. 12. Representación gráfica del casamiento de segmentos.

Cada segmento puede tener un solo segmento padre en la escala superior, aunque un segmento padre puede tener varios hijos en el nivel inferior o ninguno. El casamiento de segmentos se realiza por capas, empezando por los niveles 1 y 2, siguiendo por 2 y 3, sucesivamente hasta llegar al nivel máximo de análisis. El orden en que se realiza el casamiento es irrelevante, ya que cuando se han casado todas las capas es cuando se define el árbol de segmentos con sólo seguir la tabla de casamiento. Además, de esta forma se puede conservar el resultado de las escalas superiores con muy poco gasto de memoria, para poderlas utilizar en el tratamiento de imágenes semejantes, como las que se obtienen durante la captura de imágenes consecutivas por una cámara instalada en un robot. En estos casos, el algoritmo de casamiento de segmentos empieza relacionando los segmentos de la capa del nivel inferior (nivel 1)

con los del nivel superior (nivel 2), y así sucesivamente hasta la escala más baja que se conserva del fotograma anterior, aprovechando de éste el resto de casamientos. Otra ventaja adicional es que todos los segmentos correspondientes a un contorno se obtendrían a partir de un único contorno padre, por lo cual se simplificaría el análisis de características de la imagen, ajustes de contornos a rectas o curvas, etc.

Para identificar cada segmento, tanto en la capa inferior y superior, se les asigna un número partiendo de uno hasta el número máximo de contornos en el nivel. El segmento cero (segmento nulo) se reserva para indicar que no se ha producido una correspondencia , y por lo tanto el segmento que tiene como padre al segmento nulo, no aparece en las escalas superiores.

El algoritmo utilizado para casar segmentos examina uno a uno todos los segmentos del nivel superior y calcula el número de puntos en común que tienen con los segmentos inferiores. Para cada segmento superior se buscan todos los puntos del nivel inferior correspondientes a puntos contenidos en el segmento superior y para cada contorno inferior se toma como padre aquel que tenga más puntos correspondientes. Si el número de puntos del segmento inferior que corresponden a los del segmento superior es mayor que un porcentaje sobre el número de puntos totales del segmento inferior, se valida la relación entre los segmentos; en caso contrario, se asigna como padre el segmento nulo. El árbol resultante de este esquema algorítmico parte de los segmentos en el nivel 1 (imagen sin suavizar) hacia jerarquías superiores. Como cada segmento tiene un solo padre, es suficiente una tabla para cada nivel para guardar las correspondencias de los segmentos.

Otra forma alternativa para calcular las correspondencias entre segmentos podría ser analizar uno a uno los segmentos inferiores y buscar al padre. De los segmentos superiores que tuvieran algún punto correspondiente, bastaría con tomar aquel que tuviera más puntos correspondientes al segmento inferior. Este algoritmo es algo más simple que el utilizado en el presente proyecto, pero tiene la desventaja que para casar los puntos se utiliza los valores de la dirección del gradiente del segmento inferior y en las primeras escalas (sobre todo la primera) contienen más ruido que los de los segmentos superiores.

En el presente proyecto se ha utilizado un árbol ascendente para guardar la relación entre segmentos, partiendo de los segmentos en el nivel base hacia los niveles superiores. Otra posible alternativa es realizar un árbol descendente, en donde a partir de todos los segmentos padre superiores se fueran relacionando todos los segmentos hijos. En esta alternativa ya no se podrían utilizar tablas de casamiento (ya que un padre puede tener varios hijos) sino que sería necesario utilizar listas y cada padre superior además tendría que contener el número del nivel a que pertenece. Dado que recorrer todas las ramas de un árbol, controlar el nivel del segmento estudiado y mantener las listas de los segmentos padre es más complicado y menos eficiente que tener una tabla para cada nivel, se ha tomado la estructura de árbol ascendente en el proyecto.

Otra ventaja adicional de la estructura de datos elegida es que los segmentos que no tengan segmentos hijos asociados en las escalas base (correspondiente a la imagen sin suavizar) no son significativos en este análisis. Si se utiliza un árbol ascendente éstos no aparecen, pero si el árbol es descendente éstos aparecerían cuando un padre no tenga ningún segmento hijo.

7.2.7 Caracterización de los segmentos

La caracterización de los contornos no ha sido estudiada en el presente proyecto, y depende en gran medida del uso posterior de la extracción de contornos.

A modo de ejemplo, se ha utilizado para mostrar las imágenes segmentadas una luminosidad de los contornos equivalente a la media del gradiente de los segmentos en los distintos niveles analizados. Para cada nivel, se ha guardado el valor de la media del gradiente para cada segmento. Si en un nivel no hay correspondencia de ese contorno, el valor de la media para ese nivel es cero. Con las medias correspondientes a los niveles analizados, se calcula otra vez la media para cada segmento. Este resultado da una idea de la importancia del contorno, ya que cambios bruscos en la tonalidad pero muy localizados –como los caracterizados como tipo pulso en [JRB94] y [MAM95]– dan gradientes altos en las primeras escalas y gradientes bajos en las escalas superiores (si

no es que desaparecen en las escalas más altas). Sin embargo, los contornos que aparecen como cambios glogales de la luminosidad o color en la imagen tienen gradientes bajos en las escalas inferiores pero aumentan en las superiores, por lo cual la media se compensa. En la figura 7.13. se reproduce la variación del módulo del contorno según la caracterización tipo rampa, escalón, pulso y escalera.

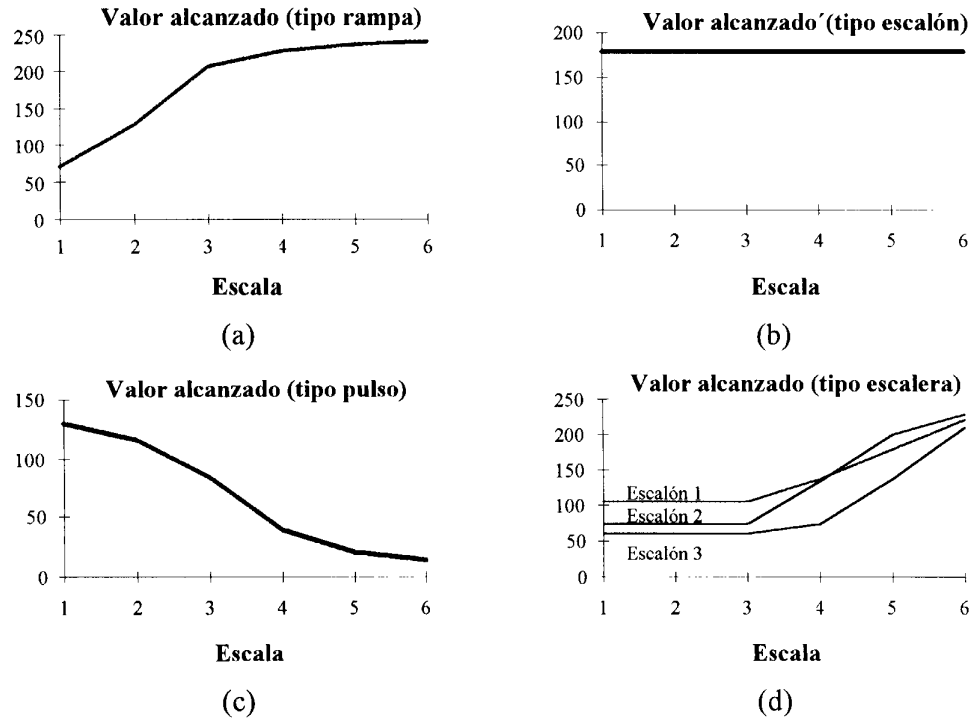


Fig. 13. Variación de los valores de un punto máximo local en función del nivel de análisis (a) rampa, (b) escalón, (c) pulso, (d) escalera. Fuente: [MAM95].

Otro tipo de caracterización es el número de puntos que tiene un contorno, no como suma de los puntos que aparecen en un determinado segmento del nivel inferior sino como la suma de los puntos contenidos en todos los segmentos que tienen un padre común (ya sea a una escala determinada o en la escala más alta en que exista segmento padre –padre superior). Esta caracterización también se ha incluido en el proyecto como validación adicional de los contornos, despreciando los contornos que tuvieran un número de puntos inferior a una longitud mínima de contornos. Esta selección de contornos se ha mostrado innecesaria, ya que con el criterio de sólo aceptar segmentos que estuvieran presentes en un nivel del análisis, es suficiente para eliminar el ruido. En el algoritmo actualmente implementado se ha tomado como longitud de un contorno la suma de la longitudes de todos los contornos del nivel base que cuelgan de un mismo

segmento padre, que tiene el inconveniente que en los niveles superiores (típicamente 5 y 6) se produce un acople de contornos distintos debido a que las imágenes están tan difuminadas que los contornos se juntan en uno sólo. Para evitar esto, se debería limitar el nivel máximo al cual se calcula la longitud de los segmentos.

Una posible caracterización es la tendencia del gradiente a aumentar de valor en el segmento. En [JRB94] se puede observar una clasificación de contornos en cuatro tipos más otro desconocido, según su evolución a lo largo de los niveles multirresolución. Sin embargo, una caracterización continua del tipo de contorno sería más robusta y aportaría más información. Como se indica en [JRB94] y [MAM95], el valor del gradiente en los contornos tipo escalón permanece constante en todos los niveles del análisis, en los contornos tipo rampa el gradiente aumenta en los niveles superiores y en los contornos tipo pulso disminuye en los niveles superiores. Dado que el número de segmentos presentes en una imagen es mucho más reducido que el número de puntos, y el valor medio del gradiente en cada nivel de un contorno es un valor robusto (se calcula como la media del gradiente en los puntos que lo forman), una posible caracterización sería el valor medio del gradiente para los distintos niveles, o incluso la pendiente de una recta ajustada por mínimos cuadrados a los valores del gradiente en cada uno de los niveles.

Otras caracterizaciones del segmento podrían estar relacionadas con la expansión o contracción del contorno a lo largo del análisis multirresolución, el porcentaje de correspondencia o casamiento entre los contornos superiores e inferiores, el desplazamiento típico de los contornos a lo largo de los niveles multirresolución, etc.

7.3 Coste computacional

El tiempo de procesado se ha reducido drásticamente respecto del que se dá en [MAM95] debido a la utilización del filtro iterativo, y secundariamente por la optimización de algoritmos y la utilización de toda la potencia de la aritmética de punteros del lenguaje de programación C.

A continuación se muestra con la memoria requerida, en bytes, para ejecutar el algoritmo en una imagen en tonos de gris, de tamaño *nxm*. El número de niveles multirresolución es *maxlevel*, y las constantes *ratio1* y *ratio2* indican la relación entre el máximo número de puntos contorno y segmentos que se pueden encontrar en la imagen respecto al máximo teórico para una imagen de dichas dimensiones. Ratio1 y ratio2 toman el valor de 1 para el programa compilado en el entorno, dado que no se conocen características de las imágenes a tratar ni parámetros que se van a utilizar y otros valores mayores que uno provocarían un fallo por falta de memoria. Para que quede claro los requerimientos de memoria, se indican los valores ocupados por las estructuras en un compilador de 32 bits (Visual C 4.0), ya que el valor del alineamiento de estructuras puede variar los requerimientos de memoria en distintos compiladores.

Imagen original:	n·m
Imagen suavizada:	4·n·m
Gradiente de la imagen (módulo)	4·n·m
Gradiente de la imagen (argumento)	n·m
Supresión de puntos no máximos	4·n·m
Buffer para almacenar los puntos contornos	maxlevel·5·n·m/(2·ratio1)
Buffer para almacenar los segmentos	maxlevel·16·n·m/(2·ratio2)
TOTAL (imágenes en tonos de gris)	menor que (14 + maxlevel·10,5)·n·m

Para las imágenes en color la tabla es la siguiente:

Imagen original:	3·n·m
Imagen suavizada:	12·n·m
Gradiente de la imagen (módulo)	4·n·m
Gradiente de la imagen (argumento)	n·m
Supresión de puntos no máximos	4·n·m
Buffer para almacenar los puntos contornos	maxlevel·5·n·m/(2·ratio1)
Buffer para almacenar los segmentos	maxlevel·16·n·m/(2·ratio2)
TOTAL (imágenes en tonos de gris)	menor que (24 + maxlevel·10,5)·n·m

En la presente tabla, se ha señalado sólo el momento crítico durante el proceso. El consumo de memoria se podría reducir si no es preciso conservar la imagen original en memoria, se modifica el algoritmo de supresión de máximos locales de forma que utilice una única imagen o se aumentan los ratios de utilización de memoria respecto al caso más desfavorable.

Con respecto al tiempo de cálculo, no se han realizado mediciones sobre los algoritmos. El coste medio de cálculo, para una imagen 256x256 está por debajo de un segundo. El tiempo de filtrado es despreciable. Sin embargo, el tiempo de cálculo del gradiente, la supresión de puntos no máximos y el seguimiento de contornos dependen del número de niveles para los que se realiza el análisis multirresolución. Por tanto, el tiempo de cálculo es proporcional al número de niveles calculado.

Dado que los resultados obtenidos utilizando niveles de multirresolución mayores que cuatro es muy reducido, no es rentable aumentar por encima de éste el nivel de análisis. Si se quisiera trabajar en estos niveles, a partir de la capa cuarta se puede aumentar la escala espacial. De esta forma se reduciría el tiempo de cálculo a la mitad.

Para aplicaciones en tiempo real, una solución práctica es utilizar sólo dos capas, el primer nivel sin suavizar y la imagen suavizada con $\sigma = 1,78$ o $\sigma = 3,14$. Dado que el algoritmo de casamiento de imágenes permite un cierto grado de libertad al casar segmentos, el nivel superior validaría o rechazaría los segmentos encontrados en el nivel base. El nivel superior sólo tendría que ser actualizado con una cierta periodicidad, ya que al basarse en una imagen muy difuminada, no recogería las pequeñas modificaciones que se van produciendo entre fotogramas sucesivos. El coste computacional entonces sería sólo ligeramente superior al algoritmo clásico de Canny.

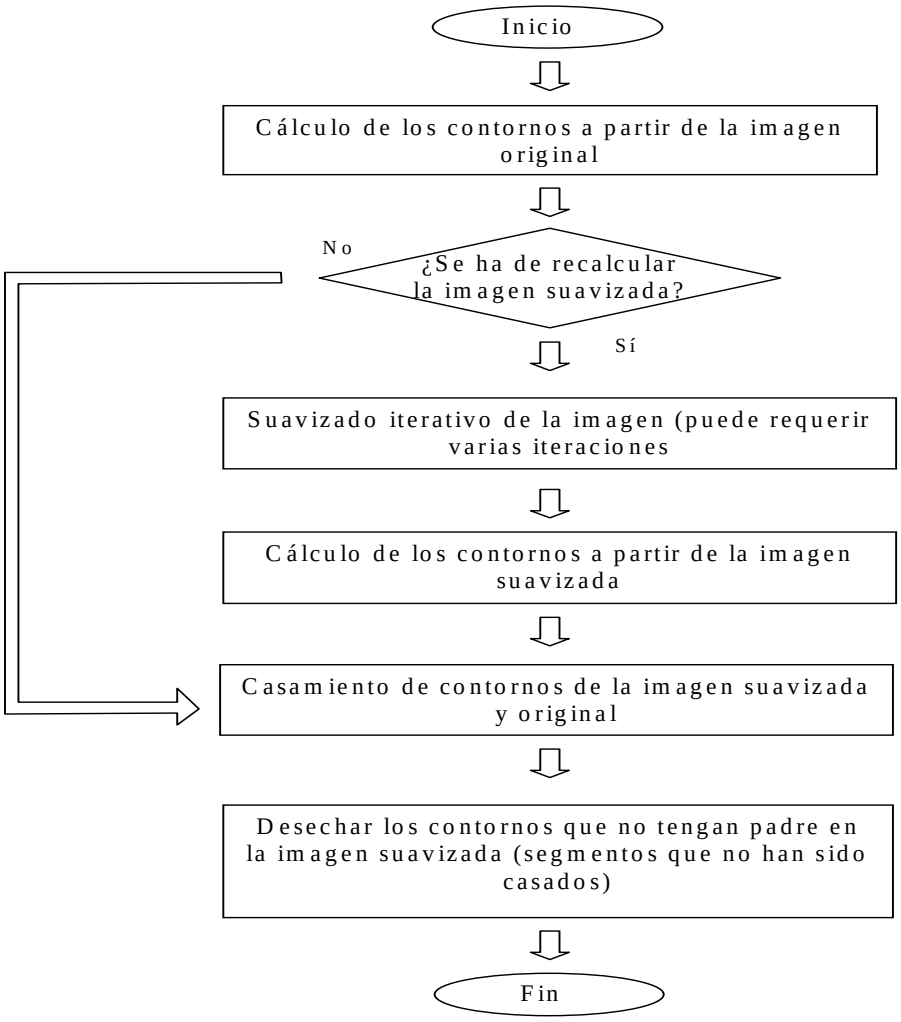


Fig. 14. Proceso multirresolución simplificado para dos niveles.

7.4 Resultados del algoritmo

A continuación se presentan varios ejemplos a los que se ha aplicado el algoritmo. En las imágenes pequeñas, el tiempo de procesado, en un ordenador Pentium 133Mhz con 32Mb de RAM, ha estado por debajo de un segundo. En las imágenes grandes y en color, como “Horsebu.jpg” y “Golf.jpg”, el tiempo de procesado se mantuvo por debajo de 8 segundos, siendo un tiempo razonable para imágenes que ocupan aproximadamadamente 1 Mb de memoria. Todas las imágenes se procesaron con seis niveles multirresolución, aunque los resultados fueron casi idénticos a los obtenidos con sólo tres niveles. Se desestimaron los contornos que no tenían padre en el nivel 3, que tenían una longitud x gradiente medio menor de 100 (este criterio da mejores resultados que imponer por separado una longitud mínima y un gradiente mínimo, ya que si no se perderían contornos tenues que son importantes por su gran

longitud). El porcentaje mínimo de coincidencias para dar dos segmentos casados ha sido del 30%.

Los niveles por superiores al mínimo –tercer nivel– no modifican la detección de contornos, sólo influyen en la caracterización de los mismos. Se ha observado que el sexto nivel no aporta ninguna información significativa, ya que la imagen está muy suavizada. El valor mínimo del gradiente conviene que sea tal que la mayoría de ruido tenga un nivel menor y no se clasifique como contorno los puntos con ruido. Si se logra eliminar el ruido del gradiente antes del cálculo de contornos, se produce una reducción del tiempo de proceso y consumo de memoria, además de vincular los segmentos de distintos niveles con mayor precisión.

Como comparativa del comportamiento del algoritmo frente al ruido en las imágenes, se ha detectado los contornos de una imagen de prueba original, corrompida por ruido gausiano de 20 dB y de 10dB. Como conclusión de dicha comparativa, se puede observar que el algoritmo multirresolución es más robusto frente al ruido que Canny, además de detectar la posición exacta de los contornos. Para obtener resultados semejantes con Canny, se tiene que aplicar un fuerte suavizado de sigma=1.0 y los contornos son desplazados de su posición original. El comportamiento frente al ruido se mejoraría notablemente al utilizar máscaras para calcular el gradiente de tamaño mayor a 2x2, tanto en Canny como utilizando la multirresolución (por esta razón, no se ha incluido los resultados).

a) b)

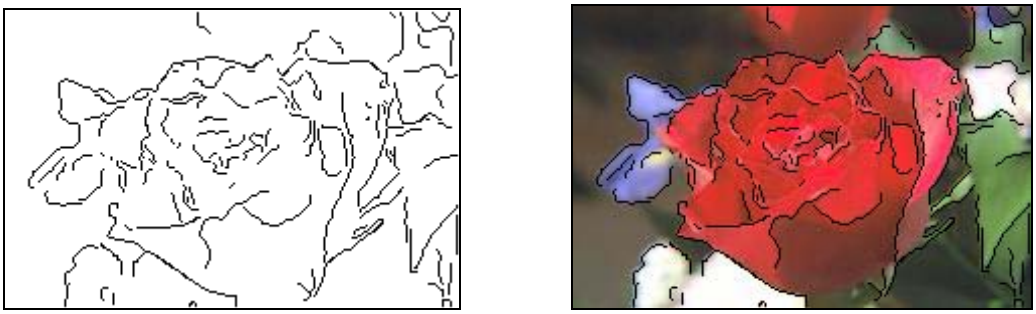
Fig. 15. Imagen Lena.gif, 256x256, 256 tonos de gris.Imagen de dominio público. a) Imagen original. b) Contornos detectados con Canny (sigma=0,2)

a) b)

Fig. 16. Imagen Lena.gif a) Contornos detectados con multirresolución b) Resultado superpuesto a la imagen original para contrastar la localización de contornos.

a) b)

Fig. 17. Imagen TestOrig.jpg, 227x149 RGB. Fuente: Independent JPEG Group. a) Imagen original b) Contornos detectados con Canny (sigma = 0.2).



a) b)

Fig. 18. Imagen TestOrig.jpg a) Contornos detectados con multirresolución b) Resultado superpuesto a la imagen original.

Fig. 19. Imagen original Horsebu.jpg, 748 x 499, color RGB. Fuente: CD-ACTUAL, Vol. 8. Shareware.

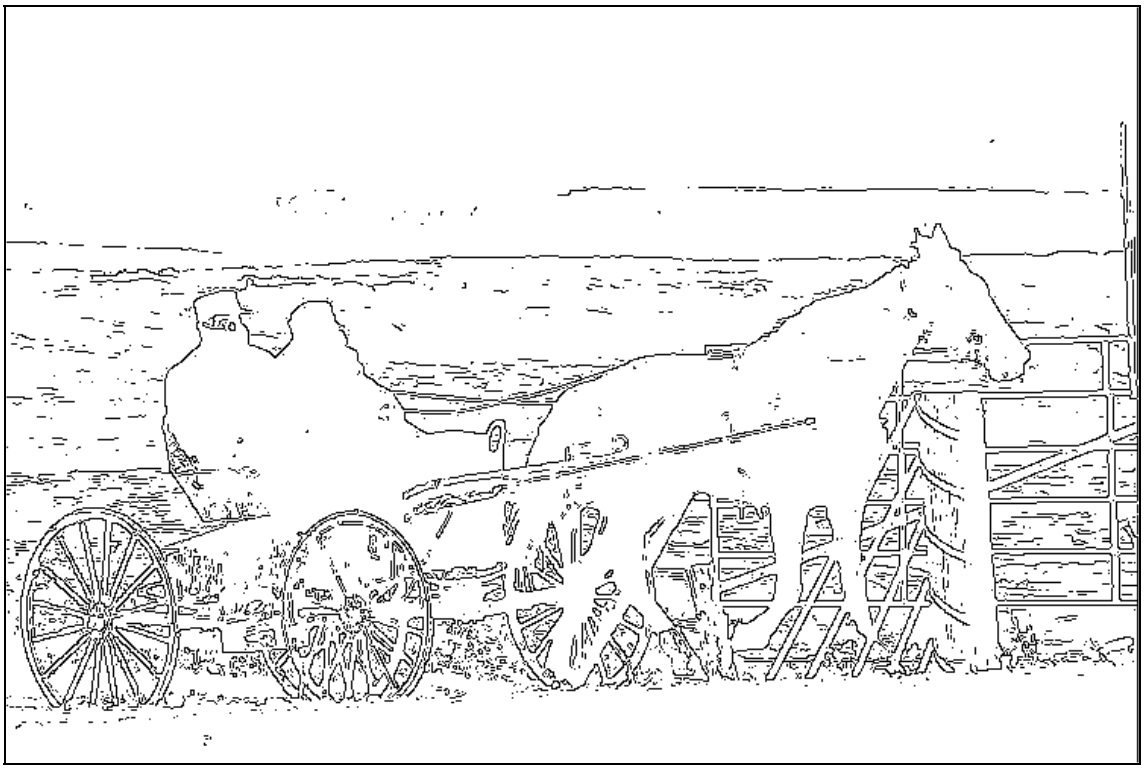


Fig. 20. Contornos detectados en Horsebu.jpg con Canny (sigma=0,2)

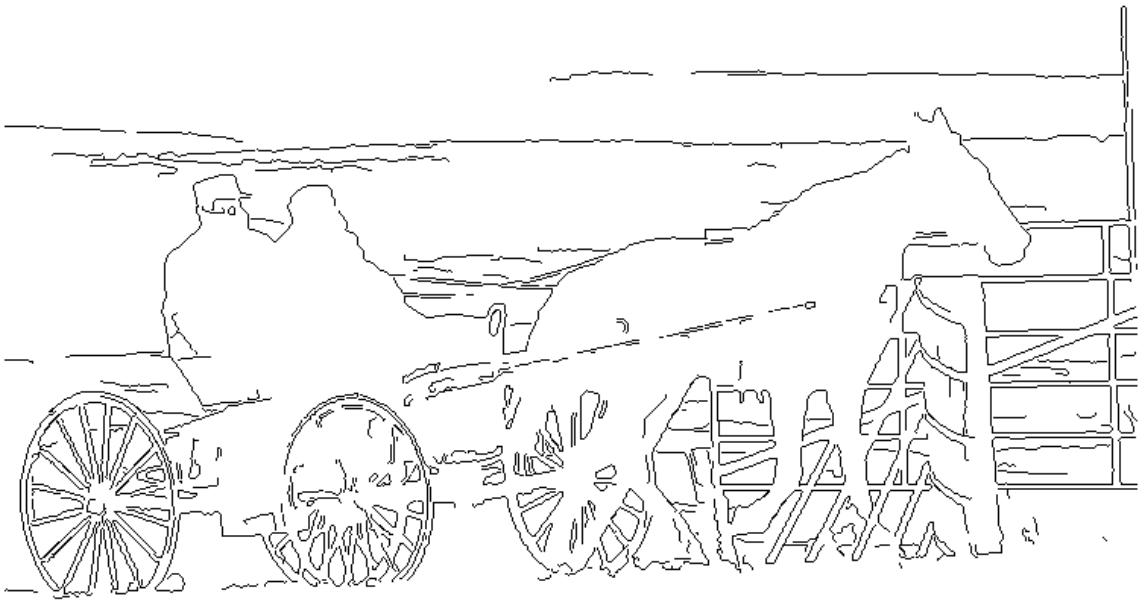


Fig. 21. Contornos detectados en Horsebu.jpg con multirresolución

Fig. 22. Imagen original Golf.jpg, 688 x 457, color RGB. Fuente: CD-ACTUAL, Vol. 8. Shareware.

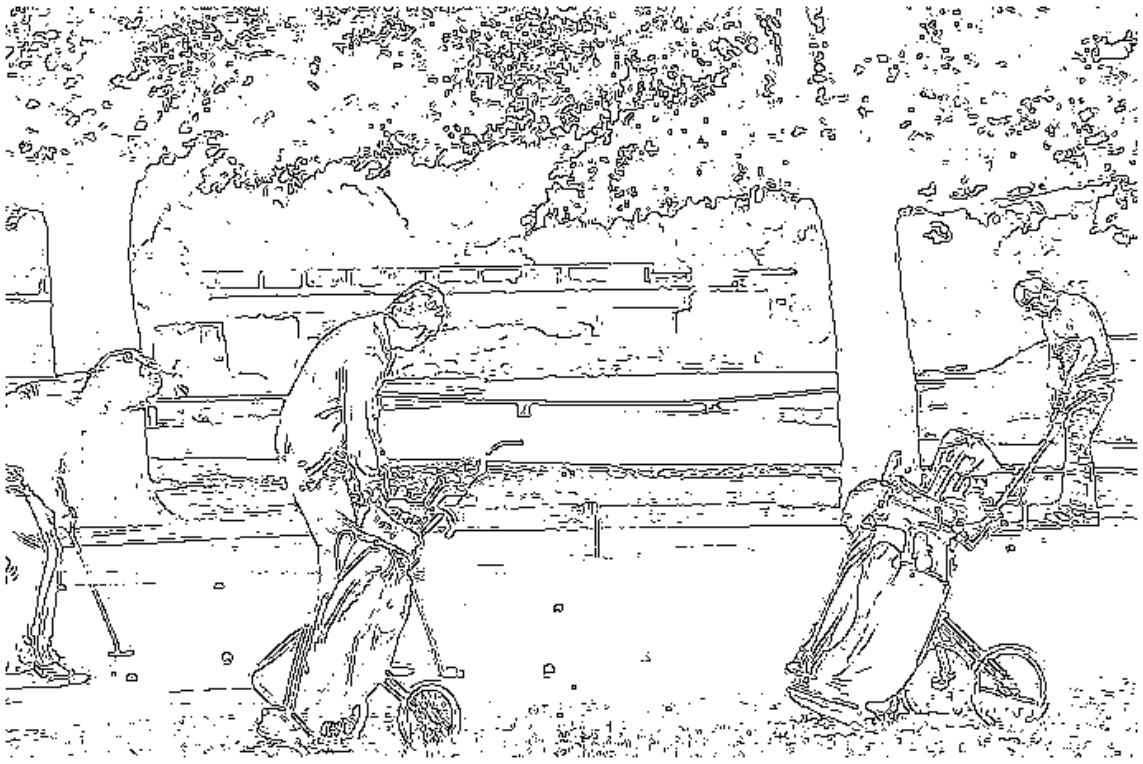


Fig. 21. Contornos detectados en Golf.jpg con Canny (sigma=0.2 y posterior umbralizado).

Fig. 21. Contornos detectados en Golf.jpg con multirresolución

7.5 Conclusiones

En imágenes contrastadas y nítidas, el algoritmo [CAN82] funciona bastante bien y la ganancia que se obtiene al aplicar el algoritmo multirresolución es reducida si se tiene en cuenta que los tiempos de procesado se triplican, de media.

Cuando las imágenes no son de calidad óptima, [CAN82] necesita aplicar un suavizado previo, produciéndose un desplazamiento de los contornos proporcional al suavizado aplicado. Sin embargo, los algoritmos multirresolución son capaces de detectar la posición correcta de los contornos. Además, en imágenes poco contrastadas, los algoritmos *multirresolución* pueden detectar contornos menos abruptos que [CAN82].

Como se puede observar en las imágenes "Horsebu.jpg" y "Golf.jpg", presentadas en este capítulo, el algoritmo multirresolución mejora notablemente los resultados obtenidos con [CAN82] cuando se trata de imágenes complejas. Además de obtener mejores resultados, los contornos están localizados sin desplazamientos y los parámetros del algoritmo multirresolución son más estables que los parámetros de [CAN82] (factor de filtrado e umbrales de detección y ruido). Por ejemplo, los ajustes para producir los contornos con el algoritmo [CAN82] fueron bastante más laboriosos que los necesarios con el algoritmo *multirresolución*, además que presentaban mucha mayor sensibilidad (un pequeño desajuste en los parámetros de [CAN82] produce unos resultados mucho más pobres).

La mejora del algoritmo *Canny Multirresolución* respecto del algoritmo del cual se ha derivado, [MAM95], es patente. Por una parte se ha acelerado el algoritmo en un factor superior a 20, y los requerimientos de memoria son inferiores. Por otra parte, la utilización del casamiento orientado a segmentos en vez de orientado a puntos mejora mucho el proceso de identificación de contornos a distintas escalas. En tercer lugar, la utilización de un estimador del gradiente de ventana 2x2 mejora la detección de contornos. En último lugar, la discriminación de los contornos según el producto longitud x valor medio del gradiente en el contorno en vez de limitarse a descartar los segmentos con longitud menor que una dada o que no superen un cierto valor medio del gradiente, se ha mostrado muy eficaz para poder detectar el ruido y eliminarlo. La clasificación de contornos que se adoptado aquí (calcular el valor medio del contorno en todos los niveles que se analizan) presenta un mejor comportamiento para la detección de contornos respecto la clasificación utilizada en [MAM95].

Los puntos del algoritmo *Canny Multirresolución* que en mi opinión tienen un mayor potencial de mejora, son la estimación del gradiente utilizando ventanas mayores (por ejemplo 4x4) que compensen el ruido presente en cada zona de la imagen, en una palabra: que sean *adaptativos*. Un paso hacia delante sería obtener los contornos cerrados de los objetos, utilizando técnicas de segmentación por crecimiento de regiones.

BIBLIOGRAFÍA

[MAM95] Miguel Angel Martínez. Análisis Multirresolución por gausianas para la detección y caracterización de contornos en imágenes 2-D. PFC. Centro Politécnico Universidad de Zaragoza. Mayo de 1995.

[JRB94] J.R. Beltrán. Detección y clasificación de contornos, en imágenes digitalizadas, utilizando la transformada de Wavelet. Tesis Doctoral. Centro Politécnico Universidad de Zaragoza. Octubre 1994.

[CAN83] J.F. Canny. Finding edges and lines. Technical Report 720. M.I.T. 1983.

[CAN86] J.F. Canny. A computational approach to edge detection. IEEE Transations on Pattern Analysis and Machine Intelligence, 8:679-698, Noviembre 1988.

[IEE92] IEEE Transactions on Information Theory. Special issue on Wavelet Transforms and Multiresolution Analysis. Marzo 1992.

[MAR82] D. Marr. Vision. W.H. Freeman, San Francisco, 1982.

[MZ92] S. G. Mallat and S. Zhong. Characterization of signal from multiscale edges. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(7): 710-732, Julio 1992.

[MAL89] S.G. Mallat. A theory for multirresolution signal descomposition: The wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(7): 674-693.

[WIS93] D.J. Williams and M. Shah. Edge characterization using normalizaed edge detector. Computer Vision Graphics and Image Processing, 5(4):311-318, Julio 1993.

[RTH71] A. Rosenfel and M. Thurston. Edge and curve detection for visual scene analysis.

[CAN76] George C. Canavos. “Probabilidad y Estadística. Aplicaciones y Métodos”. Ed. McGraw-Hill.

Capítulo 8:

Conclusiones

Las principales conclusiones que he obtenido durante la realización del proyecto son las siguientes:

— La visión artificial está en continua evolución y todo entorno de visión ha de ser capaz de ampliarse y crecer fácilmente.

— El gran abanico de formatos gráficos que existen supone un obstáculo a la hora de crear aplicaciones que puedan tratar la mayoría de las imágenes que nos podemos encontrar. Afortunadamente, hay una gran variedad de conversores gráficos que transformar los formatos menos usuales a alguno que podamos leer.

— El lenguaje de programación C++ suministra herramientas sencillas y potentes para la creación de programas profesionales. Proporciona una velocidad adecuada y permite utilizar código "C" sin apenas modificaciones.

— La biblioteca *Microsoft Fundation Classes* proporciona un método de programar el entorno *MS-Windows* de una forma más sencilla que el antiguo método *SDK* de *Microsoft*. Sin embargo, la complejidad es todavía muy elevada, sobre todo si se compara con otras herramientas como *Visual Basic* o *Borland Delphi*. La curva de aprendizaje es larga, y son necesarios al menos seis meses de estudio del entorno de programación Visual C++ para que una persona con sólidas bases de programación pueda desarrollar aplicaciones básicas.

— El diseño del entorno de visión artificial dividido en tres capas:

  - el núcleo del sistema formado por la clase `IOfile`,
  - las librerías de visión contenidas en `ImgBasic` e `Inspection` y
  - la interfaz del sistema formada por el conjunto de las clases `MFC`

  ha simplificado notablemente la aplicación.

— La estructura utilizada para almacenar imágenes en memoria se ha mostrado eficaz y versátil. El dibujo de imágenes en pantalla, para el usuario, se realiza de forma

muy sencilla y rápida, aunque ha supuesto un gran trabajo de codificación en la capa de la interfaz del sistema. La misma simplicidad que tiene el usuario a la hora de leer y grabar imágenes ha sido también a costa de trabajo extra de programación.

— La interfaz en modo texto ha sido útil durante los diseños iniciales de la aplicación, ya que ha facilitado la abstracción de la interfaz del usuario del resto del programa. Pero la utilización de la interfaz en modo texto tiene poco sentido teniendo la posibilidad de utilizar un entorno gráfico. Al final del proyecto, se ha abandonado la actualización de la plataforma de texto dada su menor importancia y el esfuerzo que supone desarrollar dos sistemas a la vez.

— Dada la amplitud de los objetivos, el número de algoritmos incluidos en este proyecto es reducido. Sin embargo, los proyectos fin de carrera de Dña. Begoña Zamora Catevilla y Dña. Gemma Benito Navaridas tienen como finalidad añadir algoritmos de inspección industrial al entorno. Según mi opinión, este proyecto es el inicio de un entorno de visión que si su uso se potencia, puede servir para realizar prácticas de visión artificial y como herramienta de trabajo en temas de tratamiento de imágenes.

— La extracción de contornos es un campo complicado y que no tiene todavía una solución general.

— Sería deseable que se implementase dentro del entorno la captura de imágenes.

— En el Área de Ingeniería de Sistemas y Automática, la visión artificial se ha enfocado, en gran medida, hacia el campo de la robótica. Sin embargo, si se utilizasen los sistemas adecuados para la interacción con personas invidentes —reconocimiento y síntesis de voz, tablas táctiles, etc—, la visión artificial también se podría encaminar a ayudar a que gente con deficiencias visuales pueda desenvolverse en la vida cotidiana con mayor facilidad.

PROYECTO FIN DE CARRERA
DEPARTAMENTO DE INFORMÁTICA
E INGENIERÍA DE SISTEMAS

DISEÑO DE UN ENTORNO DE VISIÓN PARA EL TRATAMIENTO DE IMÁGENES

Zaragoza, 14 de octubre de 1997

ANEXOS

Área de Ingeniería de Sistemas y Automática

Director: Dr. Carlos Sagüés Blázquiz.

Autor: D. Joaquín Mur Amada.

Anexo A

Funciones públicas contenidas en la clase `IOfile`

Este anexo contiene información de referencia para todas las funciones públicas de la clase IOfile. Este anexo muestra, para cada función:

- la declaración de la función
- los argumentos que toma la función
- el valor devuelto por la función
- se indicarán las funciones virtuales y virtuales públicas en la declaración (las funciones no virtuales no poseen ningún identificador específico en C++).

Las funciones se han agrupado en cuatro grupos: funciones para la lectura y escritura de imágenes en ficheros, funciones para la gestión de memoria, funciones relacionadas con el formato de las imágenes y funciones para la interacción con el usuario. Dentro de cada grupo, las imágenes están agrupadas alfabéticamente.

A.1 Lectura y escritura de una imagen en un fichero.

- ***`virtual int OpenPic ( int );`***

Esta función carga una imagen desde un fichero a memoria. Toma como argumento un entero que puede tener los valores `LOADPIC`, `RELOAD`. Los valores actualmente implementados son `LOADPIC`, que pregunta el nombre de un fichero que contiene la imagen y `RELOAD`, que vuelve a cargar la última imagen abierta (exactamente, carga el fichero cuyo nombre, extensión incluida, está en la cadena de caracteres *`fullname`*). Los valores `OP_PAGEDN` y `OP_PAGEUP` están reservados para cargar la imagen siguiente / anterior en ficheros multi–imagen como puede ser el formato GIF, pero como no es muy habitual utilizar estas características, esta opción no está desarrollada.

El valor `DFLTPIC` está reservado para cargar una imagen por defecto, que todavía no está determinada.

La función devuelve -1 en caso de fallo, liberando la memoria asignada a la imagen. En caso de éxito, devuelve 0. Si no se reconoce el formato de la imagen, se muestra el contenido del fichero en un editor de textos y se devuelve 1.

- ***`virtual int SavePic (int);`***

Esta función guarda la imagen contenida en memoria en un fichero. Toma como argumento un entero que puede tener los valores `SAVEPIC` y `REWRITEPIC`. Mientras que `SAVEPIC` pide el nombre del fichero, `REWRITEPIC` toma el nombre contenido en la cadena de caracteres *`fullname`*. El formato con el que se guarda la imagen se deduce de la extensión del archivo y de las propiedades de la imagen. En caso de que no se de ninguna extensión, se guarda la imagen en formato BMP. Para las imágenes JPEG y TIFF, antes de guardar la imagen pregunta al usuario las propiedades con las que se quiere guardar la imagen.

Las extensiones que actualmente se soportan son BMP, FTS, FITS, GIF, JFIF, JPG, JPEG, PBM, PGM, PM, PNM, PPM, RAS, RGB, TGA, TIF, TIFF, XBM, XPM, siendo indiferente si la extensión está escrita con mayúsculas o minúsculas. Si ningún formato se ajusta a la extensión, el entorno añade la extensión BMP y lo guarda como mapa de bits Microsoft Windows. La función devuelve 0 en caso de éxito, y otro valor si falla a grabar la imagen.

Cada vez que un fichero se carga en memoria o se graba, se actualiza el nombre del fichero contenido en la variable miembro de `IOfile` *`fullname`*.

A.2 Gestión de memoria

- ***`IOfile (void);`***

Este es el constructor de la clase que se llama automáticamente cada vez que se crea un objeto `IOfile`. Ajusta el tamaño de la imagen a 0 ya que no se ha cargado todavía ninguna función, y asigna los punteros a NULL. Por último inicializa todos los comentarios y mensajes contenidos en la clase con cadenas vacías.

- ***~IOfile (void);***

Es el destructor de la clase que se llama automáticamente cuando se cierra una imagen. Libera la memoria ocupada por las imágenes y comentarios contenidos en `IOfile`, de forma que la destrucción del objeto `IOfile` sea segura en cualquier situación.

- ***static int SaveMemory (void);***
  static void SaveMemory (int status);

Esta función llamada sin argumentos devuelve verdadero si ha de ahorrarse memoria (aún a costa de ralentizar las operaciones de presentación en pantalla o de inhabilitar la opción de deshacer cambios) o si se puede utilizar toda la memoria disponible. Si se le pasa un valor lógico como argumento, cambia este parámetro de la aplicación —un valor distinto de cero indica activar el ahorro de memoria—.

- ***virtual void StoreNewPic (byte *NewPic,***
  int NewPicType=PIC8, int NewMono=FALSE,
  u_int w=0, u_int h=0,
  u_int NewNumCols = 256, byte *rp=NULL,
  byte *gp=NULL, byte *bp=NULL);

Esta función instala una nueva imagen cuya dirección de inicio es *`NewPic`*, de tipo *`NewPicType`* (tipo PIC8 -imagen bicolor, en tonos de gris o con mapa de color- o PIC24 -color RGB-). Si la anchura *w* o la altura *h* es cero, las dimensiones de la nueva imagen coinciden con las de la anterior. Si no especifica el nuevo número de colores, se tomarán 256 colores si la imagen es del tipo PIC8 y 0 si la imagen es del tipo PIC24. Si no se especifica el mapa de color y la imagen es del tipo mapa de color, se mantendrá la tabla de colores anterior. Si la imagen es monocroma bicolor o en tonos de gris, la tabla de colores se ajusta automáticamente, independiente de *`rp`*, *`gp`* y *`bp`*.

En caso de que *`NewPic`* sea el puntero nulo, se libera la imagen anterior y se asigna a `IOfile` una imagen nula de 0x0 pixels de dimensión.

Si la opción de ahorrar memoria está activada, al guardar una nueva imagen se liberan los recursos ocupados por la anterior. En caso contrario, la imagen anterior se guarda para poder deshacer el último cambio o algoritmo, y se libera la penúltima

imagen. La clase `IOfile` gestiona automáticamente la opción de deshacer cambios, por lo que no se debe tomar ninguna otra precaución.

`StoreNewPic` almacena la imagen en `IOfile` pero no actualiza la presentación en pantalla. Por ello, después de almacenar la nueva imagen puede ser necesario llamar a la función *`RedrawPic`* que actualiza la imagen mostrada en la pantalla.

- ***`virtual void RestorePic (void);`***

Esta función es llamada cuando se selecciona la opción deshacer cambios del menú. Si *`SaveMem`*() es falso, restablece la imagen inmediatamente anterior al último algoritmo. Si *`SaveMem`*() es verdadero, no puede restablecer la imagen ya que la clase no guarda copias de seguridad para ahorrar memoria.

A.3 • Funciones relacionadas con el formato de las imágenes

- ***`int AskColorType(void);`***

Muestra un sencillo cuadro de diálogo en el que permite seleccionar el tipo de color que se quiere dar a la imagen (color RGB, escala de grises, blanco y negro y mapa de colores) y devuelve F_FULLCOLOR, F_GREYSCALE, F_BWDITHER, F_REDUCED o F_FULLCOLOR, respectivamente.

- ***`void ChangeImageFormat (int colType);`***

La función *`ChangeImageFormat`* cambia el formato interno de la imagen almacenada internamente en `IOfile`[1]. Toma como argumento el formato deseado.

[1] La función *`ChangeImageFormat`* sólo es necesaria llamarla si se quiere descartar el color en imágenes en color, es decir, convertir una imagen en color en monocroma para ahorrar memoria. La forma más normal de actuar es llamar a las funciones *`start24bitAlg`* y *`startGreyAlg`* para obtener la imagen en el formato deseado, en vez cambiar el formato interno de la imagen para luego utilizar directamente el puntero *`pic`*.

— Si su valor es F_FULLCOLOR, la imagen está representada por tres bytes por pixel correspondientes a la intensidad del color rojo, verde y azul (Red, Green, Blue). Los bytes se almacenan en la región de memoria apuntado por "*pic*" en orden consecutivo: R1 G1 B1 R2 G2 B2 R3 G3 B3 R4 G4 B4... , donde R1, G1 y B1 representan las intensidades del color rojo, verde y azul del primer pixel, en el rango 0 ... 255. Actualiza los valores "*picType* = PIC24", "*mono* = FALSE" y "*numcols* = 0".

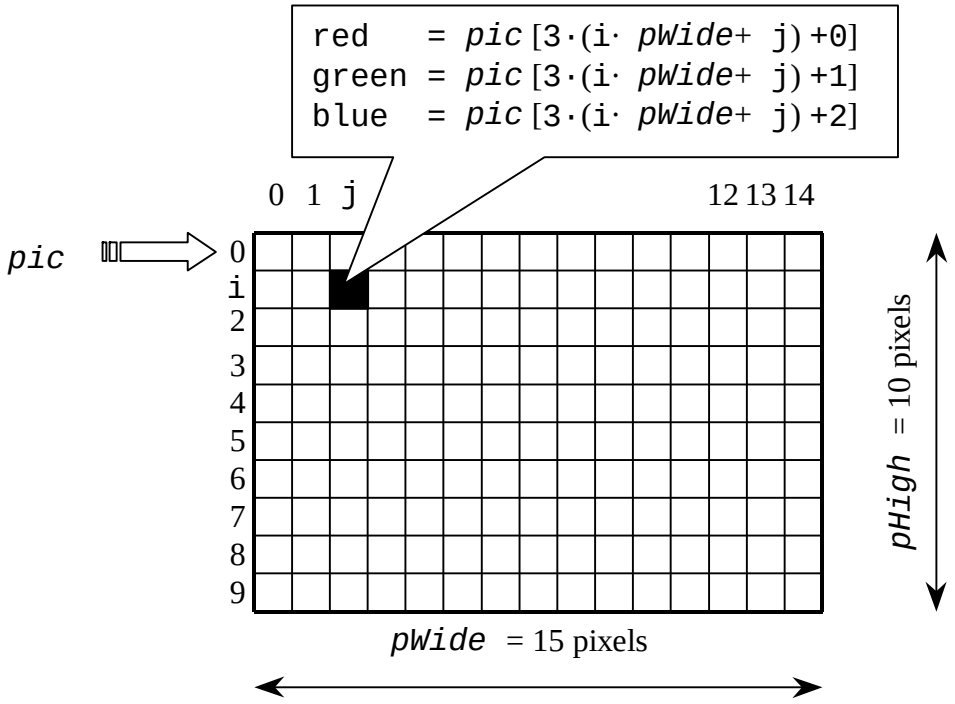


Fig. A.1: Formato de una imagen en memoria en formato F_FULLCOLOR.

— Si su valor es F_GREYSCALE, la imagen es de tonos de gris y está representada con un byte por pixel. El valor cero representa el color negro y 255 representa el color blanco. Actualiza los valores "*picType* = PIC8", "*mono* = TRUE" y "*numcols* = 256".

— Si su valor es F_BWDITHER, la imagen es bicolor monocromo y está almacenada con un byte por pixel. El valor cero representa el color negro y uno representa el color blanco. Actualiza los valores "*picType* = PIC8", "*mono* = TRUE" y "*numcols* = 2".

— Si su valor es F_REDUCED cada pixel de la imagen está almacenado como un índice (de 0 a 255) a una tabla de colores. Cada pixel ocupa un byte y el mapa de colores se ordena de mayor a menor utilización, siendo los primeros colores de la tabla los que aparecen más frecuentemente en los pixels de la imagen. Las imágenes en tonos de gris o bicolores se pueden transformar en imágenes con mapa de colores con solo ajustar el mapa. Actualiza los valores "*picType* = PIC8", "*mono* = FALSE" y calcula el número de colores distintos de la imagen en *numcols*.

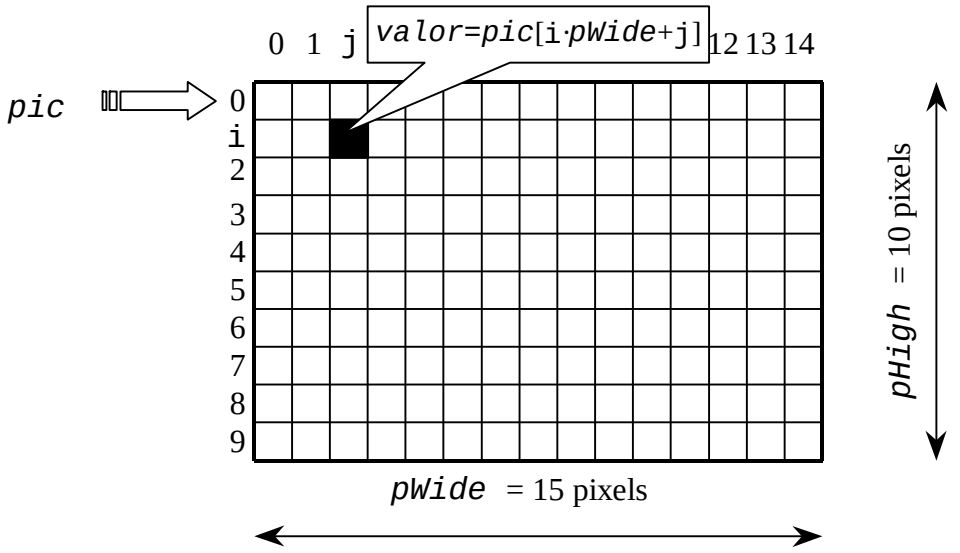


Fig. A.2: Formato de una imagen en memoria en formato F_GREYSCALE, F_BWDITHER o F_REDUCED.

- ***void changeSuffix(int fmt, int colType)***

Añade el sufijo correspondiente al formato de imagen y al tipo de color de la imagen al nombre del fichero contenido en la variable miembro *fullname*. Los formatos de imagen soportados son F_GIF, F_JPEG, F_TIFF, F_PBMRAW, F_PBMASCII, F_XBM, F_XPM, F_BMP, F_SUNRAS, F_IRIS, F_TARGA, F_FITS, F_PM, F_RAW y el tipo de color disponible es F_FULLCOLOR, F_GREYSCALE, F_BWDITHER, F_REDUCED y F_FULLCOLOR.

- ***byte *Conv24to8 (const byte *imagenRGB, int anchura, int altura, byte *RedMap, byte * GreenMap, byte * BlueMap, int NumeroColores);***

```
byte *Conv8to24    (const byte *imagenPaletaColores,
                    int anchura, int altura,
                    byte *RedMap, byte * GreenMap,
                    byte * BlueMap);
byte *Conv24toGrey (const byte *imagenRGB,
                    int anchura, int altura);
byte *ConvGreyto24 (const byte *imagenGris,
                    int anchura,int altura);
byte *Conv8toGrey  (const byte *imagenPaletaColores,
                    int anchura, int altura,
                    byte *RedMap, byte * GreenMap,
                    byte * BlueMap);
```

Esta serie de funciones transforman imagenes de tipo RGB en otras imagenes semejantes que utilicen paleta de colores o tonos de gris y viceversa. Estas funciones no modifican las imágenes contenidas en `IOfile`. Si bien esta serie de funciones podrían haberse dispuesto en una biblioteca independiente de la clase, se ha preferido incluirlas en `IOfile` para poder utilizar las funciones que muestran mensajes de error cuando falla, por ejemplo, un alojamiento de memoria.

`Conv24to8` transforma una imagen RGB en otra con paleta de colores –aproximadamente equivalente ya que al reducir el número de colores simultaneos de 16 millones a 256 se pierde información–. El proceso inverso se realiza con *`Conv8to24`*, que convierte una imagen con paleta de colores en imagen RGB. También se puede eliminar la información contenida en el color y transformar la imagen RGB en otra en tonos de gris con *`Conv24toGrey`*. También se puede descartar la información del color en una imagen en tonos de gris utilizando la función *`Conv8toGrey`* (no hay ninguna función que realice la transformación inversa porque una imagen en tonos de gris es una imagen con un mapa de colores predeterminado que contiene 256 gradaciones de gris entre el color blanco y el color negro). Si algún algoritmo sólo admite imágenes RGB, se puede convertir una imagen en tonos de gris en una en color a través de la función *`ConvGreyto24`* –las componentes roja, verde y azul de las imágenes en tonos de gris tienen en cada pixel el mismo valor–.

Los parámetros que toman estas funciones son: un puntero a la imagen en el formato original, la anchura y la altura de la imagen y el mapa de colores si el formato

original o el final lo requiere. En el caso de convertir una imagen RGB (24 bits por pixel) en una imagen con paleta de colores, se ha de especificar el número de entradas o colores distintos que contendrá la paleta de colores, hasta un máximo de 256 colores.

Si lo que se quiere es obtener la imagen contenida en la clase `IOfile` en formato RGB, con paleta de colores o en tonos de grises, es preferible utilizar la función *`GetImage`* —que a su vez llaman a alguna de estas funciones—. Otras veces puede ser ventajoso utilizar para tratar las imágenes con color[2] las funciones *`start24bitAlg`* y *`end24bitAlg`*; o *`RedComponent`*, *`GreenComponent`*, *`BlueComponent`* y *`RGBtoPIC24`*. Si sólo se quiere utilizar la luminosidad de las imágenes, las funciones *`startGreyAlg`* y *`endGreytAlg`* gestionan de forma sencilla la utilización de imágenes en tonos de gris.

- ***`void endGreyAlg (byte *OrigPic, byte *NewPic);`***
- ***`void end24bitAlg (byte *OrigPic, byte *NewPic);`***

Estas funciones actualizan la imagen contenida en `IOfile`, con la imagen apuntada por `NewPic`, en formato de escala de grises o en color RGB (24 bits por pixel), según se utilice *`startGreyAlg`* o *`start24bitAlg`*.

Estas funciones se han diseñado para ser llamadas junto con *`startGreyAlg`* o *`start24bitAlg`*, respectivamente. Los argumentos que se han de pasar a *`endGreyAlg`* o *`end24bitAlg`* son los punteros a imágenes que devuelve *`startGreyAlg`*, o *`start24bitAlg`*. *`OrigPic`* es el puntero a la imagen original y *`NewPic`* es el puntero a la nueva imagen y estos punteros no deben liberarse directamente por el usuario porque *`endGreyAlg`* y *`end24bitAlg`* ya lo hace.

Además, *`endGreyAlg`* y *`end24bitAlg`* muestran el tiempo transcurrido desde la llamada de *`startGreyAlg`* o *`start24bitAlg`*, guardan la versión anterior de la imagen para poder deshacer el algoritmo, liberan la imagen original *`OrigPic`*

[2] Todas las funciones comentadas en este párrafo realizan las conversiones internas necesarias para devolver la imagen en el formato adecuado, independientemente del formato interno con que se encuentre almacenado en `IOfile`. Por tanto, se puede obtener la imagen en formato RGB aunque contenga sólo tonos de gris, y análogamente, se puede calcular una imagen en tonos de gris a partir de una fotografía en color.

ubicada en *endGreyAlg* o *start24bitAlg* si es necesario y, por último, actualizan la imagen presentada por pantalla.

- ***virtual byte *GetImage(int *PicTypeP, int ColorType,***
  int *MustBeFreed,
  u_int *NewNumColors = NULL,
  byte **rpp = NULL,
  byte **gpp = NULL,
  byte **bpp = NULL,
  u_int MaxAllocCols = 256);

La función *GetImage* devuelve la imagen contenida en IOfile en el formato que especifican sus parámetros. Realmente devuelve un puntero a la zona de memoria que lo contiene. En caso de producirse un error dentro de la función, devuelve NULL o el puntero nulo.

El parámetro *PicTypeP* es la dirección de un entero en donde *GetImage* escribe el tipo de imagen devuelta (PIC8 o PIC24). *ColorType* especifica el formato de la imagen devuelta: F_GREYSCALE –256 tonos de gris–, F_BWDITHER –blanco y negro–, F_REDUCED –imagen con paleta de colores– o F_FULLCOLOR –imagen RGB–. *MustBeFreed* es la dirección de un entero en donde la función escribe distinto de cero si, después de utilizar la imagen devuelta por *GetImage*, ésta debe ser liberada con *free*. El resto de los parámetros son opcionales y sólo son necesarios en las imágenes en color con paleta (tipo F_REDUCED).

En *NewNumColors* se guarda el número de colores de la nueva imagen (en una imagen RGB, vale 0; en la de tonos de grises, 256, y en blanco y negro, 2). Los punteros a las tablas de color roja, verde y azul son, respectivamente, *rpp*, *gpp* y *bpp*. El máximo número de entradas de la paleta de colores se indica en *MaxAllocCols*,

Un posible esquema de un algoritmo que utilizase *GetImage* sería:

— Crear una variable para almacenar si se ha de liberar la imagen devuelta por *GetImage* después de utilizarla.

— Obtener la imagen original con *GetImage*, especificando en la llamada el formato deseado. Si el formato pedido es el utilizado internamente por IOfile se devuelve directamente el atributo *pic*. En otro caso, se devuelve una copia en el formato pedido, que se ha de liberar al final del proceso.
— Una vez comprobado que no se ha devuelto una imagen nula, ubicar la memoria necesaria para el algoritmo.
— Realizar el procesamiento.
— Una vez concluido el algoritmo, liberar con la función *free* la imagen devuelta por *GetImage*, si es necesario.
— Llamar la función *StoreNewPic* para almacenar la nueva imagen. En esta función se puede especificar completamente las dimensiones y el formato de la nueva imagen.
— Llamar a la función *RedrawPic* para que actualice la imagen mostrada en pantalla.

- ***int GetColorType(void);***

Devuelve el tipo de color (F_GREYSCALE, F_BWDITHER, F_REDUCED o F_FULLCOLOR) con que se grabaría la imagen en memoria, según las propiedades actuales y el nombre del archivo contenido en *fullname*.

- ***int GetFormatFromSuffix(void)***

Devuelve el formato de imagen según el tipo de color y la extensión del nombre del fichero, contenido en *fullname*, con que se va a guardar la imagen. Los formatos de imagen soportados son F_GIF, F_JPEG, F_TIFF, F_PBMRAW, F_PBMASCII, F_XBM, F_XPM, F_BMP, F_SUNRAS, F_IRIS, F_TARGA, F_FITS, F_PM, F_RAW y las extensiones reconocidas son *.BMP, *.DIB, *.GIF, *.IFF, *.JIF, *.JPG, *.JPEG, *.PBM, *.PGM, *.PM, *.PPM, *.PCX, *.RAS, *.RAW, *.RLE, *.TGA, *.TIFF, *.TIF, *.XWD, *.XBM y *.XPM.

- ***char *GetFullName ();***

  void SetFullName (char *st);

Estas funciones proporcionan un método para acceder a la variable miembro *fullname*, que es el nombre del fichero que se carga o se graba la imagen. *GetFullName* devuelve *fullname* y *SetFullName* copia la cadena que se le pasa como argumento en *fullname*.

- ***void hsv2rgb(double h, double s, double v,***
  int *r, int *g, int *b);
  void rgb2hsv(int r, int g, int b,
  double *h, double *s, double *v);

Estas dos funciones convierten un color expresado en el formato HSV (tono, saturación e intensidad) en el formato RGB (componentes roja, verde y azul). La función *hsv2rgb* toma como parámetros el tono *'h'*, la saturación *'s'* y la intensidad *'v'*, y devuelve por referencia las componentes roja *'r'*, verde *'g'* y azul *'b'*. La función *rgb2hsv* toma como parámetros las componentes roja *'r'*, verde *'g'* y azul *'b'*, y devuelve por referencia parámetros el tono *'h'*, la saturación *'s'* y la intensidad *'v'*. Las componentes RGB (*'r'*, *'g'* y *'b'*) tienen el rango 0 a 255, el tono *'h'* está en el rango 0.0 a 360.0 (aunque la constante "*NOHUE*" o -1.0 se utiliza para expresar la ausencia de tonalidad), la saturación *'s'* y la intensidad *'v'* varían entre 0.0 y 1.0.

Además, en el fichero "*miscelanea.cpp*" se encuentran sencillos algoritmos que pueden ser útiles: *xvbzero* rellena con el valor 0 *n* bytes; *xvbcmp* compara dos regiones de bytes; *xvbcopy* copia una zona de memoria a otra. Con la macro *MONO*, se puede convertir las componentes RGB en tonos de gris.

- ***void RedComponent (byte* component, const byte *pic24 = NULL,***
  u_int wide = 0, u_int high = 0);
  void GreenComponent (byte* component, const byte *pic24 = NULL,
  u_int wide = 0, u_int high = 0);
  void BlueComponent (byte* component, const byte *pic24 = NULL,
  u_int wide = 0, u_int high = 0);

Esta serie de funciones extraen la componente roja, verde o azul de la imagen RGB *pic24* de anchura *wide* y altura *high*, tal como se muestra en la figura A.3. *RedComponen* copia la componente roja, *GreenComponen* copia la componente verde y *BlueComponen* copia la componente azul, en una zona de memoria de *wide* * *high* bytes apuntada por *component*. Antes de llamar a esta función se ha de requerir la memoria para copiar la componente, por ejemplo, con "*malloc*(*wide*high*)" —las funciones *RedComponen*, *GreenComponent* y *BlueComponent* no alojan memoria para guardar la componente del color sino que suponen que la memoria ya ha sido ubicada con antelación—.

El parámetro opcional *pic24* permite utilizar una imagen original distinta de la contenida en IOfile: si sólo se especifica el primer parámetro, se utiliza la imagen contenida en IOfile con sus propias dimensiones *pWIDE* y *pHIGH* en vez de *wide*

y *high*. Si se quiere utilizar como imagen original otra distinta a la contenida en IOfile, se utilizará la imagen en formato RGB apuntada por *pic24*, con anchura *wide* y altura *high*.

RedComponen, *GreenComponent* y *BlueComponent* proporcionan una forma alternativa para tratar imágenes de color con *start24bitAlg* y *end24bitAlg*, adecuada cuando se necesita tratar cada componente de color por separado.

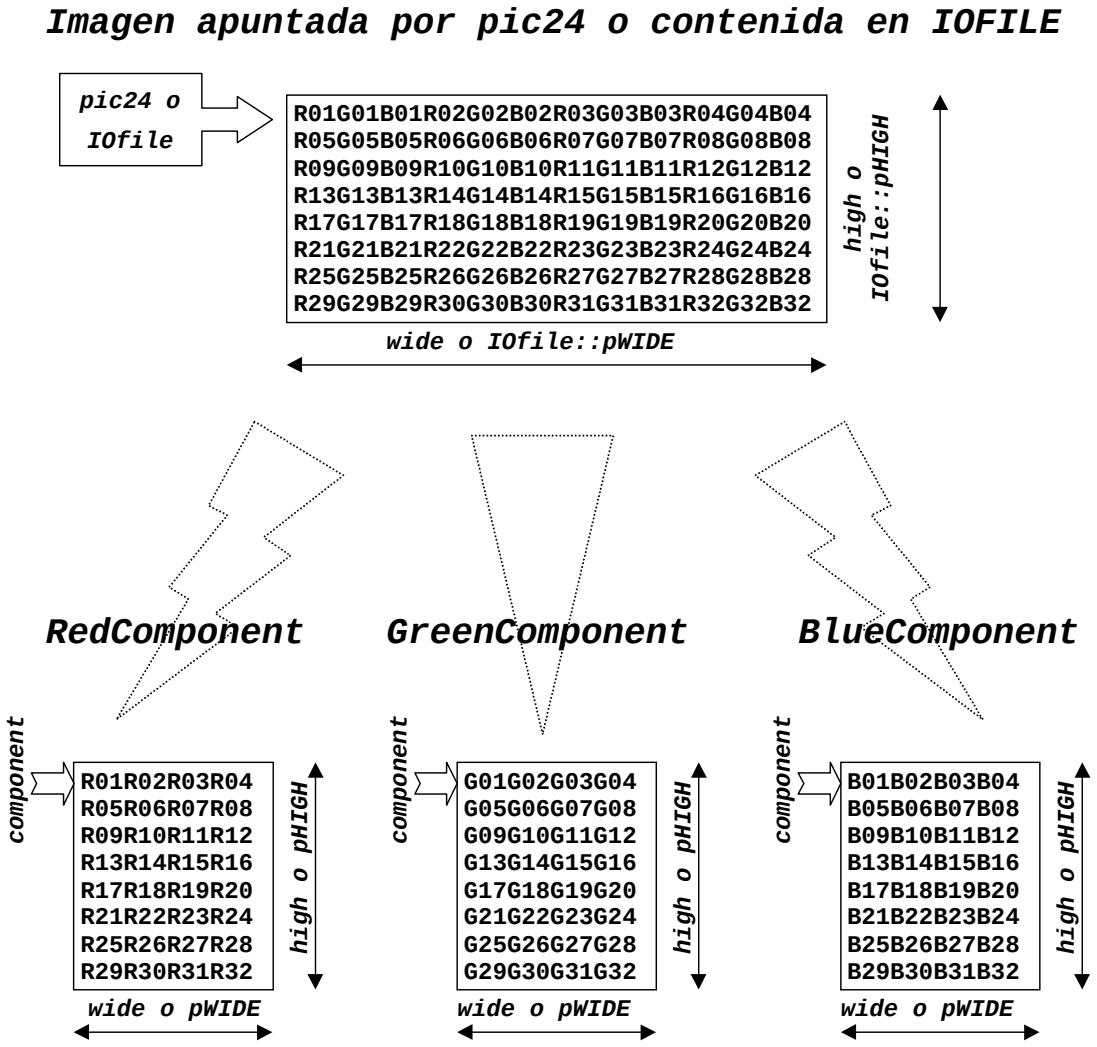


Fig. A.3: Separación de las componentes de color a partir de una imagen RGB.

RedComponent, *GreenComponen* y *BlueComponent* han sido diseñadas para trabajar conjuntamente con *RGBtoPIC24*, y su utilidad es máxima cuando se quiere tratar por separado cada componente del color. Para ilustar un caso práctico, supóngase que *w* es la anchura y *h* la altura de la imagen y que disponemos una función *doGaussianConvGrey* que realiza un suavizado gausiano, de valor *sigma*, sobre imágenes en tonos de gris o sobre componentes del color. En la variable *component*

se guardará temporalmente las componentes de color de las imágenes originales, y en *Red*, *Blue* y *Green* las componentes suavizadas. En el fragmento de código se ha suprimido las comprobaciones necesarias cuando se realiza una petición de memoria.

```
byte *component, *Red, *Green, *Blue;
int w=IOfile::pWIDE, h=IOfile::pHIGH;
int error;

  /* ALLOCATE MEMORY */
  component = (byte*) malloc( w*h );
  Red = (byte*) malloc( w*h );
  Green = (byte*) malloc(w*h );
  Blue = (byte*) malloc( w*h );

  /* BLUR RED COMPONENT! */
  RedComponent(component);
  doGaussianConvGrey(component, Red, w, h, sigma);

  /* BLUR GREEN COMPONENT! */
  GreenComponent(component);
  doGaussianConvGrey(component, Green, w, h, sigma);

  /* BLUR BLUE COMPONENT! */
  BlueComponent(component);
  doGaussianConvGrey(component, Blue, w, h, sigma);

  /* SET NEW IMAGE */
  error=RGBtoPIC24(Red, Green, Blue);
  if (error) printf("Se ha producido un error");

  /* FREE MEMORY */
  free(component);
  free(Red);
  free(Green);
  free(Blue);
```

- ***virtual void RedrawPic (void) = 0;***

Realiza las operaciones necesarias para que la imagen en pantalla se actualice con la contenida en memoria (en la versión modo texto sólo saca un mensaje por pantalla para indicar que la imagen ha cambiado). Esta función siempre es llamada, directa o indirectamente, después de cualquier modificación de la imagen. El caso típico en el que será necesario llamar a la función directamente es cuando se modifican directamente las variables miembro de la clase o se utiliza la función *StoreNewPic* para almacenar una imagen.

- ***int RGBtoPIC24 (const byte* Red, const byte* Green, const byte* Blue, u_int wide = 0, u_int high = 0);***

Esta función almacena en la clase `IOfile` la imagen RGB equivalente a las componentes roja, verde y azul, de anchura `wide` y altura `high` (si no se especifican

wide y *high* o son cero, se toman las dimensiones *pWIDE* y *pHIGH* de `IOfile`). En la figura A.4 se puede observar de forma gráfica cómo se forma una imagen RGB a partir de sus componentes.

RGBtoPIC24 devuelve 1 en caso de error y 0 si ha podido actualizar la imagen contenida en `IOfile`.

RGBtoPIC24 toma como argumentos las componentes roja (*red*), verde (*green*) y azul (*blue*) de la nueva imagen. Los parámetros opcionales wide y high permiten variar las dimensiones de `IOfile`. Además guarda la versión anterior de la imagen para poder deshacer el algoritmo y actualiza la imagen mostrada en pantalla.

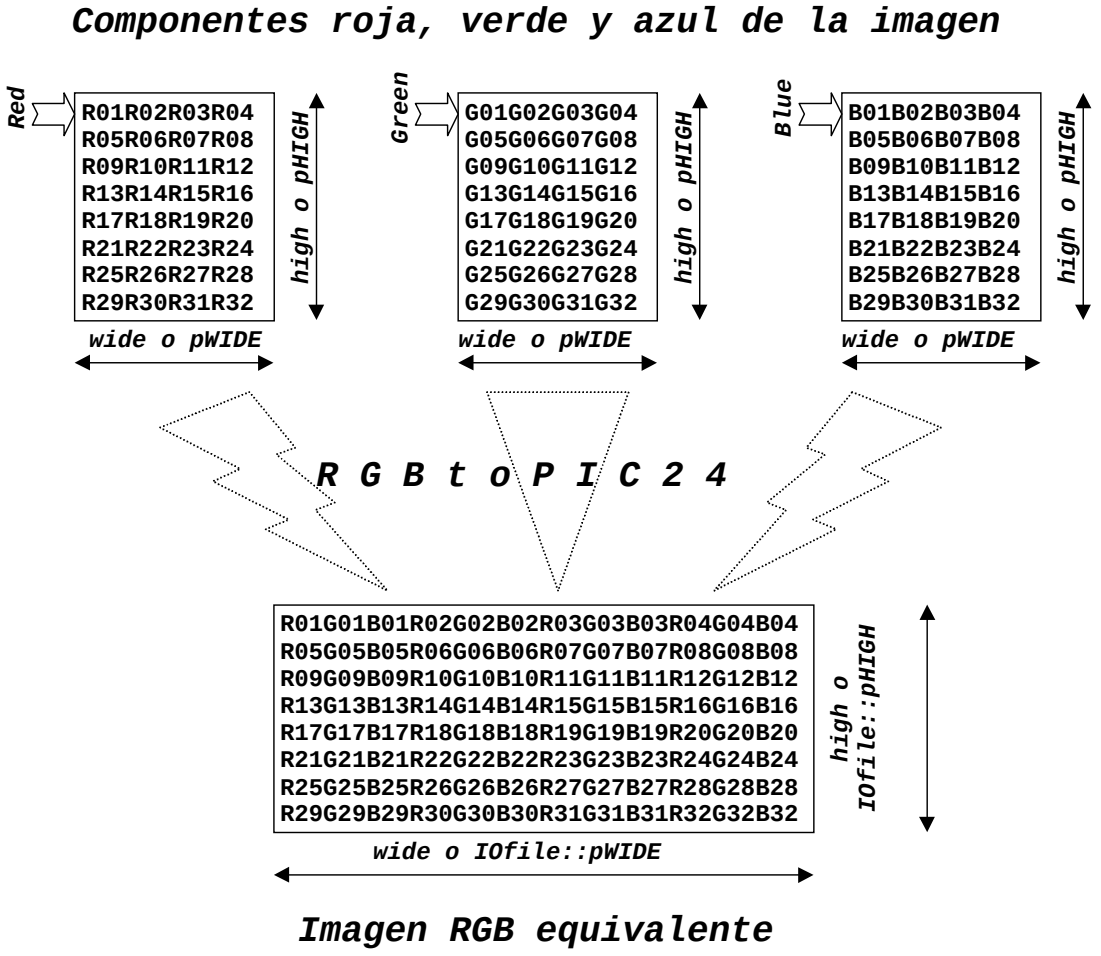


Fig. A.4: Entrelazado de una imagen RGB a partir de las componentes de color.

- ***int start24bitAlg (byte **, byte **);***
- ***int startGreyAlg (byte **, byte **);***

start24bitAlg y *startGreyAlg* devuelven, a través de las direcciones de los punteros que se les pasan, la imagen original contenida en `IOfile` convertida al formato RGB y 256 grises, respectivamente, y una imagen de las mismas

características, inicializada a cero. El valor de retorno es 0 en caso de éxito, y 1 en otro caso.

start24bitAlg proporciona una versión de la imagen RGB de 24 y una zona de memoria rellenada con ceros, del mismo tamaño – 3·*pHIGH·pWIDE* bytes–, para almacenar el resultado del algoritmo. Si el formato actual de la imagen es RGB ("*picType*=PIC24"), la imagen devuelta es la misma que la apuntada por *pic*, por lo que ésta no debe ser modificada.

startGreyAlg devuelve la imagen con 256 tonos de gris y una zona de memoria del mismo tamaño –pHIGH·pWIDE bytes–, rellenada con ceros, para almacenar el resultado del algoritmo. Si el formato actual de la imagen es el de tonos de gris ("*picType* = PIC8", "*mono* = TRUE" y "*numcols* = 256"), la imagen en tonos de gris devuelta es la misma que la apuntada por *pic*, por lo que ésta no debe ser modificada.

Estas funciones se han diseñado para utilizarse juntamente con *end24bitAlg* y *endGreyAlg.*

A.4 Funciones de la interfaz del usuario

En la clase IOfile se han incluido las funciones necesarias para la interacción del usuario con el programa. Todas las funciones del interfaz del usuario son virtuales, y la mayoría virtuales puras.

En el diseño de la clase IOfile se ha buscado mantener la independencia respecto de la interfaz del usuario. Para poder utilizar las funciones de la interfaz, es necesario tenerlas definirlas en esta clase. Debido a que la forma de interactuar con el usuario depende del tipo de plataforma escogido, el código de estas funciones no se puede concretar en la clase IOfile. La solución que se ha adoptado es crear unas funciones virtuales puras para interaccionar con el usuario. Puesto que estas funciones forman parte de la clase IOfile, se tiene acceso a ellas y dado que son funciones virtuales puras, no tienen ningún código asociado en la clase IOfile.

En las clases derivadas de `IOfile` es en dónde se encuentra su implementación, según la plataforma para la cual se compile el entorno de visión (terminal de texto, MS-Windows, *X-Windows* o *Motif*, etc.). Se debe dar código al menos en una clase derivada de `IOfile` (por ejemplo, en las clases derivadas de `IOfile` para la plataforma de texto o *MS-Windows*). Al ser funciones virtuales, el código que ejecutan dichas funciones es el correspondiente a la última clase derivada que ha sobrecargado la función.

- ***virtual int AbortRetryIgnore(char *str) = 0;***

Esta función muestra la cadena str en un cuadro de diálogo con los botones “Abortar”, “Reintentar” y “Cancelar”. Devuelve `IDABORT`, `IDRETRY` o `IDCANCEL`, dependiendo del botón pulsado.

- ***virtual void ArrowCursor (void) = 0;***

Devuelve el cursor del ratón a su estado normal (normalmente, una flecha blanca con el borde negro).

- ***virtual void CloseAlert (void) = 0;***

Destruye un mensaje de alerta creado con *`OpenAlert`*.

- ***void CropRect2Rect(int *xp, int *yp, int *wp, int hp, int cx, int cy, int cw, int ch)***

Recorta el rectángulo formado con la esquina superior derecha (*`xp`*, *`yp`*), anchura *`wp`* y altura *`hp`*, de forma que esté contenido dentro del rectángulo limitado por (*`cx`*, *`cy`*), anchura *`cw`* y altura *`ch`*.

- ***virtual void drawStrings (void) = 0;***

Muestra por pantalla información sobre la imagen contenida en el vector de cadenas *`istrs`* (esta información proviene en su mayor parte de la carga de la imagen desde fichero).

- ***void ElapsedTime(char* str, clock_t start);***

A veces es necesario medir cuanto tiempo de procesamiento consume un algoritmo. En un entorno multitarea ésto no es fácil tener el dato exacto, pero se puede tener una idea calculando el tiempo transcurrido desde el inicio del algoritmo hasta el final. Para tal menester se ha diseñado esta función. Esta función muestra en la ventana "Output" el tiempo transcurrido desde el instante indicado por *start* hasta el inicio de la funcion *ElapsedTime*. El esquema típico de utilización de esta función sería:

```
clock_t inicioAlgoritmo = clock(); // Almacena el instante actual
// AQUI SE REALIZA LAS OPERACIONES QUE SE QUIEREN CRONOMETRAR
ElapsedTime("NombreAlgoritmo", inicioAlgoritmo);
```

que generaría una salida:

```
Tiempo transcurrido en "NombreAlgoritmo": 17.2 segundos
```

en donde se ha indicado con negrita las partes que varían en cada llamada.

- ***virtual void Error (char *) = 0;***

Muestra un mensaje de error por pantalla. Toma como argumento una cadena con el mensaje. Esta función sólo ha de usarse cuando se ha producido un error, pero que no comprometa la estabilidad del programa. Si el error es muy severo, debe utilizarse *FatalError*, que después de mostrar el mensaje cerrará la aplicación para evitar males mayores.

- ***virtual void FatalError (char*) = 0;***

Muestra un mensaje de error por pantalla. Toma como argumento una cadena con el mensaje. Esta función sólo ha de usarse cuando se ha producido un error de tal magnitud que la aplicación ha de cerrarse de forma inmediata.

- ***void FilterString(char *buf, char *gsFilter, int allow);***

Filtra una cadena de caracteres, suprimiendo de la misma los códigos especiales y eliminando los caracteres que no aparecen en el filtro *gsFilter* si *allow* es distinto de cero. Si *allow* es cero, los caracteres que no son códigos especiales (ESC, '\n', DEL, '^A','^B', etc.) y no están presentes en el filtro, se sustituyen por un espacio.

- ***char *GetISTR (int) ;***

Devuelve la cadena que antes se ha almacenado con *SetISTR*.

- ***virtual void GetSelRCoords(int *xp, int *yp, int *wp, int *hp)***

Devuelve la parte de la imagen x, y, w, h en coordenadas de la imagen, seleccionada por el usuario. Nota: En el estado actual del programa, no se ha implementado la selección de parte de una imagen y por lo tanto esta función siempre devuelve los valores correspondientes a la imagen entera.

- ***virtual int HaveSelection (void);***

Devuelve un valor distinto de cero si el usuario ha seleccionado parte de la imagen. Nota: En el estado actual del programa, no se ha implementado la selección de parte de una imagen y por lo tanto esta función siempre 0.

- ***virtual int InputDialog (char *txt, char *buf, int buflen, char *filstr, int allow) = 0;***

Muestra un cuadro de diálogo para que el usuario introduzca una cadena de texto. El comentario que aparece en el cuadro, pidiendo al usuario que introduzca tal parámetro es *txt*. La cadena introducida por el usuario se guarda en *buf* (el contenido inicial es el texto por defecto del cuadro de edición). Esta función admite filtrar la cadena introducida por el usuario, eliminando los caracteres que aparecen en la cadena *filstr* si *allow* es distinto de cero. Para más información sobre el filtrado, ver *FilterString*.

- ***virtual void JPEGDialog (void) = 0;***

Muestra un diálogo con las opciones para guardar imágenes en formato JPEG.

- ***virtual int OkCancel (char *str) = 0;***

Esta función muestra la cadena str en un cuadro de diálogo con los botones “Aceptar” y “Cancelar”. Devuelve `IDOK` o `IDCANCEL`, dependiendo del botón pulsado.

- ***virtual void OpenAlert (char *) = 0;***

Esta función crea una ventana y muestra un mensaje de alerta, con el texto que se indica, que será visible hasta que se llame a *CloseAlert*.

- ***virtual int PopUp (char *, char **, int) = 0;***

Muestra un menú de botones donde se puede escoger una de las posibles opciones. Toma como argumentos una cadena que contiene el mensaje a mostrar, y un vector de cadenas de caracteres con el texto de los botones y el número de botones. Devuelve el número del botón apretado, contando desde cero hasta (n-1). En la versión actual, esta función no muestra en pantalla los botones, sino que crea un cuadro de diálogo más sencillo.

- ***virtual int PopMenu(char *, char **, int) = 0;***

La única diferencia de esta función con *PopUp* es que *PopMenu* está pensado para poner una descripción larga en cada botón, en forma de menú de botones de radio. Devuelve el número de la opción elegida, contando desde cero hasta (n-1).). En la versión actual, esta función utiliza el mismo cuadro de diálogo que *PopUp*.

- ***virtual int printf (const char *format, ...) = 0;***

Esta función se ha incluido en la clase `IOfile` ya que la plataforma MS-Windows no proporciona una manera sencilla de mostrar mensajes de texto en una ventana deslizable. Hay veces que se quiere proporcionar al usuario información para que la pueda analizar, pero no se quiere que cada vez que se muestre un mensaje el usuario tenga que confirmarlo, por ejemplo, pulsando un botón.

En plataformas UNIX, la función *printf* envía todos los mensajes a la ventana de terminal desde donde se lanzó el programa, lo cual es una característica deseable, ya que el usuario puede comprobar en cualquier momento los mensajes enviados por la aplicación. Sin embargo, la plataforma MS-Windows carece de esta característica, e incluso los programas para este entorno parecen no obedecer a las redirecciones del flujo estándar de entrada y salida.

Por todo esto, se ha diseñado la función miembro homónima de la función estándar de C *printf*. Los parámetros de la función miembro son idénticos a los que acepta la función de C, cuyas especificaciones se pueden encontrar en manual de

referencia de un compilador de lenguaje C. En modo texto, no hay ninguna diferencia entre la función miembro y la estándar de C. En modo gráfico, se muestra la misma cadena que produciría la función de C, pero en una ventana de texto creada para tal efecto. En caso que no haya ninguna ventana de texto creada dentro del entorno de visión, se crea una con el título "Output".

- ***virtual void ProgressMeter(int min, int max, int val, char *str);***

Esta función es llamada durante largas operaciones (algoritmos, suavizado, etc.) para indicar cuando va a terminar el programa. Dibuja una barra que se va llenando conforme la operación se va completando. Con `min` y `max` se ajusta el rango de la barra de progreso. Si "*`val == min`*", se dibuja la barra vacía. Si la función es llamada con "*`val == max`*", se elimina la barra. Para otros valores de *`val`* contenidos entre *`min`* y *`max`*, se dibuja la barra proporcionalmente llena. Trata de ser inteligente: sólo dibuja la barra si la operación va a durar algo más de unos pocos segundos. Actualmente, en la versión gráfica la cadena `str` no se muestra junto con la barra de progreso por un problema de implementación. En la versión de texto, aparece un mensaje en la pantalla que indica el porcentaje completado del algoritmo.

- ***virtual int RetryCancel (char *str) = 0;***

Esta función muestra la cadena str en un cuadro de diálogo con los botones "Reintentar" y "Cancelar". Devuelve `IDRETRY` o `IDCANCEL`, dependiendo del botón pulsado.

- ***virtual int scanf (const char *format, ...) = 0;***

Esta función se ha incluido en la clase `IOfile` para poder introducir datos al programa de forma sencilla en la plataforma MS-Windows. Una alternativa a crear un cuadro de diálogo cada vez que se quiera introducir un parámetro es requerir que el usuario introduzca los datos en una ventana de texto.

En plataformas UNIX, la función *`scanf`* espera que el usuario introduzca los datos en la ventana de terminal desde donde se lanzó el programa. Sin embargo, la plataforma MS-Windows carece de esta característica, e incluso los programas para este entorno parecen no obedecer a las redirecciones del flujo estándar de entrada y salida.

Por todo esto, se ha diseñado la función miembro homónima de la función estándar de C *scanf*. Los parámetros de la función miembro son idénticos a los que acepta la función de C, cuyas especificaciones se pueden encontrar en manual de referencia de un compilador de lenguaje C. En modo texto, no hay ninguna diferencia entre la función miembro y la estándar de C. En modo gráfico, se activa una ventana de texto. En caso que no haya ninguna ventana de texto creada dentro del entorno de visión, se crea una con el título "Output".

- ***void SetISTR (int stnum, ...) ;***

Modifica el valor de las cadenas contenidas en *istrs*. El primer parámetro es el número de la cadena a modificar, y los valores que puede tomar son `ISTR_INFO`, que contiene información de la última acción ejecutada, `ISTR_WARNING`, con información del último aviso, `ISTR_FILENAME` con el nombre del último fichero cargado o guardado, `ISTR_FORMAT` con el tipo de formato de la imagen en fichero, `ISTR_RES` con información de la resolución, `ISTR_CROP` con información si la imagen está recortada, `ISTR_EXPAND` con información si la imagen está expandida, `ISTR_SELECT` con información si la imagen está seleccionada y `ISTR_COLOR` con información sobre el color de la imagen. Actualmente, esta información se suele modificar cuando se carga una imagen o cuando ocurre algún aviso. El resto de los parámetros tienen la misma sintaxis de la función *printf*.

- ***void ShowImage(byte *image, int ColorType=F_GREYSCALE,***
  char* message = NULL, int w=-1,
  int h=-1, byte *rp=NULL, byte *gp=NULL,
  byte *bp=NULL);

Muestra una imagen temporal por pantalla, sin afectar a las imágenes contenidas en la clase `IOfile`. Esto es útil para ver el proceso en algunos algoritmos paso a paso y ayuda a depurar los algoritmos.

Si se especifica la cadena *message*, se muestra un cuadro de diálogo con ese texto y hasta que no se aprieta el botón aceptar no se restablece la imagen. Si no se especifica la cadena o es cadena nula, se muestra la imagen temporal hasta que se llama

a la función *RedrawPic* (por ejemplo, al terminar el algoritmo siempre se suele llamar a esta función directamente o a través de *end24bitAlg* o *endGreyAlg*).

Esta función instala una nueva imagen cuya dirección de inicio es *image*. *ColorType* especifica el formato de la imagen temporal: F_GREYSCALE –256 tonos de gris–, F_BWDITHER –blanco y negro–, F_REDUCED –imagen con paleta de colores– o F_FULLCOLOR –imagen RGB–. Si la anchura *w* o la altura *h* es -1, las dimensiones de la imagen temporal se toman de la imagen contenida en IOfile. Si la imagen es de tipo F_REDUCED y no especifica el mapa de color en *rp*, *gp* y *bp*, se mantendrá la tabla de colores contenida en IOfile. Si la imagen es monocroma bicolor, en tonos de gris o en color RGB, la tabla de colores no se considera.

- ***void TextView (char *);***

Muestra el contenido de un fichero en modo texto. En la versión de texto, simplemente vuelca el contenido del fichero en pantalla. En la versión gráfica, crea una ventana nueva en donde muestra el fichero, al estilo del bloc de notas u otro editor. Si el fichero contiene caracteres nulos o de control, como es el caso de los ficheros binarios, el editor puede no ser capaz de abrir el fichero, en cuyo caso el programa lo indica con un cuadro de diálogo. En el entorno gráfico, esta función tampoco es capaz de mostrar ficheros de más de 64K de tamaño.

- ***virtual void TIFFDialog (void) = 0;***

Muestra un diálogo con las opciones para guardar imágenes en formato TIFF.

- ***virtual void WaitCursor (void) = 0;***

Esta función cambia el cursor del ratón de su estado actual a uno que tiene forma de reloj.

- ***virtual void Warning (char *str = NULL) = 0;***

Muestra en pantalla el mensaje que se le pasa como argumento si existe o el contenido de *istr[*ISTR_WARNING*]* si no se especifica ningún argumento. Esta función ha de utilizarse cuando se informa al usuario de un aviso que no constituye un error. Para indicar errores, se debe utilizar *Error*.

- ***virtual int YesNo (char *str) = 0;***

Esta función muestra la cadena str en un cuadro de diálogo con los botones “Sí” y “No”. Devuelve `IDYES` o `IDNO`, dependiendo del botón pulsado.

- ***virtual int YesNoCancel (char *str) = 0;***

Esta función muestra la cadena str en un cuadro de diálogo con los botones “Sí”, “No” y “Cancelar”. Devuelve `IDYES`, `IDNO` o `IDCANCEL`, dependiendo del botón pulsado.

BIBLIOGRAFÍA

[MON89] Montano L., Neira J. Sagüés C., Silva M., Tardós J.D. "Introducción a la robótica Industrial". C.P.S.U.Z, 1989.

[MAR82] D. Marr. "Vision". W.H. Freeman, San Francisco, 1982.

[ROB88] K.S. Fu, R.C. González, C.S.G. Lee. "Robótica: Control, detección, visión e inteligencia". Mc. Graw Hill, 1988.

[BRA82] Brady M. "Computational Approaches to Image Understanding". ACM Computing Surveys, Vl 14, No. 1, pp 3-71, March 1982.

[DUD73] Duda R.O., Hart P.E. "Pattern Classification and Scene Analysis". Wiley, New York, 1973.

[FU82] Fu K.S. "Sintactic Pattern Recognition and Applications". Prentice Hall, Englewood Cliffs, N.J., 1982.

[GON77] Gonzalez R.C., Wintz P. "Digital Image Processing". Addison-Wesley, Reading, Mass. 1977.

[HOR86] Horn B.K.P. "Robot Vision". The MIT Press, Cambridge, Mass., 1986.

[PAV77] Pavlidis T. "Structural Pattern Recognition. Srpinger, Berlin, 1977.

[SHI85] Shimon Y.N. (ed.) "Handbook of Industrial Robotics". John Wiley & Sons Inc., New York, 1985.

ÍNDICE TEMÁTICO DEL ANEXO A

FUNCIONES PÚBLICAS CONTENIDAS EN LA CLASE IOFILE _____ 1

A.1 Lectura y escritura de una imagen en un fichero. _____ 1

OpenPic _____ 1
SavePic _____ 2

A.2 Gestión de memoria _____ 2

IOfile _____ 2
~IOfile _____ 3
SaveMemory _____ 3
SaveMemory _____ 3
StoreNewPic _____ 3
RestorePic _____ 4

A.3 Funciones relacionadas con el formato de las imágenes _____ 4

AskColorType _____ 4
ChangeImageFormat _____ 4
changeSuffix _____ 6
Conv24to8 _____ 7
Conv8to24 _____ 7
Conv24toGrey _____ 7
ConvGreyto24 _____ 7
Conv8toGrey _____ 7
endGreyAlg _____ 8
end24bitAlg _____ 8
GetImage _____ 9
GetColorType _____ 10
GetFormatFromSuffix _____ 10
GetFullName _____ 11
SetFullName _____ 11
hsv2rgb _____ 11
rgb2hsv _____ 11
RedComponent _____ 11
GreenComponent _____ 11
BlueComponent _____ 11
RedrawPic _____ 13
RGBtoPIC24 _____ 14
start24bitAlg _____ 15
startGreyAlg _____ 15

A.4 Funciones de la interfaz del usuario _____ 15

AbortRetryIgnore _____ 16
ArrowCursor _____ 16
CloseAlert _____ 16
CropRect2Rect _____ 16
drawStrings _____ 17
ElapsedTime _____ 17

Error 17
FatalError 17
FilterString 18
GetISTR 18
GetSelRCoords 18
HaveSelection 18
InputDialog 18
JPEGDialog 19
OkCancel 19
OpenAlert 19
PopUp 19
PopMenu 19
printf 19
ProgressMeter 20
RetryCancel 20
scanf 21
SetISTR 21
ShowImage 22
TextView 22
TIFFDialog 22
WaitCursor 23
Warning 23
YesNo 23
YesNoCancel 23

Bibliografía 24

ÍNDICE ALFABÉTICO DEL ANEXO A

AbortRetryIgnore 16
ArrowCursor 16
AskColorType 4
Bibliografía 24
BlueComponent 11
ChangeImageFormat 4
changeSuffix 6
CloseAlert 16
Conv24to8 7
Conv8to24 7
Conv24toGrey 7
ConvGreyto24 7
Conv8toGrey 7
CropRect2Rect 16
drawStrings 17
ElapsedTime 17
endGreyAlg 8
end24bitAlg 8
Error 17
FatalError 17
FilterString 18
Funciones de la interfaz del usuario 15
Funciones relacionadas con el formato de las imágenes 4
Gestión de memoria 2
GetImage 9
GetISTR 18
GetColorType 10
GetFormatFromSuffix 10
GetFullName 11
GetSelRCoords 18
GreenComponent 11
HaveSelection 18
hsv2rgb 11
InputDialog 18
IOfile 2
~IOfile 3
JPEGDialog 19
Lectura y escritura de una imagen en un fichero. 1
OkCancel 19
OpenAlert 19
OpenPic 1
PopUp 19
PopMenu 19
printf 19
ProgressMeter 20
RedComponent 11
RestorePic 4
rgb2hsv 11
SaveMemory 3
SaveMemory 3
SavePic 2

scanf 21
SetFullName 11
StoreNewPic 3
start24bitAlg 15
startGreyAlg 15
RedrawPic 13
RetryCancel 20
RGBtoPIC24 14
SetISTR 21
ShowImage 22
TextView 22
TIFFDialog 22
WaitCursor 23
Warning 23
YesNo 23
YesNoCancel 23

Anexo B

Funciones públicas contenidas en la clase `ImgBasic`

En el capítulo 6 de la memoria del proyecto fin de carrera, se ha intentado proporcionar una visión global de la clase, sin entrar en detalles técnicos que pudieran confundir al lector. Este anexo contiene información de referencia para todas las funciones públicas de la clase `ImgBasic`, ordenadas alfabéticamente. El anexo B muestra, para cada función:

- la declaración de la función
- los argumentos que toma la función
- el valor devuelto por la función
- se indicarán las funciones virtuales y virtuales públicas en la declaración (las funciones no virtuales no poseen ningún identificador específico en C++).

En este anexo no se explicará la forma particular de implementar cada algoritmo, ya que ello requería mucho esfuerzo y la mayoría de las veces confundiría al usuario más qu ayudarlo. Si, por alguna circunstancia alguien necesitara modificar los algoritmos, el código fuente viene suficientemente comentado para que sea fácilmente comprensible. También puede ser necesario observar el listado de código cuando se necesite información muy precisa de cómo actúan los algoritmos (aunque los algoritmos incluidos –excepto los algoritmos multirresolución– son clásicos dentro del procesamiento de imágenes, puede haber pequeñas variantes que modifiquen ligeramente los resultados).

Por otra parte, algunos algoritmos se han adaptados de [XVJ94] y [MAM95]. Estos autores permiten la utilización del material siempre que sea sin ánimo de lucro. He sido lo más riguroso posible cuando he utilizado código de otros autores, y en listado de código fuente he mantenido e indicado las referencias del material utilizado.

Aproximadamente, la mitad del código de la clase ha sido de creación propia y la otra mitad ha sido adaptada de otros programas.

B.1 Funciones generales

- ***virtual void DoAlg (int);***

Esta función realiza el algoritmo que se especifica en la llamada de la función sobre la imagen contenida en `IOfile`. La función `DoAlg` sólo tiene como parámetro un identificador con el cual se selecciona el algoritmo que se va a realizar. Toma como argumento un entero que puede tener los valores `ALG_BLUR`, `ALG_BLEND`, `ALG_CANNY`, `ALG_CANYMRES`, `ALG_CROP`, `ALG_EDGE`, `ALG_FASTMRES`, `ALG_GAMMA`, `ALG_GSBLUR`, `ALG_GRAD_ROB`, `ALG_GSNOISE`, `ALG_MEDIAN`, `ALG_MULTIRES`, `ALG_NONE`, `ALG_OIL`, `ALG_PIXEL`, `ALG_RESIZE`, `ALG_ROTATE`, `ALG_ROTATECLR`, `ALG_SATURATE`, `ALG_SHARPEN`, `ALG_SPREAD` y `ALG_TINF`. A continuación se explican los efectos que tienen cada identificador.

En este apartado no voy a explicar detalladamente los algoritmos, sino que comentaré la función `DoAlg,` que se ideó para poder llamar de forma sencilla a los algoritmos con una sola función —en el anexo B se encontrará información más detallada sobre todas las funciones contenidas en la clase `ImgBasic`—.

La función `DoAlg` sólo tiene como parámetro un identificador con el cual se selecciona el algoritmo que se va a realizar. En realidad esta función actúa como pasarela, ya que solamente llama a otra función según su parámetro. A continuación se enumeran los identificadores que tiene asociados cada identificador, con una descripción breve:

–"`ALG_BLUR`": Realiza un suavizado de la imagen pasando una máscara unidad, es decir, todos los pixels dentro de la ventana se ponderan de igual forma. El valor del pixel filtrado es la media de los pixels que forman la ventana. Equivale a la función *`Blur`*().

–"ALG_BLEND": Realiza el efecto explosión sobre la imagen. Equivale a la función *Blend*().

–"ALG_CANNY": Realiza el algoritmo de extracción de contornos de Canny. Equivale a la función *AlgCanny*().

–"ALG_CANYMRES": Aplica un análisis multirresolución para la detección de contornos tal como se indica en el capítulo siete de la memoria. Equivale a la función *CannyMRes*().

–"ALG_CROP": . Toma una porción de la imagen original como nueva función, es decir, crea una subimagen de la original recortando las dimensiones. Equivale a la función *CropPic*().

–"ALG_EDGE": Calcula el gradiente de una imagen utilizando el filtro Sobel. Equivale a la función *SobelGradient*().

–"ALG_FASTMRES": Este algoritmo se diferencia del anterior en el uso de filtros iterativos y optimizaciones para reducir el tiempo de proceso. Equivale a la función *Multires*(ALG_FASTMRES).

–"ALG_GAMMA": Corrige el valor de una imagen (por ejemplo el factor gamma) a partir de tablas de corrección de las componentes de color rojo, verde y azul, y/o tono, saturación y luminosidad. Equivale a la función

–"ALG_GRAD_ROB": Calcula el gradiente de la imagen según la fórmula de Roberts. Equivale a la función *GradientRoberts*().

–"ALG_GSBLUR": Efectúa un filtrado gausiano de la imagen. Se diferencia de "ALG_BLUR" en que la ventana de filtrado tiene los valores correspondientes a la distribución gausiana, en vez de ser una máscara con todos sus componentes la unidad Equivale a la función *GaussianBlur*().

–"ALG_GSNOISE": Superpone un ruido gausiano a la imagen. Equivale a la función *GaussianNoise*().

–"ALG_MEDIAN": Aplica el filtro de la mediana a la imagen contenida en memoria. Equivale a la función *MedianFilter*().

–"ALG_MULTIRES": Efectúa una extracción de contornos y caracterización de los mismos basada en técnicas multirresolución según [MAM95] y [JRB94].

–"ALG_NONE": Deshace el último algoritmo realizado a la imagen. Equivale a la función *RestorePic*().

–"ALG_OIL": Da el efecto de pintura al óleo a las imágenes. Equivale a la función *OilPaint*().

–"ALG_PIXEL": Crea el efecto cuadrícula de la imagen, es decir, agrupa los pixels en celdas rectangulares y da un color promedio a cada celda de la imagen. Equivale a la función *Pixelize*().

–"ALG_RESIZE": Escala la imagen, sin recortar la imagen. Equivale a la función *ResizePic*().

–"ALG_ROTATE": Realiza una rotación de la imagen sobre la original como fondo. Equivale a la función *FineRotate*(0).

–"ALG_ROTATECLR": Realiza una rotación de la imagen sobre un fondo negro. Equivale a la función *FineRotate*(1).

–"ALG_SATURATE": Esta función es un caso particular del ajuste de la luminosidad de una imagen. Multiplica la luminosidad de cada pixel por un factor, lo cual es apropiado para imágenes con poca iluminación, oscuras, como los resultados de los gradientes. Equivale a la función *Saturate*().

–"ALG_SHARPEN": Aumenta el contraste en los contornos, restando al valor de la luminosidad del pixel analizado la media de la luminosidad de los 9 pixels de una ventana 3x3 centrada en el punto. La media de la luminosidad se pondera con el valor de la luminosidad del punto analizado. Equivale a la función *Sharpen*().

–"ALG_SPREAD": Realiza un desplazamiento aleatorio sobre cada pixel de la imagen, a otro punto de la ventana considerada. Este efecto se puede considerar un tipo de ruido o distorsión de la imagen. Equivale a la función *Spread*().

–"ALG_TINF": Crea el efecto de bajorrelieve a partir de una imagen. Equivale a la función *TinFoil*().

B.1.1 Manipulaciones básicas

- ***virtual void RestorePic(void);***

Deshace el último algoritmo realizado a la imagen, siempre que no esté seleccionada la opción de ahorrar memoria. (Esta función está implementada en la clase `IOfile`, pero por su utilidad se incluye su declaración en este capítulo).

- ***virtual void CropPic ();***

Recorta la imagen, eliminando parte de los bordes de la imagen hasta los márgenes que introduce el usuario. Pregunta la nueva dimensión de la imagen. El procesamiento no lo realiza directamente esta función, sino que llama a la función *XVGetSubImage* con los parámetros correspondientes a la dispersión introducida por el usuario y a la imagen contenida en la clase `IOfile`.

- ***byte* XVGetSubImage (byte *imagen, int TipoImagen,***
  int anchura, int altura,
  int selx, int sely, int selw, int selh);

Esta función recorta imágenes, es decir, devuelve la parte de la imagen original cuyas coordenadas se especifican en *selx*, *sely*, *selw* y *selh*. La imagen original se pasa como *imagen* en el formato indicado por *TipoImagen* (PIC8 o PIC24), la *anchura* y *altura* original se pasan en tercer y cuarto lugar.

Esta función calcula la zona rectangular de la imagen cuya esquina superior se especifica e las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh*. La función retorna la dirección de memoria en donde se guarda la imagen recortada. Este algoritmo ha sido adaptado de [XVJ94].

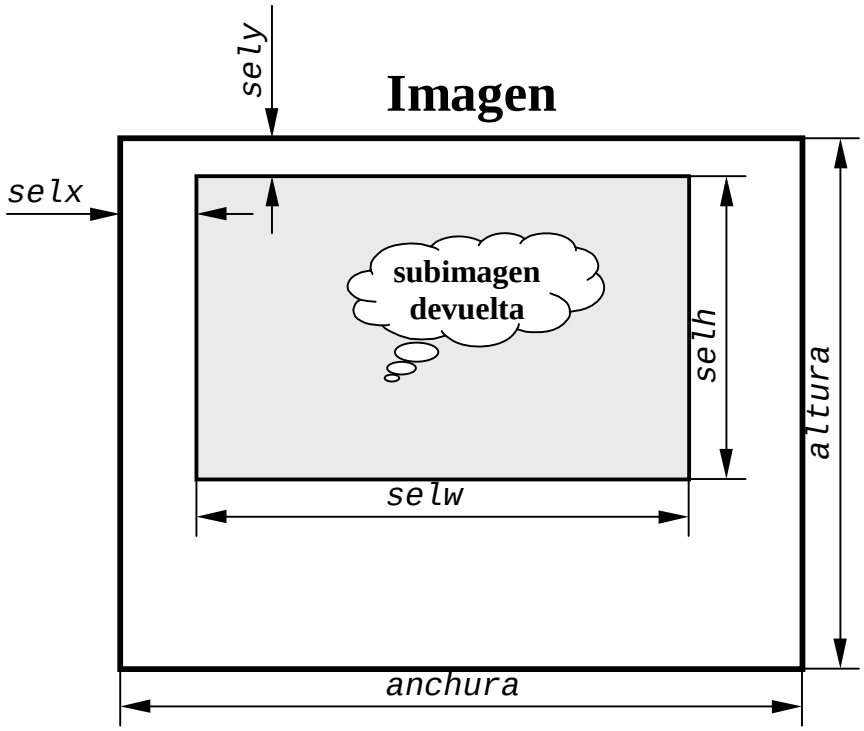


Fig. 1: Parámetros de la función *XVGetSubImage*.

- ***virtual void ResizePic ();***

Escala la imagen, reduciendo o ampliando la imagen sin recortar los bordes. Pregunta la nueva dimensión de la imagen. El escalado no lo realiza directamente esta función, sino que llama a la función *doSmooth24* con los parámetros correspondientes a la dispersión introducida por el usuario y a la imagen contenida en la clase `IOfile`.

- ***byte* Smooth24 (byte *imagen, int TipoImagen,***
  int anchuraOriginal, int alturaOriginal,
  int anchuraNueva, int alturaNueva,
  byte *rmap, byte *gmap, byte *bmap);

Esta función escala imágenes, es decir, reduce o amplia imágenes desde la anchura y altura de la imagen original hasta las nuevas anchuras y alturas. La imagen original se pasa como *imagen* en el formato indicado por *TipoImagen* (PIC8 o PIC24). La *anchura* y *altura* original se pasan en tercer y cuarto lugar. Las nuevas dimensiones se pasan en *anchuraNueva* y *alturaNueva*. La función retorna la dirección de memoria en donde se guarda la imagen escalada.

Este algoritmo ha sido adaptado de [XVJ94]. Las imágenes en tonos de gris o con mapa de color son convertidas a formato RGB, y por lo tanto si es necesario mantener el formato original, debe ser convertida al formato original después del escalado.

- ***byte* SmoothResize (byte *imagen8,***
  int anchuraOriginal, int alturaOriginal,
  int anchuraNueva, int alturaNueva,
  byte *RedMap, byte *GreenMap, byte *BlueMap,
  byte *rdisplay, byte *gdisplay, byte *bdisplay,
  int ColoresDisplay);

Esta función escala imágenes de tipo PIC8 (con mapa de colores o en tonos de gris). La imagen original se pasa como *imagen8*, el mapa de colores es *RedMap* para la componente roja, *GreenMap* para la componente verde y *BlueMap* para la componente azul. La *anchura* y *altura* original se pasan en *anchuraOriginal* y *alturaOriginal*. Las nuevas dimensiones se pasan en *anchuraNueva* y *alturaNueva*. La función retorna la dirección de memoria en donde se guarda la imagen escalada en formato PIC8.

Esta función utiliza *Smooth24* para calcular la imagen escalada y luego llama la función *DoColorDither*, que procede a la reducción y oscilación de colores para mostrar la imagen en un dispositivo con una paleta formada por *rdisplay*, *gdisplay* y *bdisplay* para las componentes roja, verde y azul respectivamente con *ColoresDisplay* colores ubicados.

Este algoritmo ha sido adaptado de [XVJ94]. Esta función no tiene mucha utilidad en el entorno de visión ya que la reducción de colores y oscilación se produce directamente en la capa que se ocupa de la visualización de las imágenes, y por tanto ya no es necesario la reducción directa del número de colores y aplicar la técnica de la oscilación del color (“*color dithering”*).

- ***byte* DoColorDither (byte *imagenRGB, byte *imagen8,***
  int anchura, int altura,
  byte *RedMap, byte *GreenMap, byte *BlueMap,
  byte *rdisplay, byte *gdisplay, byte *bdisplay,

int ColoresDisplay);

Esta función adapta imágenes de tipo PIC8 (con mapa de colores o en tonos de gris) o RGB para poder ser mostradas en un dispositivo con una paleta de colores (por ejemplo, una pantalla que sólo puede mostrar 256 colores distintos). Si la imagen es de tipo RGB, *imagenRGB* es un puntero no nulo a la imagen e *imagen8*, *RedMap*, *GreenMap*, *BlueMap* no se utilizan (estos parámetros pueden ser NULL). Si la imagen original es de tipo PIC8, *imagenRGB* debe ser un puntero nulo -NULL-, la imagen original se pasa como *imagen8,* el mapa de colores es *RedMap* para la componente roja, *GreenMap* para la componente verde y *BlueMap* para la componente azul. A*nchura* y *altura* son las dimensiones de la imagen. La función retorna la dirección de memoria en donde se guarda la imagen escalada en formato PIC8.

Esta función reduce el número de colores distintos presentes en una imagen y produce la oscilación de colores (pequeñas variaciones de color en pixels contiguos que a cierta distancia no se perciben y que producen el efecto visual de una mayor gama de colores), para mostrar la imagen en un dispositivo con una paleta formada por *rdisplay*, *gdisplay* y *bdisplay* para las componentes roja, verde y azul respectivamente con *ColoresDisplay* colores ubicados.

Este algoritmo ha sido adaptado de [XVJ94]. Esta función no tiene mucha utilidad en el entorno de visión ya que la reducción de colores y oscilación se produce directamente en las clases que se ocupan de la visualización de las imágenes, y por tanto ya no es necesario la reducción directa del número de colores y aplicar la técnica de la oscilación del color ("*color dithering"*).

- ***byte* Do332ColorDither (byte *imagenRGB, byte *imagen8,***
  int anchura, int altura,
  byte *RedMap, byte *GreenMap, byte *BlueMap,
  byte *rdisplay, byte *gdisplay, byte *bdisplay,
  int ColoresDisplay);

Esta función sólo se diferencia de *DoColorDither* en que *Do332ColorDither* se ha optimizado para dispositivos que tengan un mapa de color estándar formado por 256 colores distintos distribuidos linealmente con tres bits

dedicados a representar la intensidad del color rojo, tres bits dedicados a la componente verde y dos bits para la componente azul.

- ***void InvertPic (void);***

Invierte la imagen contenida en memoria. En una imagen en tonos de gris, cada pixel tiene una luminosidad inicial $x_{i,j}$ que toma valores entre cero y 255. Después de aplicar el algoritmo el pixel tiene la luminosidad $x_{i,j}' = 255 - x_{i,j}$. Para imágenes en color se invierte cada una de las componentes roja, verde y azul. Si $r_{i,j}$, $g_{i,j}$ y $b_{i,j}$ son las componentes roja, verde y azul respectivamente, los nuevos valores serán $r_{i,j}' = 255 - r_{i,j}$, $g_{i,j}' = 255 - g_{i,j}$ y $b_{i,j}' = 255 - b_{i,j}$. Esta función llama a su función homónima con lista de parámetros larga, indicando en los parámetros las propiedades de la imagen contenida en la clase `IOfile`.

- ***void InvertPic (byte *imagen, int TipoImagen,***
  u_int anchura, u_int altura, int monocromo = 0,
  byte *RedMap = NULL, byte *GreenMap = NULL,
  byte *BlueMap = NULL, int colores = 256);

Esta función invierte la imagen contenida en el parámetro *`imagen`*; el tipo de imagen (PIC8 o PIC24) se especifica en *`TipoImagen`*; la *`anchura`* y *`altura`* de la imagen se detallan a continuación; *monocromo* debe ser distinto de cero si la imagen es del tipo 256 tonos de gris o en blanco y negro (criterio *"min is black"*: valor cero equivale a negro, valor uno equivale a blanco); el mapa de colores es *`RedMap`* para la componente roja, *`GreenMap`* para la componente verde y *`BlueMap`* para la componente azul; el número de colores del mapa de colores se indica en el parámetro *`colores`*.

Los parámetros por defecto son una imagen en color, sin mapa de color (si la imagen es de tipo PIC8 y monocromo es falso se debe especificar los mapas de color) y con el número máximo de colores en la paleta (este parámetro sólo afecta al algoritmo si la imagen es de tipo PIC8 y monocromo es falso).

- ***void RotatePic (int direccion);***

Rota 90 grados en el sentido de las manecillas del reloj si *`direccion`* es cero o en sentido contrario a las manecillas si *`direccion`* es distinto de cero. Esta función

llama a su función homónima con lista de parámetros larga, indicando en los parámetros las propiedades de la imagen contenida en la clase `IOfile`.

- ***void RotatePic (int direccion, byte *imagen, int TipoImagen, u_int* anchura, u_int* altura);***

Rota la *imagen* 90 grados en el sentido de las manecillas del reloj si *dirección* es cero o en sentido contrario a las manecillas si *direccion* es distinto de cero; *imagen* es el puntero a la zona de memoria donde se contiene la imagen; el tipo de imagen (PIC8 o PIC24) se indica en *TipoImagen*; la anchura y altura de la imagen se han de especificar por referencia ya que al girar 90 grados hay que intercambiar la *altura* y *anchura*. Para girar un ángulo arbitrario se ha de utilizar *FineRotate*().

- ***void FlipPic (int direccion);***

Voltea la imagen contenida en `IOfile` horizontalmente si *direccion* es cero o verticalmente si *direccion* es distinto de cero. Esta función llama a su función homónima con lista de parámetros larga, indicando en los parámetros las propiedades de la imagen contenida en la clase `IOfile`.

- ***void FlipPic (int direccion, byte *imagen, int TipoImagen, u_int anchura, u_int altura);***

Voltea la *imagen* horizontalmente si *direccion* es cero o verticalmente si *direccion* es distinto de cero; *imagen* es el puntero a la zona de memoria donde se contiene la imagen; el tipo de imagen (PIC8 o PIC24) se indica en *TipoImagen*; la *anchura* y *altura* de la imagen se indican en último lugar.

B.1.2 Ajuste de la curva gamma

- ***void saturePic24 (byte *imagenRGB, int anchura, int selx, int sely, int selw, int selh, float factor=1.0);***

Esta función calcula la máxima luminosidad presente en la imagen, denotada por *ValorMáximoLuminosidad*. Si *factor* es la unidad, multiplica las componentes del color rojo, verde y azul por un coeficiente de escala tal que al punto de máxima

luminosidad le corresponda el valor 255 –es decir, expande linealmente el histograma hasta ocupar todos los valores posibles, sin que se produzca saturación–. Si *factor* es distinto de la unidad el coeficiente de escala será tal que la nueva luminosidad máxima teórica será *factor·ValorMáximoLuminosidad*. En el caso que *factor* sea mayor que uno, se produce una saturación de la imagen debido a que el valor máximo las componentes es 255 (los valores superiores a 255 son truncados a ese valor y los pixels están saturados).

La imagen original en formato RGB se pasa como *imagenRGB*, la *anchura* se pasa en segundo lugar. Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifica en *selw* y *selh* (para tratar toda la imagen, tomar *selx=sely=0*, *selw=anchura*, *selh=altura*). Esta función sobrescribe la imagen original que se pasa como argumento.

- ***void saturePicGrey (byte *imagenGris, int anchura, int selx, int sely, int selw, int selh, float factor=1.0);***

Análogamente a *saturePicGrey*, esta función calcula la máxima luminosidad presente en la imagen imagen, denotada por *ValorMáximoLuminosidad*. Si *factor* es la unidad, multiplica el valor de cada pixel por un coeficiente de escala tal que el nuevo punto de máxima luminosidad le corresponda el valor 255 –es decir, expande linealmente el histograma hasta ocupar todos los valores posibles, sin que se produzca saturación–. Si *factor* es distinto de la unidad el coeficiente de escala será tal que la nueva luminosidad máxima teórica será *factor·ValorMáximoLuminosidad*. En el caso que *factor* sea mayor que uno, se produce una saturación de la imagen debido a que el valor máximo la luminosidad es 255 (los valores superiores a 255 son truncados a ese valor y los pixels están saturados).

La imagen original con 256 tonos de gris se pasa como *imagenGris*, la *anchura* se pasa en segundo lugar. Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de

la zona a tratar se especifican en *selw* y *selh* (para tratar toda la imagen, tomar *selx=sely=0*, *selw=anchura*, *selh=altura*). Esta función sobrescribe la imagen original que se pasa como argumento.

- ***double EvalSpline(int x[],int y[],double y2[],int n, double point);***

Calcula el valor de una función *spline* que pasa por todos los pares de puntos *x*[0..*n*-1] e *y*[0..*n*-1] y cuya segunda derivada en cada uno de los pares de puntos es *y2*[0..*n*-1]. La función devuelve el valor del *spline* en el punto *point* (*x*[0] < *point* < *x*[*n*-1]), correspondiente a la coordenada *y*. El vector de puntos *x* debe estar ordenado de menor a mayor para que el algoritmo funcione correctamente.

- ***void InitSpline(int x[], int y[], int n, double y2[]);***

Dado un vector de puntos datos *x*[0..*n*-1] e *y*[0..*n*-1], calcula los valores de la segunda derivada en cada uno de los puntos dados *y2*[0..*n*-1] para poder utilizar la función *EvalSpline*. Los puntos deben estar ordenados según el valor de x de menor a mayor, es decir *x*[*i*] < *x*[*i*-1] para todo 0<*i*<*n*-1.

- ***void GenerateFSGamma (int x[],int y[],int n, byte fsgamcr[]);***

Esta función genera la curva gamma de corrección de color Floyd-Steinberg [IJG94]. Esta función genera una función *spline* de cuatro puntos, que se utilizará como un mapa de grises no lineal. El primer punto siempre es (0,0) y el último (255,255) –estos puntos no deben cambiarse–. Los puntos intermedios pueden cambiarse para ajustar la respuesta del color. Por ejemplo, si especifica los puntos (50,0) y (200,255), los puntos con luminosidad entre 0 y 50 se convertirán en negro (luminosidad 0); los puntos con luminosidad mayor que 200 se convertirán en blanco (luminosidad 255) y los valores entre 50 y 200 se transformarán en el rango 0-255 –la curva de respuesta tendrá una forma parecida a una rampa curvada–.

- ***byte* FSDither(byte *imagen, int* TipoImagen*, int anchura, int altura, byte *fsgamcr, byte *RedMap = NULL, byte *GreenMap = NULL,***

```
byte *BlueMap = NULL);
```

Esta función efectúa el algoritmo de dispersión/oscilación *Floyd-Steinberg* según una curva de ajuste gamma a la *imagen* especificada en el primer parámetro; el tipo de imagen —PIC8 o PIC24— se especifica en *TipoImagen* (si la imagen es de tipo PIC8 se debe especificar los mapas de color); la *anchura* y *altura* de la imagen se detallan a continuación; *fsgamcr* es la tabla de corrección de la luminosidad; el mapa de colores está formado por *RedMap* para la componente roja, *GreenMap* para la componente verde y *BlueMap* para la componente azul. El resultado es una imagen tal como se especifica con el formato *picType* = PIC8, *mono* = TRUE, *numcols* = 2 (es decir, una imagen con dos valores posibles, blanco correspondiente al valor 1 y negro correspondiente al valor 0, utilizando un byte para almacenar cada pixel).

- ```
  byte *GammifyPic24(byte *imagenRGB, int anchura,
          int altura, byte RedRemap[/*256*/],
          byte GreenMap[/*256*/],byte BlueMap[/*256*/],
          int Saturation =0, int ValueRemap[/*256*/]=NULL,
          int HueRemap[/*360*/] = NULL);
  ```

Aplica las modificaciones según los ajustes RGB/HSV a cada pixel en una imagen de tipo RGB–24 bits por pixel. Crea y devuelve una nueva imagen, o NULL en caso de fallo. También comprueba si la imagen procesada es la misma que la original y en tal caso devuelve NULL para ahorrar tiempo.

El rango del tono, es decir, el rango de los valores contenidos en *HueRemap*[], están entre 0 y 359, la saturación *Saturation* está entre 0 y 100, y la intensidad, contenida en *ValueRemap*[], está entre 0 y 255. Las componentes roja, verde y azul, representadas por *RedRemap*[], *GreenRemap*[] y *BlueRemap*[], respectivamente, toman valores entre 0 y 255.

La *imagen* original se especifica en el primer parámetro y la *anchura* y *altura* de se detallan a continuación. El mapa de corrección de colores es *RedRemap* para la componente roja, *GreenRemap* para la componente verde y *BlueRemap* para la componente azul (Si no se requiere corrección de color, basta con hacer *RedRemap*[i] = *GreenRemap*[i] = *BlueRemap*[i] =i; para todo i entre 0 y 255). *Saturation* es el desplazamiento de la saturación con respecto al original
  ```

(el valor cero indica que la saturación no se corregirá). `ValueRemap` es la tabla de corrección de la luminosidad para cada valor de la intensidad (si no se requiere corrección de la luminosidad basta con hacer `ValueRemap[i] = i;` para todo `i` entre 0 y 255). `HueRemap` es la tabla de corrección para el tono (si no se requiere corrección del tono basta con hacer `HueRemap[i] = i;` para todo `i` entre 0 y 359).

Si sólo se quiere corregir las componentes roja, verde y azul se pueden utilizar los valores por defecto de los parámetros HSV, siendo innecesario transformar las coordenadas RGB en HSV, y por tanto, reduciéndose el tiempo de cálculo. Si `Saturation`, `ValueRemap` o `HueRemap` toman un valor distinto del utilizado por defecto, se utilizará la transformación RGB y HSV, la corrección posterior en modo HSV, la transformación inversa HSV a RGB, y la corrección de las componentes roja, verde o azul.

- ***void SortSpline (int x[],int y[],int n);***
  void intsort (int x[], int n);

La función *intshort* ordena de menor a mayor un vector de enteros con *n elementos*. La función *ShortSpline* ordena *n* pares de coordenadas (x,y) según el valor de la ordenada *x*, de menor a mayor. Utiliza el algoritmo de D.L. Shell [WIR76], que proporciona buenos resultados para pequeños grupos de datos y evita la sobrecarga asociada a las llamadas de funciones del programa *qsort*().

B.1.3 Efectos sobre las imágenes

- ***void TinFoil()***

Crea el efecto de bajorrelieve a partir de una imagen restando a la luminosidad de la imagen original la misma desplazada hacia abajo e izquierda aproximadamente un pixel (en realidad se aplica el filtro *doAngleConvolv)*.

- ***void doAngleConvolv (byte *imagenRGB, int anchura,***
  int altura, byte * imagenFiltradaRGB,
  int selx, int sely, int selw, int selh);

Realiza el efecto de bajorrelieve sobre una imagen, resultado de aplicar la máscara $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ a la luminosidad de la imagen.

La imagen original en formato RGB se pasa como *imagenRGB*, la *anchura* y *altura* se pasan en segundo y tercer lugar. La imagen filtrada se guarda en la zona de memoria apuntada por el cuarto parámetro.

Esta función puede realzar zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para tratar toda la imagen, tomar *selx=sely=0*, *selw=anchura*, *selh=altura*). Este algoritmo ha sido adaptado de [XVJ94].

- ***virtual void OilPaint (void);***

Crea el efecto de pintura al óleo. Esta función llama a *doSharpConvolv* para realizar el filtrado sobre la imagen contenida en memoria.

- ***void doOilPaint (byte *imagenRGB, int anchura,***
  int altura, byte * imagenFiltradaRGB,
  int selx, int sely, int selw, int selh,
  int ventana);

Realiza el efecto de pintura al óleo sobre una imagen tal como se describe[HOZ90]. La imagen original en formato RGB se pasa como *imagenRGB*, la *anchura* y *altura* se pasan en segundo y tercer lugar. La imagen filtrada se guarda en la zona de memoria apuntada por el cuarto parámetro.

Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para tratar toda la imagen, tomar *selx*=*sely*=0, *selw*=*anchura*, *selh*=*altura*).

El tamaño de la *ventana* se utiliza para calcular el histograma de la ventana centrada en el pixel bajo estudio, y se reemplaza el color original del pixel por el color que aparece con más frecuencia en la ventana (*ventana* debe ser impar).

La versión original de este algoritmo fue tomada de "pgmoil.c", escrita por Wilson Bent y distribuida en el paquete informático PBMPLUS (dirección URL “wuarchive.wustl.edu/graphics/graphics/packages/NetPBM. Más tarde John Bradley la adaptó al programa XV [XVJ94], y esta última versión es la que se ha utilizado finalmente a la clase `ImgBasic`.

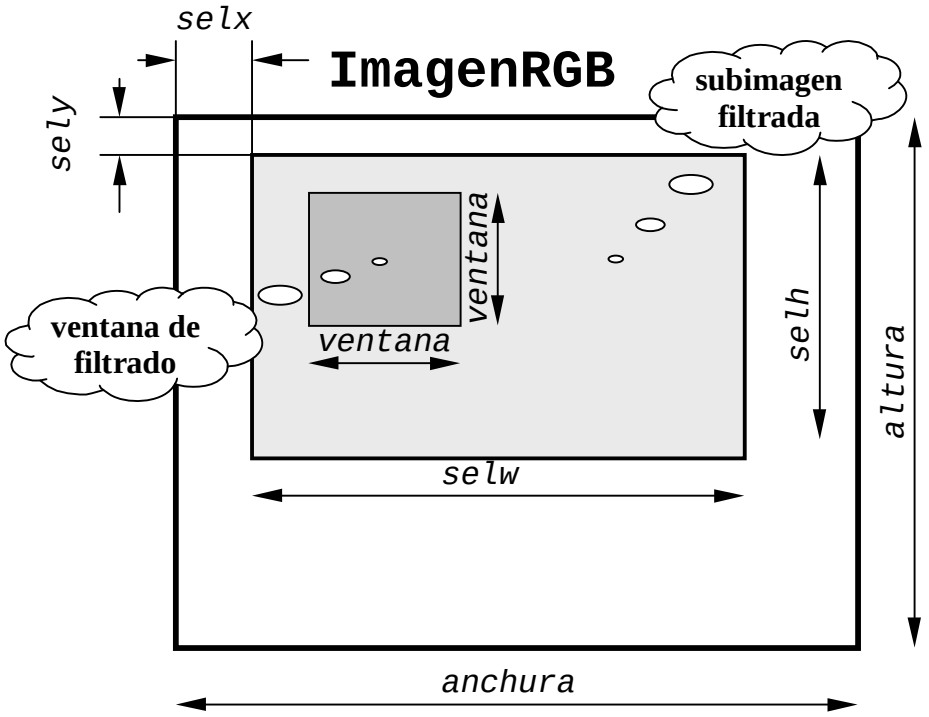


Fig. 2: Parámetros utilizados en la función *`doOilPaint`*.

- ***`virtual void Blend (void);`***

Crea el efecto de explosión de la imagen. Esta función llama a *`doBlend`* para realizar el filtrado sobre la imagen contenida en memoria.

- ***`void doBlend (byte *imagenRGB, int anchura,`***
  `int altura, byte * imagenFiltradaRGB,`
  `int selx, int sely, int selw, int selh);`

Realiza el efecto explosión sobre la imagen. Calcula el color promedio de todos los pixels pertenecientes a una recta que parte del centro de la imagen al exterior. En el centro pone el color promedio de los pixels que forman una recta que parte del centro de la imagen hacia el exterior. En el resto de los puntos de la recta, calcula un color sopesando el color promedio y el color original del pixel.

La imagen original en formato RGB se pasa como *imagenRGB*, la *anchura* y *altura* se pasan en segundo y tercer lugar. La imagen filtrada se guarda en la zona de memoria apuntada por el cuarto parámetro.

Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para tratar toda la imagen, tomar *selx*=*sely*=*0*, *selw*=*anchura*, *selh*=*altura*). Este algoritmo ha sido adaptado de [XVJ94].

- ***virtual void FineRotate (int FondoNegro);***

Pregunta el ángulo de giro de la ventana de suavizado. Si *FondoNegro* vale cero, la imagen rotada se muestra con la imagen original de fondo; en otro caso la imagen rotada se superpone a un fondo negro .El giro no lo realiza directamente esta función, sino que llama a la función *doRotate* con los parámetros correspondientes al giro introducido por el usuario y a la imagen contenida en la clase `IOfile`.

- ***void doRotate (byte *imagenRGB, int anchura,***
  int altura, byte * imagenFiltradaRGB,
  int selx, int sely, int selw, int selh,
  double Rotacion, int FondoNegro);

Rota la imagen original un ángulo cualquiera tomando como fondo la imagen original o bien un fondo negro.

La imagen original en formato RGB se pasa como *imagenRGB*, la *anchura* y *altura* se pasan en segundo y tercer lugar. La imagen filtrada se guarda en la zona de memoria apuntada por el cuarto parámetro.

Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para rotar toda la imagen, tomar *selx*=*sely*=*0*, *selw*=*anchura*, *selh*=*altura*).

Rotacion indica en grados sexagesimales el giro de la imagen en sentido contrario al de las agujas del reloj (si el ángulo es negativo la rotación se produce en el sentido de las agujas de reloj). Si *FondoNegro* tiene un valor distinto de cero, la imagen rotada

se superpone a un fondo negro. Si *FondoNegro* vale cero, la imagen rotada se muestra con la imagen original de fondo. Este algoritmo ha sido adaptado de [XVJ94].

- ***virtual void Pixelize ();***

Pregunta el tamaño de la cuadrícula y procesa la imagen. El tratamiento no lo realiza directamente esta función, sino que llama a la función *doPixel* con los parámetros correspondientes al giro introducido por el usuario y a la imagen contenida en la clase IOfile.

- ***void doPixel (byte *imagenRGB, int anchura,***
  int altura, byte * imagenFiltradaRGB,
  int selx, int sely, int selw, int selh,
  int pixX, int pixY);

Crea el efecto cuadrícula de la imagen, es decir, agrupa los pixels en celdas rectangulares y asigna un color promedio a cada celda de la imagen. El tamaño de la cuadrícula se especifica en los dos últimos parámetros de la función, pixX-por-pixY.

La imagen original en formato RGB se pasa como *imagenRGB*, la *anchura* y *altura* se pasan en segundo y tercer lugar. La imagen filtrada se guarda en la zona de memoria apuntada por el cuarto parámetro.

Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para rotar toda la imagen, tomar *selx*=*sely*=*0*, *selw*=*anchura*, *selh*=*altura*). Este algoritmo ha sido adaptado de [XVJ94].

B.1.4 Filtrados

- ***virtual void Blur ();***

Pregunta el tamaño de la ventana de suavizado y difumina la imagen pasando una máscara unidad, es decir, todos los pixels dentro de la ventana se ponderan de igual forma. El valor del pixel filtrado es la media de los pixels que forman la ventana.

El filtrado no lo realiza directamente esta función, sino que llama a las funciones *doBlurConvol24* y *doBlurConvolGrey* con los parámetros correspondientes a

la ventana de filtrado introducida por el usuario y a la imagen contenida en memoria (en la clase `IOfile`).

- ***void doBlurConvolv24(byte *imagenRGB, int anchura,***
  int altura, byte *imagenFiltradaRGB,
  int ventana);
  void doBlurConvolvGrey(byte *imagenGrises, int anchura,
  int altura, byte *imagenFiltradaGrises,
  int ventana);

Producen un filtrado utilizando una ventana cuya dimensión se especifica en la llamada de la función. La máscara utilizada tiene todos sus componentes la unidad, y produce un desplazamiento de la imagen equivalente a la mitad del tamaño de la ventana en el eje horizontal y vertical. Si no se desea que se produzca el desplazamiento se ha de utilizar las funciones *`FastBlurGrey`* y *`FastBlur24`* de la biblioteca "*Miguelan*"[1], ubicada en el subdirectorio "*Miguelan*" del proyecto (se recomienda utilizar esta última versión).

[1] La librería "Miguelan" se creó al principio del proyecto fin de carrera para contener código en lenguaje "C"de [MAM95], y cuyo fin era separar de forma clara este código del resto del entorno de visión artificial. Actualmente, contiene el código original optimizado, y además se ha añadido código nuevo para utilizarlo en los algoritmos multirresolución de detección de contornos.

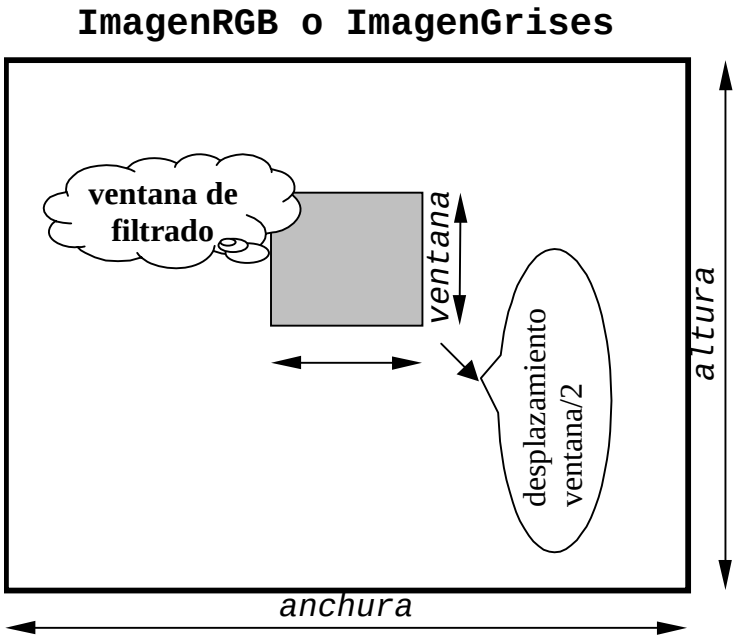


Fig. 3: Parámetros utilizados en las funciones *doBlurConvolvGrey* y *doBlurConvolv24*.

- ***virtual void GaussianBlur ();***

Pregunta el tamaño de la ventana de suavizado y el factor de suavizado. A continuación difumina la imagen pasando una máscara gausiana (se ponderan más los pixels más cercanos y menos los más lejanos). El filtrado gausiano se efectúa con orlado especular en los bordes de la imagen, es decir, para calcular los puntos cercanos al borde, se utiliza los valores reflejados en el límite de la imagen.

El filtrado no lo realiza directamente esta función, sino que llama a las funciones *doGaussianConv24* y *doGaussianConvGrey* con los parámetros correspondientes a la ventana de filtrado introducida por el usuario y a la imagen contenida en la clase `IOfile`.

- ```
  void doGaussianConv24(byte *imagenRGB, int anchura,
                        int altura, byte *imagenFiltradaRGB,
                        float sigma);
  void doGaussianConvGrey(byte *imagenGrises, int anchura,
                        int altura, byte *imagenFiltradaGrises,
                        float sigma);
  ```

Producen un filtrado utilizando una ventana cuya dimensión se especifica en la llamada de la función. La máscara utilizada tiene todos sus componentes la unidad, y
  ```

produce un desplazamiento de la imagen equivalente a la mitad del tamaño de la ventana. Mientras *doGaussianConv24* efectúa el filtrado de una imagen en color tipo RGB, *doGaussianConvGrey* se ha optimizado para imágenes con 256 tonos de gris.

- ***virtual void Sharpen (void);***

Aumenta el contraste en los contornos, restando al valor de la luminosidad del pixel analizado la media de la luminosidad de los 9 pixels de una ventana 3x3 centrada en el punto. La media de la luminosidad se pondera con el valor de la luminosidad del punto analizado. El porcentaje de aumento de contraste es el porcentaje del valor medio de la luminosidad que se resta del valor original de la luminosidad. Esta función llama a *doSharpConvolv* para realizar el filtrado.

- ***void doSharpConvolv(byte *imagenRGB, int anchura, int altura, byte * imagenFiltradaRGB, int selx, int sely, int selw, int selh, int porcentaje);***

Aplica el algoritmo de realce de contornos a una imagen RGB apuntada por *imagenRGB*, de *anchura* y *altura* especificadas como segundo y tercer parámetros. La imagen filtrada se guarda en la zona de memoria apuntada por el cuarto parámetro. Esta función puede realzar zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para tratar toda la imagen, tomar *selx=sely=0*, *selw=anchura*, *selh=altura*). El porcentaje de realce de contornos es el último parámetro de la función. Este algoritmo ha sido adaptado de [XVJ94].

- ***virtual void MedianFilter ();***

Aplica el filtro de la mediana a la imagen. Pregunta la dimensión de la ventana de procesamiento. El filtrado no lo realiza directamente esta función, sino que llama a la función *doMedianFilter* con los parámetros correspondientes a la dispersión introducida por el usuario y a la imagen contenida en la clase IOfile.

- ***void doMedianFilter (byte *imagenRGB, int anchura,***
  int altura, byte * imagenFiltradaRGB,
  int selx, int sely, int selw, int selh,
  int ventana);

Aplica el filtro de la mediana a la imagen. La imagen original en formato RGB se pasa como *imagenRGB*, la *anchura* y *altura* se pasan en segundo y tercer lugar. La imagen filtrada se guarda en la zona de memoria apuntada por el cuarto parámetro.

Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para filtrar toda la imagen, tomar *selx=sely=0*, *selw=anchura*, *selh=altura*).

El tamaño de la *ventana* se utiliza para calcular el histograma de la ventana centrada en el pixel bajo estudio, y se reemplaza el color original del pixel por el color que aparece con más frecuencia en la ventana (*ventana* debe ser impar). Este algoritmo ha sido adaptado de [XVJ94].

B.1.5 Adición de ruido a la imagen

- ***virtual void Spread ();***

Realiza un desplazamiento aleatorio sobre cada pixel de la imagen, a otro punto de la ventana considerada. Pregunta la distancia máxima de desplazamiento y procesa la imagen. El tratamiento no lo realiza directamente esta función, sino que llama a la función *doSpread* con los parámetros correspondientes a la dispersión introducida por el usuario y a la imagen contenida en la clase `IOfile`.

- ***void doSpread (byte *imagenRGB, int anchura,***
  int altura, byte * imagenFiltradaRGB,
  int selx, int sely, int selw, int selh,
  int pixX, int pixY);

Realiza un desplazamiento aleatorio sobre cada pixel de la imagen, a otro punto de la ventana considerada (este efecto se puede considerar un tipo de ruido o distorsión de la imagen). El algoritmo intercambia cada pixel en la imagen con otro dentro de una

ventana cuyo tamaño se especifica en los dos últimos parámetros de la función, *pixX* por *pixY* (si *pixX* es negativo, se toma una ventana cuadrada de módulo de *pixX*).

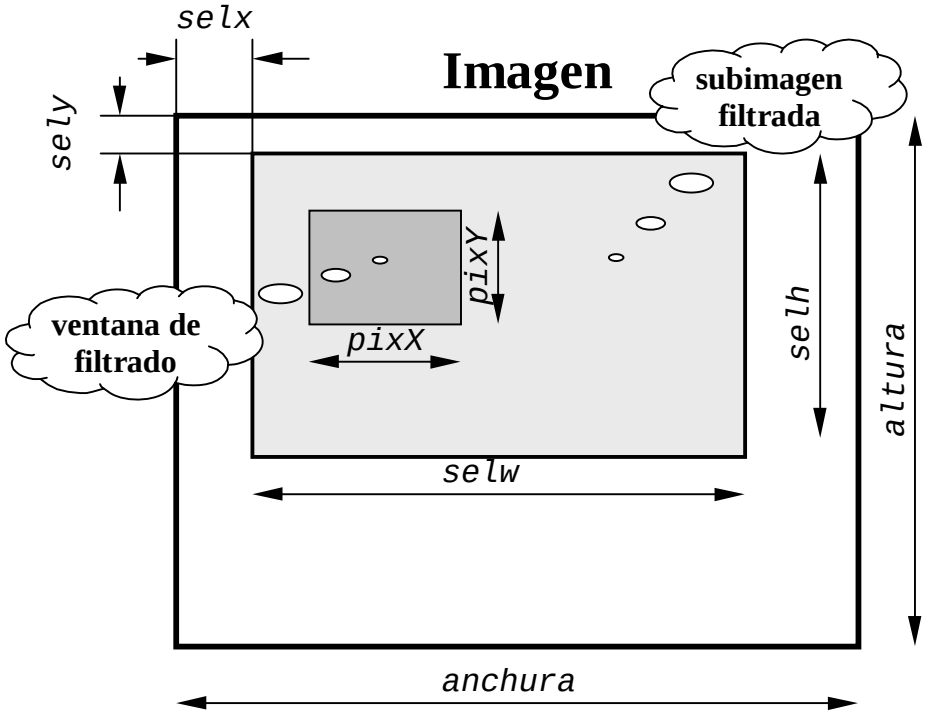


Fig. 4: Ventana de filtrado en el algoritmo doSpread.

La imagen original en formato RGB se pasa como *imagenRGB* (se ha mantenido este parámetro para unificar la interfaz de las funciones ya que el algoritmo utiliza directamente *imagenFiltradaRGB* sin usar *imagenRGB*). La *anchura* y *altura* se pasan en segundo y tercer lugar. *ImagenFiltradaRGB* debe ser inicializada antes de la llamada de la función con la imagen original.

Esta función procesa zonas rectangulares de la imagen, cuya esquina superior se especifica en las coordenadas *selx* y *sely* (tomando como ejes el borde superior izquierdo de la imagen). La anchura y la altura de la zona a tratar se especifican en *selw* y *selh* (para tratar toda la imagen, tomar *selx*=*sely*=*0*, *selw*=*anchura*, *selh*=*altura*). Este algoritmo ha sido adaptado de [XVJ94].

- ***virtual void GaussianNoise();***

Superpone un ruido gausiano a la imagen. Pregunta la relación señal–ruido en decibelios y ensucia la imagen. El tratamiento no lo realiza directamente esta función, sino que llama a la función *doGaussianNoise24* o *doGaussianNoiseGrey*, dependiendo de si la imagen es en color o en tonos de gris, con los parámetros

correspondientes a la dispersión introducida por el usuario y a la imagen contenida en la clase `IOfile`.

- ***void doGaussianNoise24 (byte *imagenRGB, byte * imagenFiltradaRGB, int anchura, int altura, float decibelios);***

Superpone un ruido gausiano a la imagen con la relación señal–ruido en *`decibelios`* (cuanta menor es la relación señal–ruido en decibelios, más ruido presenta la imagen procesada). La imagen original en formato RGB se pasa como *`imagenRGB`*, la imagen generada con el ruido se almacena en *`imagenFiltradaRGB`*, la *`anchura`* y *`altura`* de la imagen se pasan en tercer y cuarto lugar. Para calcular la relación señal–ruido, se separan las componentes roja, verde y azul y se aplica por separado el algoritmo para imágenes en tonos de gris a cada componente.

- ***void doGaussianNoiseGrey (byte *imagenGris, byte * imagenFiltradaGris, int anchura, int altura, float decibelios);***

Superpone un ruido gausiano a la imagen con la relación señal–ruido en *`decibelios`*. La imagen original en tonos de gris se pasa como *`imagenGris`*, la imagen generada con el ruido se almacena en *`imagenFiltradaGris`*, la *`anchura`* y *`altura`* de la imagen se pasan en tercer y cuarto lugar. Este algoritmo ha sido tomado de [MAM95].

B.1.6 Extracción de contornos

En esta sección del capítulo se presentan una serie de algoritmos para la extracción y caracterización de contornos. Los algoritmos clásicos de extracción de contornos se encuentran explicados en la mayoría de los libros de visión artificial. Los algoritmos multirresolución de extracción de contornos que se han incluido en este entorno de visión se basan en [MAM95], que a su vez se fundamentan [JRB94]. Desde mi modesta opinión, recomiendo [MAM95] para obtener una visión profunda de los

algoritmos multirresolución, que además incluye una interesante comparativa entre los algoritmos clásicos para la extracción de contornos.

- ***void SobelGradient (void);***

Aplica el filtro de Sobel para hallar los contornos de la imagen. En realidad realiza la convolución con dos máscaras de detección de contornos (una en dirección horizontal y otra en dirección vertical) y toma el valor absoluto máximo. En los bordes en donde la ventana Sobel tiene puntos fuera de los límites de la imagen, no se detectan contornos. La convolución de las máscaras con la imagen lo realiza la función *doEdgeConvolv24* o *doEdgeConvolvGrey*, dependiendo de si la imagen es en color o en tonos de gris, pasando los parámetros correspondientes a la imagen contenida en la clase `IOfile`.

- ***void doEdgeConvolv24 (byte *imagenRGB, int anchura, int altura, byte * imagenFiltradaRGB, int selx, int sely, int selw, int selh);***

Aplica el filtro de Sobel para hallar los contornos de la imagen. En realidad realiza la convolución con dos máscaras de detección de contornos para las componentes roja, verde y azul y se toma el valor absoluto máximo para cada componente. El valor del gradiente obtenido de la componente roja se guarda en la componente roja, el valor del gradiente de la componente verde se guarda en la componente verde y lo mismo para la componente azul. En los bordes en donde la ventana Sobel tiene puntos fuera de los límites de la imagen, no se detectan contornos (en esos puntos se toma gradiente nulo).

Las dos máscaras utilizadas para el cálculo del filtro Sobel detectan los contornos en la dirección horizontal y vertical, respectivamente.

$$Hor = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad y \quad Vert = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

a efectos prácticos, se ha utilizado cuatro máscaras para disminuir el tiempo de cálculo del algoritmo:

$$a = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad c = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad d = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

La componentes horizontal y vertical se pueden calcular como composición de éstas últimas: Hor = a + 2*b + c, Vert = a - c - 2*d. El módulo del gradiente de la imagen se obtiene calculando el módulo de las dos componentes Sobel. En esta función se ha utilizado el módulo máximo para reducir el tiempo de cálculo:

```
gradiente = max(abs(Hor),abs(Vert)).
```

La imagen original en formato RGB se pasa como ***imagenRGB***, la imagen generada se almacena en imagenFiltradaRGB, la ***anchura*** y ***altura*** de la imagen se pasan en segundo y tercer lugar.

Este algoritmo ha sido tomado y adaptado de [XVJ94]. Los valores del gradiente calculados con la máscara son el cuádruple de los obtenidos con la fórmula $\frac{\Delta f(x)}{\Delta x}$. El valor máximo del gradiente es 1024 y raramente se alcanza porque equivalente a un salto del color blanco al negro –poco habitual en una imagen fotorrealista–. Como el gradiente de cada componente de color se almacena en un byte, todos los gradientes superiores a 255 se truncan a 255 –máximo valor que se puede contener en un byte–, ya que si el valor del gradiente se divide por cuatro para que case en un byte sin tener que truncar el valor se pierde precisión.

- ***void doEdgeConvolvGrey(byte *imagenGris, int anchura,***
  int altura, byte * imagenFiltradaGris,
  float RelaciónSeñalRuido,
  int selx, int sely, int selw, int selh);

Aplica el filtro de Sobel para hallar los contornos de la imagen. En realidad realiza la convolución con dos máscaras de detección de contornos (una en dirección horizontal y otra en dirección vertical) y toma el valor absoluto máximo. En los bordes en donde la ventana Sobel tiene puntos fuera de los límites de la imagen, no se detectan contornos. La imagen original en tonos de gris se pasa como ***imagenGris***, la imagen generada se almacena en ***imagenFiltradaGris***, la ***anchura*** y ***altura*** de la imagen se pasan en segundo y tercer lugar.

Las dos máscaras utilizadas para el cálculo del filtro Sobel detectan los contornos en la dirección horizontal y vertical, respectivamente.

```
-1 0 1          -1 -2 -1
```

```
Hor = -2 0 2     y   Vert =  0  0  0
      -1 0 1                 1  2  1
```

A efectos prácticos, se ha utilizado cuatro máscaras para disminuir el tiempo de cálculo del algoritmo:

```
     -1 0 0         0 0 0         0 0 1        0  1 0
a =   0 0 0 ,  b = -1 0 1 ,  c =   0 0 0 ,  d = 0  0 0
      0 0 1         0 0 0        -1 0 0        0 -1 0
```

La componente horizontal y vertical se puede calcular como Hor = a + 2*b + c, Vert = a - c - 2*d. El módulo del gradiente de la imagen se obtiene calculando el módulo de las dos componentes Sobel. En esta función se ha utilizado el módulo máximo, es decir:

```
gradiente = max(abs(Hor),abs(Vert)).
```

En los puntos contiguos a los bordes directamente se toma el gradiente nulo, ya que la ventana Sobel tiene puntos fuera de la imagen y sería necesario establecer algún tipo de orlado (por ejemplo, el orlado especular) para calcular el gradiente. El valor máximo del gradiente es 1024 y raramente se alcanza porque equivalente a un salto del color blanco al negro –poco habitual en una imagen fotorrealista–. Como el gradiente de cada componente de color se almacena en un byte, todos los gradientes superiores a 255 se truncan a 255 –máximo valor que se puede contener en un byte–, ya que si el valor del gradiente se divide por cuatro para que case en un byte sin tener que truncar el valor se pierde precisión. Este algoritmo ha sido adaptado de [XVJ94].

- ***`void GradientRoberts (void);`***

Aplica el filtro de Roberts para hallar los contornos de la imagen. En realidad realiza la convolución con dos máscaras de detección y calcula el módulo según la fórmula escogida de las cuatro posibles. En los bordes en donde la ventana Roberts tiene puntos fuera de los límites de la imagen, se utiliza el orlado especular para calcular el gradiente. La convolución de las máscaras con la imagen lo realiza la función *`doRobGradient24`* o *`doRobGradientGrey`*, dependiendo de si la imagen es en

color o en tonos de gris, pasando los parámetros correspondientes a la imagen contenida en la clase `IOfile`.

- ***void doRobGradient24 (byte *imagenRGB, byte * imagenFiltradaRGB, char tipo, int anchura, int altura);***

Aplica el filtro de Roberts para hallar los contornos de la imagen. En realidad realiza la convolución con dos máscaras de detección de contornos para las componentes roja, verde y azul y se toma el módulo del gradiente para cada componente. El valor del gradiente obtenido de la componente roja se guarda en la componente roja, el valor del gradiente de la componente verde se guarda en la componente verde y lo mismo para la componente azul. En los bordes en donde la ventana Roberts tiene puntos fuera de los límites de la imagen, se utiliza el orlado especular para calcular el gradiente.

Las dos máscaras utilizadas para el cálculo del filtro Roberts detectan los contornos en la dirección diagonal.

$$G1 = \begin{matrix} -1 & 0 \\ 0 & 1 \end{matrix} \qquad G2 = \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}$$

El gradiente de la imagen se obtiene calculando el módulo de las dos componentes Roberts. La fórmula utilizada para calcular el módulo se elige mediante el argumento de la función ***tipo***.

	Cálculo del módulo
Tipo=='A'	$\sqrt{\frac{G1 \cdot G1 + G2 \cdot G2}{2}}$
Tipo=='B'	Max {|G1|,|G2|}
Tipo=='C'	$\sqrt{0'9 \cdot max\{\|G1\|,\|G2\|\} + 0.1 \cdot media\{\|G1\|,\|G2\|\}}$
Tipo=='D'	$0.4 \cdot max\{\|G1\|,\|G2\|\} + 0.6 \cdot media\{\|G1\|,\|G2\|\}$

La imagen original en formato RGB se pasa como ***imagenRGB***, la imagen generada con el ruido se almacena en ***imagenFiltradaRGB***, , en ***tipo*** se indica el tipo de fórmula para calcular el módulo del gradiente y la ***anchura*** y ***altura*** de la

imagen se pasan en último lugar. Este algoritmo ha sido adaptado a imágenes en color a partir de [MAM95].

- ***void doRobGradientGrey(byte *imagenGris, byte * imagenFiltradaGris, char tipo, int anchura, int altura);***

Aplica el filtro de Roberts para hallar los contornos de la imagen, utilizando dos máscaras de detección de contornos. En los bordes en donde la ventana Roberts tiene puntos fuera de los límites de la imagen, se utiliza el orlado especular para calcular el gradiente.

Las dos máscaras utilizadas para el cálculo del filtro Roberts detectan los contornos en la dirección diagonal.

$$G1 = \begin{matrix} -1 & 0 \\ 0 & 1 \end{matrix} \qquad G2 = \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}$$

El gradiente de la imagen se obtiene calculando el módulo de las dos componentes Roberts. La fórmula utilizada para calcular el módulo se elige mediante el argumento de la función *tipo.*

	Cálculo del módulo
Tipo=='A'	$\sqrt{\frac{G1 \cdot G1 + G2 \cdot G2}{2}}$
Tipo=='B'	Max {|G1|,|G2|}
Tipo=='C'	$\sqrt{0'9 \cdot max\{\|G1\|,\|G2\|\} + 0.1 \cdot media\{\|G1\|,\|G2\|\}}$
Tipo=='D'	$0.4 \cdot max\{\|G1\|,\|G2\|\} + 0.6 \cdot media\{\|G1\|,\|G2\|\}$

La imagen original en tonos de gris se pasa como ***imagenGris***, la imagen generada con el ruido se almacena en ***imagenFiltradaGris***, en ***tipo*** se indica el tipo de fórmula para calcular el módulo del gradiente y la ***anchura*** y ***altura*** de la imagen se pasan en último lugar. Este algoritmo ha sido tomado de [MAM95].

- ***void AlgCanny (void);***

Aplica el algoritmo de Canny para hallar los contornos de la imagen. El algoritmo pregunta al usuario el parámetro sigma correspondiente al filtrado gausiano de la imagen, el umbral mínimo y máximo para la detección de contornos, la longitud mínima de los segmentos hallados.

Básicamente, este algoritmo pasa un filtro gausiano por la imagen, calcula los contornos de la imagen filtrada a través de la ventana Roberts. Si la imagen es en color, el valor del gradiente de las componentes roja, verde y azul se ponderan para obtener un único gradiente de la imagen. A continuación se eliminan los puntos cuyo gradiente no presente un máximo local o cuyo valor del gradiente sea menor que el umbral mínimo (el resto de puntos se consideran puntos contorno). Los puntos contorno contiguos se pueden agrupan en segmentos para eliminar los que tengan una longitud menor que la mínima.

En este algoritmo han sido codificados tres tipos de agrupamientos de puntos contorno en segmentos: no utilizar reagrupamiento; utilizar agrupamiento buscando algún “vecino 8” que sea punto contorno; y utilizar agrupamiento buscando los cuatro puntos de los ocho posibles, más alineados con el contorno. Cuando se utiliza el agrupamiento, se empieza de un punto contorno –cuyo gradiente supere el valor del umbral máximo– y se busca todos los puntos con vecindad ocho o agrupar los puntos de vecindad ocho que están en dirección perpendicular del gradiente. Si sólo se busca contornos en cuatro puntos en vez de ocho, se desestiman los vecinos que no siguen una línea de contorno suave, y por tanto los contornos con radios de curvatura muy pequeños se fragmentan, como los producidos por el ruido o por objetos pequeños.

Los puntos contorno hallados tienen una luminosidad proporcional al valor del gradiente de la imagen en el punto detectado. De esta forma es fácil eliminar los puntos contorno que corresponden al ruido presente en la imagen.

Esta función utiliza las funciones *doCanny24* y *doCannyGrey* (dependiendo de si la imagen contenida en memoria es en color o en tonos de gris, respectivamente) para efectuar el algoritmo. Para obtener más detalles de este algoritmo debe consultarse la documentación original [CAN83] y los ficheros fuente "canny.cpp" y "chany32.c" para la implementación específica.

- ***byte* doCanny24 (byte *imagenRGB, int anchura,***
  int altura, float sigma, int UmbralMinimo,

```
int UmbralMaximo, int LongitudMinima,
int TipoAgrupamientoContornos, int pasoApaso);
```

Esta función aplica el algoritmo de Canny a una imagen en tonos de gris. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenRGB*; la *anchura* y *altura* de la misma se especifican a continuación; el parámetro *sigma* se refiere al valor de suavizado del filtrado gausiano que se aplica a la imagen previo a la extracción de contornos; en *UmbralMinimo* se especifica el valor del gradiente mínimo para que un punto pueda ser contorno; en *UmbralMaximo* se especifica el valor del gradiente a partir cual se inicia la agrupación de puntos para formar *segmentos* –en adelante se utilizará el termino de segmentos para referirse a una agrupación de puntos contorno contiguos, sin necesidad de que formen un contorno asimilable a una recta–; en *LongitudMinima* se indica el mínimo número de puntos que han de tener los segmentos para considerarlos contornos válidos; en *TipoAgrupamientoContornos* se especifica la estrategia para agrupar los puntos contorno en segmentos –este argumento puede tomar el valor cero para indicar que no se produzca agrupamiento, uno para que se utilice el agrupamiento clásico de vecindad ocho, o dos para utilizar una variante del agrupamiento de vecindad ocho en el cual sólo se buscan los vecinos que siguen la dirección del contorno–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

Si el argumento *TipoAgrupamientoContornos* es cero no se produce agrupamiento de puntos contorno en segmentos y, por tanto, no tiene sentido hablar de *UmbralMaximo* ni *LongitudMinima*. Por su parte, si *UmbralMaximo* es menor o igual que *UmbralMinimo*, no tiene efecto alguno el valor.

Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralMinimo* para eliminar el ruido como posible punto contorno, con *UmbralMaximo* se ha de escoger la importancia de los contornos que se quieren obtener, con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises correspondientes al valor del gradiente –obtenido

ponderando el resultado del gradiente de las componentes roja, verde y azul– de los puntos contornos hallados.

- ***byte* doCannyGrey (byte *imagenGris, int anchura, int altura, float sigma, int UmbralMinimo, int UmbralMaximo, int LongitudMinima, int TipoAgrupamientoContornos, int pasoApaso);***

Esta función aplica el algoritmo de Canny a una imagen en tonos de gris. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenGris*; la *anchura* y *altura* de la misma se especifican a continuación; el parámetro *sigma* se refiere al valor de suavizado del filtrado gausiano que se aplica a la imagen previo a la extracción de contornos; en *UmbralMinimo* se especifica el valor del gradiente mínimo para que un punto pueda ser contorno; en *UmbralMaximo* se especifica el valor del gradiente a partir cual se inicia la agrupación de puntos para formar *segmentos* –en adelante se utilizará el termino de segmentos para referirse a una agrupación de puntos contorno contiguos, sin necesidad de que formen un contorno asimilable a una recta–; en *LongitudMinima* se indica el mínimo número de puntos que han de tener los segmentos para considerarlos contornos válidos; en *TipoAgrupamientoContornos* se especifica la estrategia para agrupar los puntos contorno en segmentos –este argumento puede tomar el valor cero para indicar que no se produzca agrupamiento, uno para que se utilice el agrupamiento clásico de vecindad ocho, o dos para utilizar una variante del agrupamiento de vecindad ocho en el cual sólo se buscan los vecinos que siguen la dirección del contorno–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

Si el argumento *TipoAgrupamientoContornos* es cero no se produce agrupamiento de puntos contorno en segmentos y, por tanto, no tiene sentido hablar de *UmbralMaximo* ni *LongitudMinima*. Por su parte, si *UmbralMaximo* es menor o igual que *UmbralMinimo*, no tiene efecto alguno el valor.

Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralMinimo* para eliminar el ruido como posible punto contorno, con *UmbralMaximo* se ha de escoger la importancia de los contornos que se quieren

obtener, con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises correspondientes al valor del gradiente de los puntos contornos hallados.

- ***void Multires (int MultiresType);***

Esta función aplica el algoritmo de extracción de contornos multirresolución realizado en [MAM95]. El algoritmo se ha modificado para poder tratar imágenes de cualquier dimensión, en tonos de gris o en color, y se ha modificado la implementación para aumentar la velocidad de procesamiento. Con el parámetro *MultiresType* se elige el tipo de algoritmo que se quiere aplicar: si *MultiresType*== ALG_MULTIRES, se utiliza un filtro gausiano en las diversas escalas multirresolución detallado en [MAM95]; si *MultiresType*== ALG_FASTMRES se utiliza una versión acelerada del algoritmo, que entre otras modificaciones utiliza el filtrado iterativo en vez del filtrado gausiano; si *MultiresType*== ALG_CANYMRES se utiliza el algoritmo Canny multirresolución que se especifica en el siguiente capítulo del proyecto. Para poder ajustar de forma completa los parámetros del algoritmo Canny multirresolución, se debe utilizar la función *CannyMRes*(), que se explica más adelante, en vez de *Multires*(ALG_ CANYMRES*)*.

Esta función pide al usuario los parámetros ajustables de los algoritmos y llama a las funciones *doGaussMultires24*, *doGaussMultiresGrey*, *doFastMultires24*, *doFastMultiresGrey*, *doCannyMultires24* o *doCannyMultiresGrey* para que realicen el procesamiento de la imagen contenida en memoria —la función que se llama depende del parámetro *MultiresType* y de si la imagen es en color o en tonos de gris.

- ***byte* doGaussMultires24 (byte *imagenRGB, int anchura,***
  int altura, int NivelMinimo, int NivelMaximo,
  int UmbralGradiente, float PorcenReclasificacion,
  int LongitudMinima, int* TipoReclasificacion = *0,

```
int pasoApaso = 0);
```

Esta función aplica el algoritmo multirresolución [MAM95] a una imagen en color. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenRGB*; la *anchura* y *altura* de la misma se especifican a continuación; en *NivelMinimo* se especifica la escala a la cual ha de aparecer un punto contorno para diferenciarlo de puntos ruidosos(este valor debe estar entre 1 y cinco); el parámetro *NivelMaximo* se refiere a la mayor escala del análisis (este valor debe estar entre dos y seis, y debe ser mayor o igual que *NivelMinimo*); en *UmbralGradiente* se especifica el valor del gradiente mínimo de los puntos contorno; para que todos los puntos de un segmento se califiquen de un tipo escalón, escalera, rampa o pulso, el porcentaje de puntos de un mismo tipo que aparecen en un segmento debe ser mayor que *PorcenReclasificacion*; en *LongitudMinima* se indica el mínimo número de puntos han de tener los segmentos para considerarlos contornos válidos; en *TipoReclasificacion* se especifica la estrategia para agrupar los puntos contorno en segmentos –este argumento puede tomar el valor cero para indicar que no se produzca agrupamiento, uno para que se utilice el agrupamiento clásico de vecindad ocho, o dos para utilizar una variante del agrupamiento de vecindad ocho en el cual sólo se buscan los vecinos que siguen la dirección del contorno–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

Básicamente este algoritmo sólo valida los puntos contorno que aparecen en la imagen original que además tengan un gradiente en los niveles uno a *NivelMinimo* mayor o igual que *UmbralGradiente*. El parámetro *NivelMaximo* sólo afecta a la clasificación de contornos, y debe ser seis para que la calificación sea correcta. Si *TipoReclasificacion* es distinto de cero, se unifica la clasificación de los puntos que forman un contorno, dándoles la clasificación del tipo más abundante si llega al porcentaje *PorcenReclasificacion*. Si *TipoReclasificacion* es distinto de cero se desechan los contornos que contengan menos de *LongitudMinima* pixels.

Si el argumento *TipoAgrupamientoContornos* es cero no se produce agrupamiento de puntos contorno en segmentos y, por tanto, no tiene sentido hablar de *LongitudMinima* ni *PorcenReclasificacion*. Por su parte, si

NivelMinimo debe ser menor o igual que *NivelMaximo*. Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralGradiente* para eliminar la posibilidad de que la presencia de ruido en la imagen produzca contornos falsos. Con *NivelMinimo* se ha de escoger la importancia de los contornos que se quieren obtener y con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises correspondientes a las clasificaciones de los puntos. El valor de cada pixel indica su clasificación: cero para indicar la ausencia de contorno, *INDETERMINADO* para los puntos contorno que no se han ajustado a ninguna clasificación, *ESCALERA* para los puntos contorno clasificados como tipo escalera, *RAMPA* para los puntos contorno clasificados como rampa y *ESCALON* para los puntos ajustados al tipo escalón.

- ***byte* doGaussMultiresGrey (byte *imagenGris, int anchura, int altura, int NivelMinimo, int NivelMaximo, int UmbralGradiente, float PorcenReclasificacion, int LongitudMinima, int* TipoReclasificacion = *0, int pasoApaso = 0);***

Esta función aplica el algoritmo multirresolución [MAM95] a una imagen en tonos de gris. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenGris*; la *anchura* y *altura* de la misma se especifican a continuación; en *NivelMinimo* se especifica la escala a la cual ha de aparecer un punto contorno para diferenciarlo de puntos ruidosos(este valor debe estar entre 1 y cinco); el parámetro *NivelMaximo* se refiere a la mayor escala del análisis (este valor debe estar entre dos y seis, y debe ser mayor o igual que *NivelMinimo*); en *UmbralGradiente* se especifica el valor del gradiente mínimo de los puntos contorno; para que todos los puntos de un segmento se califiquen de un tipo escalón, escalera, rampa o pulso, el porcentaje de puntos de un mismo tipo que aparecen en un segmento debe ser mayor que *PorcenReclasificacion*; en *LongitudMinima* se indica el mínimo número de puntos han de tener los segmentos para considerarlos contornos válidos; en *TipoReclasificacion* se especifica la estrategia para

agrupar los puntos contorno en segmentos –este argumento puede tomar el valor cero para indicar que no se produzca agrupamiento, uno para que se utilice el agrupamiento clásico de vecindad ocho, o dos para utilizar una variante del agrupamiento de vecindad ocho en el cual sólo se buscan los vecinos que siguen la dirección del contorno–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

Básicamente este algoritmo sólo valida los puntos contorno que aparecen en la imagen original que además tengan un gradiente en los niveles uno a *NivelMinimo* mayor o igual que *UmbralGradiente*. El parámetro *NivelMaximo* sólo afecta a la clasificación de contornos, y debe ser seis para que la calificación sea correcta. Si *TipoReclasificacion* es distinto de cero, se unifica la clasificación de los puntos que forman un contorno, dándoles la clasificación del tipo más abundante si llega al porcentaje *PorcenReclasificacion*. Si *TipoReclasificacion* es distinto de cero se desechan los contornos que contengan menos de *LongitudMinima* pixels.

Si el argumento *TipoAgrupamientoContornos* es cero no se produce agrupamiento de puntos contorno en segmentos y, por tanto, no tiene sentido hablar de *LongitudMinima* ni *PorcenReclasificacion*. Por su parte, *NivelMinimo* debe ser menor o igual que *NivelMaximo*. Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralGradiente* para eliminar la posibilidad de que la presencia de ruido en la imagen produzca contornos falsos. Con *NivelMinimo* se ha de escoger la importancia de los contornos que se quieren obtener y con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises correspondientes a las clasificaciones de los puntos. El valor de cada pixel indica su clasificación: cero para indicar la ausencia de contorno, *INDETER-MINADO* para los puntos contorno que no se han ajustado a ninguna clasificación, *ESCALERA* para los puntos contorno clasificados como tipo escalera, *RAMPA* para los puntos contorno clasificados como rampa y *ESCALON* para los puntos ajustados al tipo escalón.

- ***byte* doFastMultires24 (byte *imagenRGB, int anchura,***

```
int altura, int NivelMinimo, int NivelMaximo,
int UmbralGradiente, float PorcenReclasificacion,
int LongitudMinima, int TipoReclasificacion = 0,
int pasoApaso = 0);
```

Esta función aplica una variante del algoritmo multirresolución presentado en [MAM95] que se diferencia en la utilización de un filtro iterativo en vez del filtro gausiano, y en otras mejoras que aumentan significativamente la velocidad de ejecución.. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenRGB*; la *anchura* y *altura* de la misma se especifican a continuación; en *NivelMinimo* se especifica la escala a la cual ha de aparecer un punto contorno para diferenciarlo de puntos ruidosos (este valor debe estar entre 1 y cinco); el parámetro *NivelMaximo* se refiere a la mayor escala del análisis (este valor debe estar entre dos y seis, y debe ser mayor o igual que *NivelMinimo*); en *UmbralGradiente* se especifica el valor del gradiente mínimo de los puntos contorno; para que todos los puntos de un segmento se califiquen de un tipo escalón, escalera, rampa o pulso, el porcentaje de puntos de un mismo tipo que aparecen en un segmento debe ser mayor que *PorcenReclasificacion*; en *LongitudMinima* se indica el mínimo número de puntos han de tener los segmentos para considerarlos contornos válidos; en *TipoReclasificacion* se especifica la estrategia para agrupar los puntos contorno en segmentos –este argumento puede tomar el valor cero para indicar que no se produzca agrupamiento, uno para que se utilice el agrupamiento clásico de vecindad ocho, o dos para utilizar una variante del agrupamiento de vecindad ocho en el cual sólo se buscan los vecinos que siguen la dirección del contorno–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

Básicamente este algoritmo sólo valida los puntos contorno que aparecen en la imagen original que además tengan un gradiente en los niveles uno a *NivelMinimo* mayor o igual que *UmbralGradiente*. El parámetro *NivelMaximo* sólo afecta a la clasificación de contornos, y debe ser seis para que la calificación sea correcta. Si *TipoReclasificacion* es distinto de cero, se unifica la clasificación de los puntos que forman un contorno, dándoles la clasificación del tipo más abundante si llega al porcentaje *PorcenReclasificacion*. Si *TipoReclasificacion* es distinto

de cero se desechan los contornos que contengan menos de *LongitudMinima* pixels.

Si el argumento *TipoAgrupamientoContornos* es cero no se produce agrupamiento de puntos contorno en segmentos y, por tanto, no tiene sentido hablar de *LongitudMinima* ni *PorcenReclasificacion*. Por su parte, *NivelMinimo* debe ser menor o igual que *NivelMaximo*. Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralGradiente* para eliminar la posibilidad de que la presencia de ruido en la imagen produzca contornos falsos. Con *NivelMinimo* se ha de escoger la importancia de los contornos que se quieren obtener y con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises correspondientes a las clasificaciones de los puntos. El valor de cada pixel indica su clasificación: cero para indicar la ausencia de contorno, *INDETERMINADO* para los puntos contorno que no se han ajustado a ninguna clasificación, *ESCALERA* para los puntos contorno clasificados como tipo escalera, *RAMPA* para los puntos contorno clasificados como rampa y *ESCALON* para los puntos ajustados al tipo escalón.

- ***byte* doFastMultiresGrey (byte *imagenGris, int anchura,***
  int altura, int NivelMinimo, int NivelMaximo,
  int UmbralGradiente, float PorcenReclasificacion,
  int LongitudMinima, int* TipoReclasificacion = *0,
  int pasoApaso = 0);

Esta función aplica una variante del algoritmo multirresolución presentado en [MAM95] que se diferencia en la utilización de un filtro iterativo en vez del filtro gausiano, y en otras mejoras que aumentan significativamente la velocidad de ejecución. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenGris*; la *anchura* y *altura* de la misma se especifican a continuación; en *NivelMinimo* se especifica la escala a la cual ha de aparecer un punto contorno para diferenciarlo de puntos ruidosos(este valor debe estar entre 1 y cinco); el parámetro *NivelMaximo* se refiere a la mayor escala del análisis (este valor debe estar entre dos

y seis, y debe ser mayor o igual que *NivelMinimo*); en *UmbralGradiente* se especifica el valor del gradiente mínimo de los puntos contorno; para que todos los puntos de un segmento se califiquen de un tipo escalón, escalera, rampa o pulso, el porcentaje de puntos de un mismo tipo que aparecen en un segmento debe ser mayor que *PorcenReclasificacion*; en *LongitudMinima* se indica el mínimo número de puntos han de tener los segmentos para considerarlos contornos válidos; en *TipoReclasificacion* se especifica la estrategia para agrupar los puntos contorno en segmentos –este argumento puede tomar el valor cero para indicar que no se produzca agrupamiento, uno para que se utilice el agrupamiento clásico de vecindad ocho, o dos para utilizar una variante del agrupamiento de vecindad ocho en el cual sólo se buscan los vecinos que siguen la dirección del contorno–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

Básicamente este algoritmo sólo valida los puntos contorno que aparecen en la imagen original que además tengan un gradiente en los niveles uno a *NivelMinimo* mayor o igual que *UmbralGradiente*. El parámetro *NivelMaximo* sólo afecta a la clasificación de contornos, y debe ser seis para que la calificación sea correcta. Si *TipoReclasificacion* es distinto de cero, se unifica la clasificación de los puntos que forman un contorno, dándoles la clasificación del tipo más abundante si llega al porcentaje *PorcenReclasificacion*. Si *TipoReclasificacion* es distinto de cero se desechan los contornos que contengan menos de *LongitudMinima* pixels.

Si el argumento *TipoAgrupamientoContornos* es cero no se produce agrupamiento de puntos contorno en segmentos y, por tanto, no tiene sentido hablar de *LongitudMinima* ni *PorcenReclasificacion*. Por su parte, *NivelMinimo* debe ser menor o igual que *NivelMaximo*. Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralGradiente* para eliminar la posibilidad de que la presencia de ruido en la imagen produzca contornos falsos. Con *NivelMinimo* se ha de escoger la importancia de los contornos que se quieren obtener y con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises correspondientes a las clasificaciones de los puntos. El valor de

cada pixel indica su clasificación: cero para indicar la ausencia de contorno, INDETERMINADO para los puntos contorno que no se han ajustado a ninguna clasificación, ESCALERA para los puntos contorno clasificados como tipo escalera, RAMPA para los puntos contorno clasificados como rampa y ESCALON para los puntos ajustados al tipo escalón.

- ***void CannyMRes (void);***

Esta función aplica el algoritmo de extracción de contornos Canny multirresolución que se detalla en el capítulo siete. Esta función pide al usuario los parámetros ajustables de los algoritmos y llama a las funciones *doCannyMultires24* o *doCannyMultiresGrey* para que realicen el procesamiento de la imagen contenida en memoria –dependiendo de si la imagen es en color o en tonos de gris–.

- ***byte* doCannyMultires24 (byte *imagenRGB, int anchura,***
  int altura, int NivelMinimo, int NivelMaximo,
  int UmbralGradiente, int NivelRuidoEnGradiente,
  float PorcenReclasificacion, int LongitudMinima,
  int OrdenCasamiento = 0, int pasoApaso = 0);

Esta función aplica una variante del algoritmo Canny que se detalla en el capítulo siete. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenRGB*; la *anchura* y *altura* de la misma se especifican a continuación; en *NivelMinimo* se especifica la escala a la cual ha de aparecer un punto contorno para diferenciarlo de puntos ruidosos(este valor debe estar entre 1 y cinco); el parámetro *NivelMaximo* se refiere a la mayor escala del análisis (este valor debe estar entre dos y seis, y a su vez debe ser mayor o igual que *NivelMinimo*); en *UmbralGradiente* se especifica el valor mínimo del gradiente para que se empiece a buscar un contorno a partir de dicho punto; en *NivelRuidoEnGradiente* se especifica el valor medio del ruido en el gradiente, que suele ser algunas veces menor que *UmbralGradiente*; el porcentaje mínimo de coincidencia entre los contornos correspondientes a dos escalas consecutivas es *PorcenReclasificacion*; en *LongitudMinima* se indica el mínimo número de puntos han de tener los segmentos

para considerarlos contornos válidos; en *OrdenCasamiento* se especifica el orden de casamiento –si el valor es cero el orden es descendente, si es distinto de cero el orden es ascendente–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

NivelMinimo debe ser menor o igual que *NivelMaximo*. Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralGradiente* para eliminar la posibilidad de que la presencia de ruido en la imagen produzca contornos falsos. Con *NivelMinimo* se ha de escoger la importancia de los contornos que se quieren obtener y con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises proporcionalmente a la importancia del contorno(un color gris oscuro indica un contorno menos relevante que uno blanco brillante).

- ***byte* doCannyMultiresGrey (byte *imagenGris, int anchura, int altura, int NivelMinimo, int NivelMaximo, int UmbralGradiente, int NivelRuidoEnGradiente, float PorcenReclasificacion, int LongitudMinima, int OrdenCasamiento = 0, int pasoApaso = 0);***

Esta función aplica una variante del algoritmo Canny que se detalla en el capítulo siete. Los parámetros que toma la función son por orden: la imagen original se pasa en *imagenGris*; la *anchura* y *altura* de la misma se especifican a continuación; en *NivelMinimo* se especifica la escala a la cual ha de aparecer un punto contorno para diferenciarlo de puntos ruidosos(este valor debe estar entre 1 y cinco); el parámetro *NivelMaximo* se refiere a la mayor escala del análisis (este valor debe estar entre dos y seis, y a su vez debe ser mayor o igual que *NivelMinimo*); en *UmbralGradiente* se especifica el valor mínimo del gradiente para que se empiece a buscar un contorno a partir de dicho punto; en *NivelRuidoEnGradiente* se especifica el valor medio del ruido en el gradiente, que suele ser algunas veces menor que *UmbralGradiente*; el porcentaje mínimo de coincidencia entre los contornos correspondientes a dos escalas consecutivas es *PorcenReclasificacion*; en

LongitudMinima se indica el mínimo número de puntos han de tener los segmentos para considerarlos contornos válidos; en *OrdenCasamiento* se especifica el orden de casamiento –si el valor es cero el orden es descendente, si es distinto de cero el orden es ascendente–; y por último si *pasoApaso* es distinto de cero se muestra paso a paso las operaciones que realiza el algoritmo.

NivelMinimo debe ser menor o igual que *NivelMaximo*. Para obtener un resultado óptimo se ha de escoger el parámetro del *UmbralGradiente* para eliminar la posibilidad de que la presencia de ruido en la imagen produzca contornos falsos. Con *NivelMinimo* se ha de escoger la importancia de los contornos que se quieren obtener y con *LongitudMinima* se eliminan los contornos que tienen pocos puntos y que carecen de interés práctico.

Esta función retorna un puntero a una imagen de dimensión *anchura* por *altura*, con 256 tonos de grises proporcionalmente a la importancia del contorno(un color gris oscuro indica un contorno menos relevante que uno blanco brillante).

BIBLIOGRAFÍA

[MAM95] Miguel Angel Martínez. Análisis Multirresolución por gausianas para la detección y caracterización de contornos en imágenes 2-D. Centro Politécnico Universidad de Zaragoza. Mayo de 1995.

[JRB94] J.R. Beltrán. Detección y clasificación de contornos, en imágenes digitalizadas, utilizando la transformada de Wavelet. Centro Politécnico Universidad de Zaragoza. Octubre 1994.

[CAN83] J.F. Canny. Finding edges and lines. Technical Report 720. M.I.T. 1983.

[CAN86] J.F. Canny. A computational approach to edge detection. IEEE Transations on Pattern Analysis and Machine Intelligence, 8:679-698, Noviembre 1988.

[XVJ94] John Bradley, 1989-94. "XV User manual". 1053 Floyd Terrace. Bryn Mawr, PA 19010

[IJG94] Thomas G. Lane. "IJG Group Library". Independent JPEG Group's software, 1994.

[TAR97] Varios autores. "TargetJunior User's manual" Robotics Research Group, Oxford University. Disponible en la dirección URL `robots.oxford.ac.uk.`

[WIR76] Niklaus Wirth. "Algoritmos + Estructuras de datos = Programas". Ediciones del Castillo.

[STR76] G. Strang. "Algebra lineal y sus aplicaciones". Ed. Addison-Wesley Iberoamericana.

[CAN76] George C. Canavos. "Probabilidad y Estadística. Aplicaciones y Métodos". Ed. McGraw-Hill.

[HOZ90] Holzmann. "Beyond Photography", capítulo 7.

[REC92] Press, William H., Saul A. Teukolsky, William T. Vettering, and Brian P. Flannery. "Numerical Recipes in C", Second Edition, Cambridge University Press, 1992.

ÍNDICE TEMÁTICO DEL ANEXO B

B.1 FUNCIONES GENERALES 2

DoAlg 2

B.1.1 Manipulaciones básicas 4

RestorePic 4
CropPic 5
XVGetSubImage 5
ResizePic 6
Smooth24 6
SmoothResize 7
DoColorDither 7
Do332ColorDither 8
InvertPic 9
RotatePic 9
FlipPic 10

B.1.2 Ajuste de la curva gamma 10

saturePic24 10
saturePicGrey 11
EvalSpline 12
InitSpline 12
GenerateFSGamma 12
FSDither 12
GammifyPic24 13
SortSpline 14
intshort 14

B.1.3 Efectos sobre las imágenes 14

TinFoil 14
doAngleConvolv 14
OilPaint 15
doOilPaint 15
Blend 16
doBlend 16
FineRotate 17
doRotate 17
Pixelize 18
doPixel 18

B.1.4 Filtrados 18

Blur 18
doBlurConvolv24 19
doBlurConvolvGrey 19
GaussianBlur 20
doGaussianConv24 20
doGaussianConvGrey 20
Sharpen 21

doSharpConvolv 21
MedianFilter 21
doMedianFilter 22

B.1.5 Adición de ruido a la imagen 22

Spread 22
doSpread 22
GaussianNoise 23
doGaussianNoise24 24
doGaussianNoiseGrey 24

B.1.6 Extracción de contornos 24

SobelGradient 25
doEdgeConvolv24 25
doEdgeConvolvGrey 26
GradientRoberts 27
doRobGradient24 28
doRobGradientGrey 28
AlgCanny 29
doCanny24 30
doCannyGrey 31
Multires 33
doGaussMultires24 33
doGaussMultiresGrey 35
doFastMultires24 36
doFastMultiresGrey 38
CannyMRes 40
doCannyMultires24 40
doCannyMultiresGrey 41

Bibliografía 43

ÍNDICE ALFABÉTICO DEL ANEXO B

Adición de ruido a la imagen 22
Ajuste de la curva gamma 10
AlgCanny 29
Blur 18
Blend 16
Bibliografía 43
CannyMRes 40
DoAlg 2
doAngleConvolv 14
doBlend 16
doBlurConvolv24 19
doBlurConvolvGrey 19
doCanny24 30
doCannyGrey 31
doCannyMultires24 40
doCannyMultiresGrey 41
DoColorDither 7
Do332ColorDither 8
doEdgeConvolv24 25
doEdgeConvolvGrey 26
doFastMultires24 36
doFastMultiresGrey 38
doGaussianConv24 20
doGaussianConvGrey 20
doGaussianNoise24 24
doGaussianNoiseGrey 24
doGaussMultires24 33
doGaussMultiresGrey 35
doMedianFilter 22
doOilPaint 15
doPixel 18
doRobGradient24 28
doRobGradientGrey 28
doRotate 17
doSharpConvolv 21
doSpread 22
CropPic 5
Efectos sobre las imágenes 14
EvalSpline 12
Extracción de contornos 24
Filtrados 18
FineRotate 17
FlipPic 10
FSDither 12
Funciones generales 2
GammifyPic24 13
GaussianBlur 20
GaussianNoise 23
GenerateFSGamma 12
GradientRoberts 27
InitSpline 12

intshort 14
InvertPic 9
Manipulaciones básicas 4
MedianFilter 21
Multires 33
OilPaint 15
Pixelize 18
ResizePic 6
RestorePic 4
RotatePic 9
saturePic24 10
saturePicGrey 11
Sharpen 21
SmoothResize 7
Smooth24 6
SobelGradient 25
SortSpline 14
Spread 22
TinFoil 14
XVGetSubImage 5

Anexo C

Ejemplos de utilización del entorno

C.1 INTRODUCCIÓN.

En la memoria se han expuesto los principios que han guiado el diseño del entorno de visión. En los anexos A y B se han presentado las funciones más representativas de las clases `IOfile` e `ImgBasic`, respectivamente.

Sin embargo, es difícil comprender la utilidad sin imaginarla dentro de un trozo de código. Para remediar esta situación, he extraído algunos fragmentos de código del entorno para analizarlos.

Este anexo es informativo y no tiene carácter exhaustivo. Dentro de mis posibilidades, he intentado que el código estuviera lo más comentado posible. Personalmente, me gusta utilizar la técnica de clonación en la programación: a partir de un código que funciona, se copia y se modifica para adaptarlo a las necesidades propias. Por ejemplo, esta técnica me ha sido útil en la programación de la interfaz de ventanas con las clases *MFC*.

C.2 Ejemplo de acceso a los datos de la imagen

En este ejemplo, se verá como se puede acceder a los datos contenidos en las clases IOfile a través de una clase derivada de ella, Inspection.

```
/*****************************************************************************\
*     File:                  E J E M P L O 1 . C P P                          *
*                                                                             *
*     Standard   C++                                                          *
*                                                                             *
\*****************************************************************************/
/*
 * ejemplo1.cpp: main overview
 *
 *  Contains:
 *         void Ejemplo1();
 */

#include "inspect.h"
#include "..\Win32\YesNo.h"


/*****************************************************************************/
void Inspection::Ejemplo(void)
{
  Warning("Esta función está sin implementar");


  // NOTA: Esta función es llamada por la clase CInspectionDoc
  // AppWizard genera una función que se llama cuando se elige
  // en el menú Inspección la opción ejemplo
  // Desde la función que genera AppEstudio, se debe llamar
  // a esta función.

  // Hay dos alternativas:

  // A) Modificando en el fichero Win32/InspDoc.cpp la función
  //    que AppWizard genera para ese eventento de menú:
  //
  // «afx_msg void CInspectionDoc::OnAlgProyA(void)
  // {
  //     Ejemplo(); //Llama a Inspection::Ejemplo
  //
  // }»

  // B) Modificando en el fichero Win32/InspDoc.h la línea
  //    en que aparece la función que AppWizard genera para
  //    ese eventento de menú:
  //
  // afx_msg void OnAlgProyA() {Ejemplo();} //Llama a Inspection::Ejemplo
  //
  //    y a continuación borrar la función CInspectionDoc::OnAlgProyA
  //    del fichero Win32/InspDoc.cpp



  printf("Función Inspection::AlgProyA\n"
         "============================\n");
  printf("La anchura de la imagen es: %d\n",pWIDE);
  printf("La altura de la imagen es: %d\n",pHIGH);
  printf("La imagen es en %s\n",(mono) ? "tonos de gris" : "color");
```

```
// Pregunta al usuario un punto
u_int i, j; // número de fila y columna
do {
printf("Introduce la fila y columna del pixel cuyo valor\n"
       "quieres conocer...\n");
if (scanf("%u %u",&i,&j) != 2)
  printf("No se han leido los dos parámetros.\n");
} while (i<1 || j<1 || i>pHIGH || j>pWIDE);

// El primer elemento de la imagen es el punto (0,0) y no el (1,1)
// por lo que se debe restar una unidad
i--; j--;

byte *thepic;         /* Copy of the formatted original image */
byte *tmpPic;         /* Buffer where store transformed image */

// Pongo el cursor en modo reloj de arena
WaitCursor();

////////////////////////////////////////////////////////////////////////////////

// Calcula la imagen en RGB COLOR
if (start24bitAlg(&thepic, &tmpPic)) {
  printf("¡¡Fallo de la función start24bitAlg!!\n"
         "  (posiblemente al solicitar memoria).\n");
  return;
}

//En thepic tenemos la imagen original
printf("El valor de la intensidad del punto (%i,%i) es R=%i, G=%i, B=%i\n"
       "(0 indica ausencia de esa componente,\n"
       "255 saturación de la componte)\n",
       i+1 /*fila*/,
       j+1 /*columna*/,
       thepic[3*(i*pWIDE+j)+0] /*componente roja*/,
       thepic[3*(i*pWIDE+j)+1] /*componente verde*/,
       thepic[3*(i*pWIDE+j)+2] /*componente azul*/
       );

// La imagen tmpPic está indefinida.
// Como no queremos modificar la imagen, copiaremos la
// imagen original en la nueva (si la imagen original
// era en color, después del algoritmo será en tonos
// de gris.
memmove((char *) tmpPic/*dst*/,(char *) thepic/*src*/,
        (size_t) 3*pWIDE*pHIGH/*len*/);

end24bitAlg(thepic, tmpPic);

////////////////////////////////////////////////////////////////////////////////
// Calcula la imagen en TONOS DE GRIS
if (startGreyAlg(&thepic, &tmpPic)) {
  printf("¡¡Fallo de la función startGreyAlg!!\n"
         "  (posiblemente al solicitar memoria).\n");
  return;
}

//En thepic tenemos la imagen original
printf("El valor de la intensidad del punto (%i,%i) es %i\n"
       "(0 indica color negro, 255 blanco)\n",
       i+1,j+1,thepic[i*pWIDE+j]);


// La imagen tmpPic está indefinida.
// Como no queremos modificar la imagen, copiaremos la
// imagen original en la nueva (si la imagen original
// era en color, después del algoritmo será en tonos
// de gris.
```

```
  memmove((char *) tmpPic/*dst*/,(char *) thepic/*src*/,
          (size_t) pWIDE*pHIGH /*len*/);

  endGreyAlg(thepic, tmpPic);

  if (!mono)
    printf("La función endGreyAlg ha convertido la imagen\n"
           "original de color a tonos de gris\n");

  ////////////////////////////////////////////////////////////////////////////

  // Actualizo la imagen mostrada en la pantalla
  RedrawPic();

  // Pongo el cursor en modo reloj de arena
  ArrowCursor();

  ////////////////////////////////////////////////////////////////////////////
  // Para ver la definición de la función YesNo, mira el fichero "YesNo.h" del
  // directorio Win32
  int result =
    YesNo("Otra forma alternativa de pregunar al usuario es por "
          "medio de cuadros de diálogo.\n"
          "¿Quieres guardar la imagen en fichero?");
  if (result == IDYES)
    Warning("Has pulsado \"Sí\"");

  if (result == IDNO)
    Warning("Has pulsado \"No\"");

  if (result == IDYES) {
    // Pregunta el nuevo fichero
    result =
        YesNo("¿Quieres guardar la imagen en un fichero distinto?");
    result =
      SavePic( (result==IDYES) ? SAVEPIC : REWRITEPIC);
    if (result != 0)
      Warning("¡Fallo al guardar la imagen!");
  }

  result =
    YesNo("Quieres cargar una nueva imagen");
  if (result == IDYES) {
    // Carga una nueva imagen
    result =
      OpenPic(LOADPIC);
    if (result != 0)
      Warning("¡Fallo al guardar la imagen!");
  }

  Warning("Fin de la función AlgProyA");
}
```

C.3 Ejemplo de un algoritmo genérico para color y tonos de gris

El siguiente algoritmo ilustra de forma sencilla, como diseñar un algoritmo para que trabaje con el conjunto de rutinas implementadas en la clase `ImgBasic`. La función *`GaussianBlur`* es llamada cuando el usuario selecciona en el menú la opción de realizar un suavizado gausiano sobre la imagen. También se puede apreciar que las imágenes en color se tratan en este algoritmo como tres componentes de color que se procesan separadamente. En algunos algoritmos, esta estrategia proporciona simplicidad a la implementación.

La función *`doGaussianConvGrey`* realiza un suavizado gausiano sobre imágenes en tonos de grises o sobre componentes del color. El algoritmo *`doGaussianConv24`* , que suaviza imágenes RGB, se ha implementado de dos formas: En la segunda versión se ha utilizado las funciones *`RedComponent`*, *`GreenComponent`*, *`BlueComponent`* y *`RGBtoPIC24`* para manipular de forma sencilla las componentes del color. La primera versión es menos didáctica y muestra cómo se podría haber hecho sin utilizar estas funciones.

```
/*************************************************************************\
*     File:                 G S B L U R . C P P                           *
*                                                                         *
*     Standard   C++                                                      *
*                                                                         *
\*************************************************************************/
/*
 * gsblur.cpp: gaussian blur filter
 *
 *  Contains:
 *         void GaussianBlur();        // Blur a image !
 *         void doGaussianConv24();    // Color implementation
 *         void doGaussianConvGrey();  // Greyscale implementation
 */

#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "image.h"
#include "miguelan.h"

/*************************************************************************/
```

```
void ImgBasic::GaussianBlur(void)
{
  /* This funcition calls doGaussianConv24 if the image is coloured  *
   * or doGaussianConvGrey if the image is monocrome                 */

  byte *thepic;         /* Copy of the formatted original image */
  byte *tmpPic;         /* Buffer where store transformed image */
  char buf[64] = "3";   /* Buffer for user input                */
  int  i;               /* Counter                              */
  float sigma;          /* Gaussian parameter                   */
  static char *labels[] = { "Ok" }; /* Button for user input    */

  /* Ask the value of the parameter to the user                 *
   * Only alow numbers and decimal point: "0123456789."         */
  i = GetStrPopUp("Filtro de suavizado gausiano:"
                  "\nIntroduce la característica sigma",
                  labels, 1, buf, 64, "0123456789.", 1);

  /* Check for right values                                      */
  if (strlen(buf)==0) {
    Error("El formato del parametro no es correcto");
    return;
  }
  sigma = atof(buf);
  if (sigma<0.1) {
    Error("El valor de sigma es muy pequeno (no se produce suavizado)");
    return;
  }

  /* Call a function depending of the color information */

if (mono) { // MONOCHROME IMAGE

    /* Get a standard greyscale version of the real image    */
    if (startGreyAlg(&thepic, &tmpPic))
       return;      /* Failed to get the greyscale image    */

    /* Call the actual algorithm for greyscale images        */
    doGaussianConvGrey(thepic, tmpPic, pWIDE, pHIGH, sigma);

    /* Store new image, allow 'undo', uptade display, etc.   */
    endGreyAlg(thepic, tmpPic);

  } else {  // COLOR IMAGE

    /* Get a standard 24-bit RGB version of the real image   */
    if (start24bitAlg(&thepic, &tmpPic))
           return; /* Failed to get the RGB-24bit image         */

    /* Call the actual algorithm for color images            */
    doGaussianConv24(thepic, tmpPic, pWIDE, pHIGH, sigma);

    /* Store new image, allow 'undo', uptade display, etc.   */
    end24bitAlg(thepic, tmpPic);
  }

}   /* End of GaussianBlur image                             */


/*********************************************************************/
void ImgBasic::doGaussianConv24   ( byte *pic24, byte *results,
                                    int w, int h,float sigma )
{
  register int i;
  register byte *src, *dst;
  byte *aux_src, *aux_dst;

  /* ALLOCATE MEMORY */
```

```
  aux_src = (byte*) malloc( (long)w*(long)h );
  if (!aux_src) {
    Error("Fallo al requerir memoria");
    return;
  }
  aux_dst = (byte*) malloc( (long)w*(long)h );
  if (!aux_dst) {
    free(aux_src);
    Error("Fallo al requerir memoria");
    return;
  }

  /* BLUR RED COMPONENT! */
  src=pic24; dst=aux_src;
  for(i=w*h ; i; i--, src+=3) *dst++ = *src;
  doGaussianConvGrey(aux_src, aux_dst, w, h, sigma);
  src=aux_dst; dst=results;
  for(i=w*h ; i; i--, dst+=3, src++) *dst = *src;

  /* BLUR GREEN COMPONENT! */
  src=pic24+1; dst=aux_src;
  for(i=w*h ; i; i--, src+=3) *dst++ = *src;
  doGaussianConvGrey(aux_src, aux_dst, w, h, sigma);
  src=aux_dst; dst=results+1;
  for(i=w*h ; i; i--, dst+=3, src++) *dst = *src;

  /* BLUR BLUE COMPONENT! */
  src=pic24+2; dst=aux_src;
  for(i=w*h ; i; i--, src+=3) *dst++ = *src;
  doGaussianConvGrey(aux_src, aux_dst, w, h, sigma);
  src=aux_dst; dst=results+2;
  for(i=w*h ; i; i--, dst+=3, src++) *dst = *src;

  /* FREE MEMORY */
  free(aux_src);
  free(aux_dst);
}


/*********************************************************************/
void ImgBasic::doGaussianConvGrey   ( byte *picGrey, byte *gausiana,
                                      int columnas,int filas, float sigma )
{
  /* Blur grey imagen with gaussian weight: f=exp(-xx/2ss) */
  float far *aux;      /* Auxiliar image for separating components */
  int indice1,indice2,ancho,ancho_max;
  register int posicion1,posicion2;
  float valor_1,factor;
  float gaus[MAX_SIGMA];

  /* Allocate memory */
  aux = (float*) malloc( (long)filas*(long)columnas*sizeof(float) );
  if (!aux) {
    Error("Fallo al requerir memoria");
    return;
  }

  /* width of gaussian window */
  ancho_max=2*sigma+1;
  ancho_max=MIN(ancho_max,MAX_SIGMA);

  /* gaussian weights (only half window because of simetry) */
  gaus[0]=1;
  for ( indice1=1 ; indice1<ancho_max ; indice1++ )
     gaus[indice1]=exp(-indice1*indice1/(2*sigma*sigma));

  /* Scale factor */
  factor=gaus[0];
```

```
  for ( ancho=1 ; ancho<ancho_max ; ancho++)
       factor += 2*gaus[ancho];
  factor *= factor; /* factor^2 => saves fil*col multipications */


  /* BLUR X (calculated with a "mirror image" beyond the border )*/
  for ( indice1=0 ; indice1<filas ; indice1++ ) {
     for ( indice2=0 ; indice2<columnas ; indice2++ ) {
       valor_1=picGrey[indice2+indice1*columnas];
       for ( ancho=1 ; ancho<ancho_max ; ancho++) {
         /* Don't excess limits! */
         posicion1=indice2+ancho;
         posicion1= (posicion1<0)         ? (-posicion1) :
                    (posicion1>=columnas) ? (posicion1-columnas) :
                                            posicion1;
                  posicion2=indice2-ancho;
         posicion2= (posicion2<0)         ? (-posicion2) :
                    (posicion2>=columnas) ? (posicion2-columnas) :
                                            posicion2;
         valor_1 += ( picGrey[posicion1+indice1*columnas] +
                         picGrey[posicion2+indice1*columnas] ) * gaus[ancho];
       }
       aux[indice2 + indice1*columnas]=valor_1;
     }
  }


  /* BLUR Y (calculated with a "mirror image" beyond the border )*/
  for ( indice1=0 ; indice1<columnas ; indice1++ )
     for ( indice2=0 ; indice2<filas ; indice2++ ) {
       valor_1=aux[indice1+indice2*columnas];
       for ( ancho=1 ; ancho<ancho_max ; ancho++) {
         /* Don't excess limits! */
         posicion1 =indice2+ancho;
         posicion1 = (posicion1<0)         ? (-posicion1) :
                     (posicion1>=filas)    ? (posicion1-filas) :
                                             posicion1;
         posicion2 =indice2-ancho;
         posicion2 = (posicion2<0)         ? (-posicion2) :
                     (posicion2>=filas)    ? (posicion2-filas) :
                                             posicion2;
         valor_1 += (aux[indice1+posicion1*columnas] +
                      aux[indice1+posicion2*columnas] ) * gaus[ancho];
       }
       posicion1 = valor_1/factor;
       gausiana [indice1 + indice2*columnas]=(byte)
                           (posicion1 > 255) ? 255 :
                           (posicion1 < 0  ) ? 0   :
                            posicion1;
     }

  /* FREE MEMORY */
  free(aux);
}
/*********************************************************************/
```

- Otra forma equivalente del algoritmo doGaussianConv24 sería:

```
/*********************************************************************/
void ImgBasic::doGaussianConv24   ( byte *pic24, byte *results,
                                    int w, int h,float sigma )
{
byte *component, *Red, *Green, *Blue;

  /* ALLOCATE MEMORY */
  component = (byte*) malloc( w*h );
  if (!component) {
    Error("Fallo al requerir memoria");
    return;
  }

  Red = (byte*) malloc(w*h);
  if (!Red) {
    free(component);
    Error("Fallo al requerir memoria");
    return;
  }

  Green = (byte*) malloc(w*h);
  if (!Green) {
    free(component);
    free(Red);
    Error("Fallo al requerir memoria");
    return;
  }

  Blue = (byte*) malloc(w*h);
  if (!Blue) {
    free(component);
    free(Red);
    free(Green);
    Error("Fallo al requerir memoria");
    return;
  }

  /* BLUR RED COMPONENT! */
  RedComponent(component);
  doGaussianConvGrey(component, Red, w, h, sigma);

  /* BLUR GREEN COMPONENT! */
  GreenComponent(component);
  doGaussianConvGrey(component, Green, w, h, sigma);

  /* BLUR BLUE COMPONENT! */
  BlueComponent(component);
  doGaussianConvGrey(component, Blue, w, h, sigma);

  /* SET NEW IMAGE */
  RGBtoPIC24(Red, Green, Blue);

  free(component);
  free(Red);
  free(Green);
  free(Blue);
}
/****************************** E N D ***************************/
```

C.4 Ejemplo de derivación de una clase de `ImgBasic`

En este capítulo se mostrarán muy rápidamente, todos los ficheros que se han de construir para derivar una clase de ImgBasic, que tenga sus propios menús y recursos. Después de observar por encima las clases necesarias, seguramente llegará a la conclusión de que es mejor ampliar alguna clase existente que crear una nueva. Sin embargo, si quiere disponer de toda la potencia para definir sus propios menús, manejar los eventos de ratón, personalizar la vista mostrada en pantalla, etc. deberá seguir este camino.

- Fichero "*Inspect.h*", que contiene la declaración de los algoritmos de visión (el fichero "*Inspect.cpp*" podría ser el código mostrado en el apartado B.2).

```
#ifndef _INSPECT_H_
#define _INSPECT_H_

#include "..\ImgBasic\ImgBasic.h"

class Inspection : virtual public ImgBasic
{
  /************************** Ejemplo.CPP **************************/
  public:
     virtual void Ejemplo (void);
     //resto de funciones de la clase
};

#endif //#ifndef _INSPECT_H_
```

- Fichero opcional "*InspView.h*", que contiene la definición de la clase vista específica para reprensentar la clase `Inspection`.

```
// InspView.h : interface of the CInspectionView class

#ifndef _INSPECTIONVIEW_H_
#define _INSPECTIONVIEW_H_

#include "ImgView.h"

class CInspectionView : public CImgBasicView
{
protected:
       DECLARE_DYNCREATE(CInspectionView)
};

#endif //#define _INSPECTIONVIEW_H_
```

- Fichero "*InspView.cpp*" que contiene la implementación (vacía) de la clase `CInspectionView`.

```
/ InspView.cpp : implementation of the CInspectionView class
//

#include "stdafx.h"
#include "..\res\resource.h"
#include "InspView.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CImgBasicDoc

IMPLEMENT_DYNCREATE(CInspectionView, CScrollView)
```

- Fichero "*InspDoc.h*", que sirve de puente entre la clase que contiene los algoritmos de inspección, `Inspection`, y la clase que gestiona la representación de imágenes en pantalla `CImgBasicDoc`. Lo más importante de esta clase es que cuando el usuario llama a una opción del menú se activa la función *`OnEjemplo`* que ejecuta el algoritmo *`Ejemplo`* contenido en la clase `Inspection`.

```
// InspDoc.h : interface of the CInspectionDoc class

#ifndef _INSPECTIONDOC_H
#define _INSPECTIONDOC_H

#include "ImgDoc.h"
#include "..\inspect\inspect.h"

class CInspectionDoc : public CImgBasicDoc, virtual public Inspection
{

// Constructor and destructor
protected:
  CInspectionDoc() {};
  virtual ~CInspectionDoc() {};

protected: // create from serialization only
       DECLARE_DYNCREATE(CInspectionDoc)

//Dialog user interface overridabled functions
public:

// Generated message map functions
protected:
  //{{AFX_MSG(CInspectionDoc)
  afx_msg void OnEjemplo() {Ejemplo();}
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

#endif //#define _INSPECTIONDOC_H
```

- Fichero "*InspDoc.cpp*", que contiene el mapa de mensajes de Windows. Es necesario repetir el mapa de mensajes contenido en la clase CImgBasicDoc.cpp para que la clase funciones correctamente. También he de notar que el identificador de recursos `IDR_INSPECTIONTYPE` se asocia al nuevo tipo de documento.

```
// CImgBasicDoc.cpp : implementation of the CInspectionDoc class

#include "stdafx.h"
#include "..\res\resource.h"
#include "InspDoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CInspectionDoc

IMPLEMENT_DYNCREATE(CInspectionDoc, CDocument)

BEGIN_MESSAGE_MAP(CInspectionDoc, CDocument)
	//{{AFX_MSG_MAP(CInspectionDoc)
	ON_COMMAND(ID_ALG_PROYA, OnEjemplo)
	ON_COMMAND(ID_ALG_PROYB, OnAlgProyB)
	ON_COMMAND(ID_ALG_PROYC, OnAlgProyC)
	ON_COMMAND(ID_ALG_PROYD, OnAlgProyD)
	//}}AFX_MSG_MAP
	ON_COMMAND(ID_EDIT_COPY, CDibDoc::OnEditCopy)
	ON_COMMAND(ID_EDIT_CUT, CDibDoc::OnEditCut)
	ON_COMMAND(ID_EDIT_PASTE, CDibDoc::OnEditPaste)
	ON_COMMAND(ID_EDIT_UNDO, CDibDoc::OnEditUndo)
	ON_COMMAND(ID_SAVE_MEMORY, CDibDoc::OnSaveMemory)
	ON_COMMAND(ID_FORMAT_BWDITHER, CDibDoc::OnFormatBwdither)
	ON_COMMAND(ID_FORMAT_GREYSCALE, CDibDoc::OnFormatGreyscale)
	ON_COMMAND(ID_FORMAT_REDUCED, CDibDoc::OnFormatReduced)
	ON_COMMAND(ID_FORMAT_FULLCOLOR, CDibDoc::OnFormatFullcolor)
	ON_UPDATE_COMMAND_UI(ID_SAVE_MEMORY, CDibDoc::OnUpdateSaveMemory)
	ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, CDibDoc::OnUpdateEditUndo)
	ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, CDibDoc::OnUpdateEditPaste)
	ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, CDibDoc::OnUpdateEditCut)
	ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, CDibDoc::OnUpdateEditCopy)
	ON_COMMAND(ID_MIRROR, CImgBasicDoc::OnMirror)
	ON_COMMAND(ID_FLIP, CImgBasicDoc::OnFlipVer)
	ON_COMMAND(ID_ROTATE90, CImgBasicDoc::OnRotate90)
	ON_COMMAND(ID_ROTATE_90, CImgBasicDoc::OnRotate_90)
	ON_COMMAND(ID_INVERT, CImgBasicDoc::OnInvert)
	ON_COMMAND(ID_MEDIAN, CImgBasicDoc::OnAlgMedian)
	ON_COMMAND(ID_OIL, CImgBasicDoc::OnAlgOil)
	ON_COMMAND(ID_PIXEL, CImgBasicDoc::OnAlgPixel)
	ON_COMMAND(ID_ROTATE, CImgBasicDoc::OnAlgRotate)
	ON_COMMAND(ID_ROTATECLR, CImgBasicDoc::OnAlgRotateclr)
	ON_COMMAND(ID_SPREAD, CImgBasicDoc::OnAlgSpread)
	ON_COMMAND(ID_TINF, CImgBasicDoc::OnAlgTinf)
	ON_COMMAND(ID_SHARPEN, CImgBasicDoc::OnAlgSharpen)
	ON_COMMAND(ID_BLEND, CImgBasicDoc::OnAlgBlend)
	ON_COMMAND(ID_EDGE, CImgBasicDoc::OnAlgEdge)
	ON_COMMAND(ID_BLUR, CImgBasicDoc::OnAlgBlur)
	ON_COMMAND(ID_SETTINGS, CImgBasicDoc::OnSettings)
	ON_COMMAND(ID_NOISE_GAUSSIAN, CImgBasicDoc::OnNoiseGaussian)
	ON_COMMAND(ID_BLUR_GAUSSIAN, CImgBasicDoc::OnBlurGaussian)
	ON_COMMAND(ID_CROP, CImgBasicDoc::OnCrop)
	ON_COMMAND(ID_SMOOTH, CImgBasicDoc::OnSmooth)
	ON_COMMAND(ID_MULTIRESGAUSS, CImgBasicDoc::OnMultiresgauss)
	ON_COMMAND(ID_GRADIENTE_ROBERTS, CImgBasicDoc::OnGradienteRoberts)
	ON_COMMAND(ID_ALG_CANNY, CImgBasicDoc::OnAlgCanny)
	ON_COMMAND(ID_FAST_MULTIRES, CImgBasicDoc::OnFastMultires)
	ON_COMMAND(ID_ALG_FMCANNY, CImgBasicDoc::OnAlgFmcanny)
END_MESSAGE_MAP()
```

- Fichero "*InspFrm.h*", que define el marco de una ventana que contiene asociada a la clase `Inspection`. Esta clase se encarga de que se cargen los recursos (menú, iconos, etc.) de la clase `Inspection` cada vez que una ventana de este tipo se haga activa.

```
/////////////////////////////////////////////////////////////////////////////

class CInspectionFrame : public CChildFrame
{
  friend CMainFrame;

       DECLARE_DYNCREATE(CInspectionFrame)

// Menu Data
protected: // menu to give to the the MDI Frame
       static CMenu m_menu;     // menu for all Inspection windows

// Load Icon and Menu on create
public:
  /* ClassWizard don't understand this  */
       virtual BOOL Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName,
                      DWORD    dwStyle    =    WS_CHILD    |    WS_VISIBLE    |
WS_OVERLAPPEDWINDOW,
                      const RECT& rect = rectDefault, CMDIFrameWnd* pParentWnd
= NULL,
                      CCreateContext* pContext = NULL);
  /* Neither this ( shit the bullocks)
  virtual BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle , const RECT& rect
= rectDefault, CMDIFrameWnd* pParentWnd = NULL, CCreateContext* pContext =
NULL);
  */
};
```

- Fichero "*InspFrm.cpp*", que contiene la implementación de la clase de tipo `CMdiChildWnd`, `CInspectionFrame`.

```
/////////////////////////////////////////////////////////////////////////////
// CInspectionFrame

IMPLEMENT_DYNCREATE(CInspectionFrame, CMDIChildWnd)

CMenu CInspectionFrame::m_menu;

BOOL CInspectionFrame::Create(  LPCTSTR lpszCDibViewClass,
                                LPCTSTR lpszWindowName,
                                DWORD dwStyle/* = 0 */,
                                const RECT& rect/* = rectDefault */,
                                CMDIFrameWnd* pParentWnd /* = NULL */,
                                CCreateContext* pContext /* = NULL */)
{
  // TODO: Add your specialized code here and/or call the base class

  // Setup the shared menu
  if (m_menu.m_hMenu == NULL)
  if (!m_menu.LoadMenu(IDR_INSPECTIONTYPE))
      return FALSE;

  // when the list output frame window is active, use this menu
  m_hMenuShared = m_menu.m_hMenu;

  // Register a custom WndClass and create a window.
  // This must be done because CInspectionDoc has a custom icon.
```

```
  if (lpszCDibViewClass == NULL)
      lpszCDibViewClass =
        AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW,
                    LoadCursor(NULL, IDC_ARROW),
                    (HBRUSH) (COLOR_WINDOW+1),
                    LoadIcon(AfxGetInstanceHandle(),
                   MAKEINTRESOURCE(IDR_INSPECTIONTYPE)));

  // create the CMDIChildWnd
  return CChildFrame::Create(lpszCDibViewClass, lpszWindowName,
                             dwStyle, rect, pParentWnd, pContext);
}
```

Instalación del entorno de visión

REQUERIMIENTOS DE HARDWARE

Para ejecutar el entorno de visión es necesario un ordenador que cumpla los siguientes requisitos:

— Ordenador compatible PC con procesador 386 DX o superior.

— El tamaño de la memoria RAM disponible debe ser al menos 4 Mb., aunque si se han de utilizar imágenes grandes es recomendable tener 8 Mb. o más.

— Disponer de al menos 2 Mb libres para la instalación en el disco duro, aunque si se utilizan imágenes grandes esta cifra deberá ser mayor.

— Tener instalado Windows 3.1 con la extensión Win32, Windows 95 o Windows NT. El ejecutable que acompaña por defecto al proyecto está compilado con Visual C++ 4.2 y sólo funciona bajo Windows 95 o NT. Si se quiere utilizar Windows 3.1, el proyecto debe estar compilado con Visual C++ 4.0 (en este caso, Ud. puede ponerse en contacto con D. Joaquín Mur para conseguir la versión adecuada).

Si además quiere programar el entorno de visión, deberá disponer del compilador Visual C++, un ordenador bastante más potente, 16 Mb de RAM, al menos 50 Mb de espacio libre en el disco duro y tener instalados Windows 95 o NT. También es recomendable tener una pantalla grande y una tableta de aspirinas a mano.

PROCESO DE INSTRALACIÓN

En los anexos del proyecto fin de carrera se adjuntan los discos de instalación del entorno. Introduzca el primer disco en la disquetera y ejecute el programa llamado "`setup.exe`". A partir de este momento, un programa asistente para la instalación le guiará a través de las opciones posibles.

La primera pregunta que tendrá que responder es qué tipo de instalación desea. Puede optar entre la instalación mínima, que sólo instalará los programas necesarios

para la ejecución del entorno, la instalación típica, que además instalará las ayudas del programa y algunas imágenes de ejemplo (es la versión recomendada) y la instalación a medida, en la cual podrá además instalar los ficheros fuentes del proyecto y la documentación del programa.

Los ficheros fuentes del proyecto se instalarán en el subdirectorio "`.\source`" Por conveniencia, he dispuesto todo los ficheros de código dentro de un archivo comprimido en el formato "`PKZIP`", que puede ser expandido por la mayoría de los descompresores actuales (Pkunzip, WinZip, etc.). Para poder compilar sin problemas la aplicación, debe expandirse con la opción de restituir la estructura de los directorios.

La documentación se compone de la memoria y de los anexos del proyecto fin de carrera. El formato de los documentos es Word 97 y se encuentran en un archivo comprimido en el formato "PKZIP" que se sitúa en el subdirectorio "`.\docs`".

Una vez escogido el tipo de instalación deberá responder el directorio en dónde quiere instalar el entorno, y del que colgarán los subdirectorios "`.\source`" y "`.\docs`". Por último, el programa de instalación permite ejecutar el entorno nada más concluir la instalación.

La desinstalación del programa es automática. Sólo es necesario ir al panel del control y ejecutar "Agregar o quitar programas". Si selecciona la aplicación "`Visio`", el entorno se eliminará completamente.

```
/* Copyright Notice
 * ================
 * Copyright 1989, 1994 by John Bradley
 *
 * Permission to copy and distribute XV in its entirety, for non-commercial
 * purposes, is hereby granted without fee, provided that this license
 * information and copyright notice appear unmodified in all copies.
 *
 * Note that distributing XV 'bundled' in with any product is considered
 * to be a 'commercial purpose'.
 *
 * Also note that any copies of XV that are distributed must be built
 * and/or configured to be in their 'unregistered copy' mode, so that it
 * is made obvious to the user that XV is shareware, and that they should
 * consider donating, or at least reading this License Info.
 *
 * The software may be modified for your own purposes, but modified
 * versions may not be distributed without prior consent of the author.
 *
 * This software is provided 'as-is', without any express or implied
 * warranty.  In no event will the author be held liable for any damages
 * arising from the use of this software.
 *
 * If you would like to do something with XV that this copyright
 * prohibits (such as distributing it with a commercial product,
 * using portions of the source in some other program, etc.), please
 * contact the author (preferably via email).  Arrangements can
 * probably be worked out.
 *
 * XV is shareware for PERSONAL USE only.  You may use XV for your own
 * amusement, and if you find it nifty, useful, generally cool, or of
 * some value to you, your non-deductable donation would be greatly
 * appreciated.  $25 is the suggested donation, though, of course,
 * larger donations are quite welcome.  Folks who donate $25 or more
 * can receive a Real Nice bound copy of the XV manual for no extra
 * charge.
 *
 * Commercial, government, and institutional users must register their
 * copies of XV, for the price of $25 per workstation/X terminal or per
 * XV user, whichever is less.  Note that it does NOT say 'simultaneous user',
 * but rather, the total number of people who use XV on any sort of
 * recurring basis. Site licenses are available (and recommended) for those
 * who wish to run XV on a large (>10) number of machines.
 * Contact the author for more details.
 *
 * The author may be contacted via:
 *
 *    Email:    bradley@cis.upenn.edu       (preferred!)
 *
 *    FAX:      (610) 520-2042
 *
 *    US Mail:  John Bradley
 *              1053 Floyd Terrace
 *              Bryn Mawr, PA  19010
 *
 * The author may not be contacted by (voice) phone.  Please don't try.
 *
 */
```