DEPARTAMENT D'INFORMÀTICA
UNIVERSITAT JAUME I

E80
PROYECTOS INFORMÁTICOS

INGENIERÍA INFORMÁTICA
Curso 2003--2004

Memoria Técnica del Proyecto

Representación de Flujo óptico mediante ficheros SWF

Proyecto presentado por el Alumno

Maikel Alonso Baña

Dirigido por **Raúl Montoliu Colás**
Tutorizado por **Javier Traver Roig**

Castellón, a 26 de Abril de 2004

INDICE

1. RESUMEN

En este documento se presenta la memoria del proyecto fin de carrera titulado "Representación de Flujo Óptico mediante ficheros SWF" realizado durante los meses de Mayo de 2003 a Abril de 2004. Este proyecto ha sido propuesto por el grupo de visión por computador y consiste en la creación de ficheros SWF que representen, de una forma interactiva, los resultados obtenidos en algoritmos destinados a calcular el flujo óptico. El proyecto ha sido desarrollado mediante la utilización de la librería MING, el lenguaje de programación C++ y librerías estándar, como la librería STL. Aunque el objetivo principal del proyecto es la creación de animaciones flash que representen de una forma interactiva el flujo óptico para que puedan ser utilizadas en tareas de investigación, podemos utilizar las técnicas desarrolladas en el presente proyecto en otro tipo de contextos como, por ejemplo, la docencia. Los resultados obtenidos a la finalización de este proyecto muestran la gran utilidad que aporta el uso de la librería Ming, frente al uso de una aplicación comercial, en la creación de animaciones flash así como las ventajas que se obtienen representando el flujo óptico mediante ficheros SWF en lugar de la utilización de ficheros postscript.

PALABRAS CLAVE

Visión por computador, análisis del movimiento, representación flujo óptico, Ming, SWF

2. INTRODUCCIÓN

La visión por computador es un área cuyo objetivo es crear un modelo del mundo real a partir de imágenes. Para ello, se pretende que un ordenador sea capaz de "comprender" las características de una o varias imágenes pertenecientes a una escena. Aunque, hasta el momento, un ordenador no realiza las mismas funciones que un ser humano en cuanto al "trabajo" con imágenes, sí que es capaz de llevar a cabo ciertas tareas que resultan muy interesantes para la visión por computador. Algunos de los objetivos que persigue la visión por computador son la detección, localización y reconocimiento de objetos en imágenes o seguimiento de un objeto en secuencias de imágenes.

Una parte importante de la visión por computador comprende el procesamiento de imágenes, campo en el cual las técnicas de visión por computador transforman imágenes para obtener otras con información útil.

La inteligencia artificial tiene como objetivo principal hacer a los computadores "inteligentes", tanto para que sean más útiles como para comprender los principios que hacen posible la inteligencia. Una de las áreas que podemos encontrar dentro de este campo es la visión artificial, cuyo objetivo es hacer que una máquina pueda "ver", o recibir mediante señales visuales la información necesaria del entorno que le rodea para que pueda tomar decisiones oportunas o realizar las acciones o cálculos adecuados en cada momento como, por ejemplo, reconocer ciertos objetos o desplazarse por un entorno sin dificultades.

Para conseguir este objetivo la máquina deberá ser capaz de realizar una estimación de movimiento mediante el procesamiento de una secuencia de imágenes. El movimiento percibido en una secuencia de imágenes no está determinado únicamente por la proyección del campo de velocidades tridimensionales asociado a la escena real, también conocido como movimiento real, sino que además hay que tener en cuenta una serie de características que pertenecen a lo que se conoce como movimiento aparente. Estas características son, por ejemplo, la iluminación de la escena, las propiedades reflectantes de los objetos, la forma y estructura de las superficies, etc... Podemos hablar pues de movimiento real y de movimiento aparente, también conocido como flujo óptico.

El hecho de llevar a cabo el cálculo del flujo óptico nos permite obtener resultados que posteriormente pueden aplicarse en tareas como:

- Segmentación de movimiento.
- Cálculo del foco de expansión (punto de la imagen hacia o desde el cual los rasgos de la imagen parecen moverse).
- Cálculo de tiempos de colisión.

Pero antes de todo esto debemos comprender en qué consiste el flujo óptico. A continuación se va a llevar a cabo una pequeña introducción a este concepto.

2.1 Descripción del concepto de flujo óptico

El flujo óptico se puede entender, desde una perspectiva coloquial, como el movimiento visual que un observador experimenta mientras se mueve. Cuando el observador se mueve mirando hacia el frente, cree percibir por parte de los objetos a los que se va acercando, un movimiento contrario al suyo propio. Este movimiento de los objetos es lo que, vulgarmente, se puede denominar flujo óptico o movimiento aparente.

Cuando se producen movimientos de objetos en el mundo real se pueden capturar en una secuencia de imágenes cada una de las cuáles es el resultado de la proyección de las intensidades de los objetos del mundo real en un plano imagen. Desde un punto de vista más técnico, el flujo óptico, también denominado vector desplazamiento o vector velocidad, es un campo vectorial que asigna a cada píxel de una imagen las dos componentes de desplazamiento, utilizando la variación de intensidad de una imagen a otra de la secuencia de imágenes. Este campo vectorial no corresponde al desplazamiento real tridimensional sino al movimiento aparente en el plano de la imagen.

En definitiva, el término flujo óptico se puede entender como el movimiento aparente que se puede observar en el plano imagen debido a cambios de intensidades producidos en una secuencia de imágenes.

En este proyecto se ha optado por representar el flujo óptico mediante ficheros SWF (animaciones flash), frente a los ficheros postscript que han sido el método usado hasta el momento. La razón por la que se ha decidido utilizar animaciones flash es la gran interactividad que estos aportan a la representación del flujo óptico, además de la compatibilidad que este tipo de ficheros tiene con casi cualquier sistema, algo de lo que los ficheros postscript carecen.

Para la creación de los ficheros SWF se ha utilizado la librería Ming, en lugar de una aplicación comercial, ya que esta librería da la posibilidad de desarrollar código en lenguaje C++ mediante el cual generar la animación flash, característica que hace muy interesante su uso. Esto se debe a que la creación de ficheros SWF mediante código de un determinado lenguaje de programación puede resultar sencilla y muy útil frente al uso de una aplicación comercial en la que, generalmente, el método de trabajo es muy “manual” y, por tanto, puede llegar a resultar más costosa.

Los resultados obtenidos a la finalización del proyecto muestran de una manera muy clara las ventajas de utilizar ficheros SWF para representar flujo óptico. Además, demuestran que la elección de la librería Ming como herramienta para generar ficheros SWF ha sido correcta, ya que aporta muchas ventajas incluso no sólo en cuanto a la representación de flujo óptico.

2.2 Organización del resto del documento

A continuación, se comentan brevemente las distintas partes en las que se divide este documento, así como los puntos más importantes que se tratan en cada uno de ellos.

En la sección "Planificación del proyecto" se comenta cuál ha sido la duración total del proyecto, las tareas que se han llevado a cabo para la realización del mismo, así como en qué consiste cada una de ellas y el tiempo que ha requerido llevarlas a cabo.

El apartado "Trabajo realizado" comenta el hecho de que en el presente proyecto se ha optado por representar el flujo óptico mediante animaciones flash, con las ventajas que esto implica. Hasta el momento, tras calcular el flujo óptico se recurría a una representación mediante ficheros postscript. Este tipo de representación cuenta con varios inconvenientes. Uno de ellos es la poca interactividad que ofrecen, ya que la representación es muy estática en ficheros postscript, pues no hay forma de modificar la manera en la que el flujo óptico es mostrado. Otro problema es el hecho de que no todos los sistemas cuentan con aplicaciones que permitan la apertura de ficheros de tipo postscript.

Por todo esto, se puede afirmar que la representación de flujo mediante animaciones flash es mucho más conveniente. Hay que tener en cuenta que las animaciones flash tienen la ventaja, frente a los ficheros postscript, de tener una mayor compatibilidad con cualquier sistema, ya que casi todos tienen alguna aplicación que permite abrir ficheros SWF. Otra ventaja que aporta la utilización de ficheros SWF es la posibilidad de crear animaciones formadas por secuencias de imágenes, algo que en el caso del flujo óptico puede llegar a ser muy útil. Las animaciones flash cuentan también con una interactividad tal que aporta multitud de opciones que permiten representar el flujo óptico de diferentes formas, según las necesidades de quién está haciendo uso de la animación. Todo esto se comentará con más detalle en la presente memoria.

En este apartado de la memoria también se detallan las acciones necesarias para insertar un fichero SWF en un documento HTML, para así poder visualizar la animación flash. Esto se debe hacer una vez leída la información de un fichero de flujo óptico y generado el fichero SWF. Además, se pone un ejemplo para un mejor entendimiento del método y se explica detalladamente cada uno de los parámetros necesarios para una correcta visualización de la animación flash en un navegador web.

En el apartado "Librería Ming", se comenta qué herramienta se ha usado para la creación de ficheros SWF que representen el flujo óptico. En este caso, se ha optado por el uso de la librería Ming, capaz de generar ficheros SWF mediante el desarrollo de código C++, PHP, u otros lenguajes. El hecho de usar esta herramienta se debe a la gran facilidad con la que se pueden tratar, mediante programación, datos que sean resultados de experimentos de investigaciones, frente al uso de los mismos mediante una aplicación comercial como, por ejemplo, Macromedia Flash. El hecho de utilizar la programación para tratar dichos datos, facilita mucho las tareas de creación de animaciones flash mientras que con una aplicación comercial el trabajo a llevar a cabo sería excesivamente

elevado y complejo. Además, en este proyecto se ha utilizado el lenguaje de programación C++[1], el cual está muy extendido entre los programadores. Por todo ello, se ha optado por el uso de la librería Ming y el lenguaje C++ como herramientas básicas en el desarrollo de este proyecto.

Por otra parte, en el apartado "Insertar un fichero SWF en Powerpoint" se explica, paso a paso, las acciones necesarias para insertar una animación flash en un documento de Powerpoint, ya que, en caso de requerir el uso de animaciones flash para hacer exposiciones sobre investigaciones o para la docencia, el Powerpoint puede ser una aplicación muy cómoda para su visualización.

En las secciones "Resultados" y "Conclusiones" se explican cuáles han sido los resultados obtenidos a la finalización de este proyecto. En este caso, se han obtenido dos aplicaciones desarrolladas en lenguaje C++, junto con la librería STL y la librería Ming. Ambas aplicaciones permiten generar ficheros SWF que podemos utilizar para distintos fines. En nuestro caso, hemos utilizado una de las aplicaciones para generar animaciones flash mediante las cuáles representar el flujo óptico calculado anteriormente. La representación de flujo óptico mediante animaciones flash en lugar de usar ficheros postscript es un gran avance, ya que permite una mayor interactividad en la representación. Además, mediante ficheros SWF se puede representar el flujo óptico de una secuencia de imágenes de forma animada, algo que con los ficheros postscript no es posible.

En cuanto a la otra aplicación desarrollada, su función es generar animaciones flash compuestas de secuencias de imágenes de las que se dispone en formato JPEG. El objetivo de la creación de estas animaciones es utilizarlas en áreas de investigación o de docencia, ya que mediante animaciones es mucho más fácil representar, por ejemplo, procesos iterativos o que varían en el tiempo.

Una de las mayores ventajas que ofrece la librería Ming es la posibilidad de generar la animación flash a partir de código en un determinado lenguaje de programación, en lugar de tener que usar una aplicación comercial como, por ejemplo, Macromedia Flash. Esta característica es muy interesante ya que, si se quiere generar una animación flash de, por ejemplo, los resultados de ciertos experimentos, si el número de elementos a representar en la animación es muy elevado, es un trabajo muy costoso de realizar mediante una aplicaciones comerciales, ya que la mayoría de estas suelen tener un funcionamiento más bien "manual" (hay que trabajar elemento a elemento). Sin embargo, si se utiliza un lenguaje de programación (con sus estructuras de control) y una serie de métodos, como los que ofrece la librería Ming, esta tarea puede convertirse en algo muy sencillo. Por todo ello, veremos que el uso de la librería Ming puede ser muy recomendable, no solo en las tareas en las que se ha utilizado en el presente proyecto sino en otras tareas futuras.

Por último, se adjunta con la memoria un apéndice en el cual se puede observar el código de los principales módulos desarrollados en el presente proyecto para generar las animaciones flash que representen el flujo óptico. Estos ficheros son adjuntados a la

[1] *Documentación de apoyo utilizada: ver Bibliografía [5]*

memoria ya que en varias secciones de la misma se hace referencia a ellos y, en algún caso, pueden clarificar ciertas explicaciones descritas en esta memoria mediante pseudocódigo. También se muestra, casi al final del documento la bibliografía utilizada para el desarrollo de este proyecto.

3. PLANIFICACIÓN DEL PROYECTO

En este apartado de la memoria se comentan ciertos detalles referentes a la gestión del presente proyecto. De esta manera, se explican ciertas características del proyecto como son la duración que ha tenido este, las tareas llevadas a cabo, con su coste temporal y una explicación detallada sobre qué se ha hecho en cada una de estas tareas.

3.1 Duración del proyecto

La elección del proyecto se llevó a cabo en Abril de 2003. Tras varias reuniones con el director del proyecto, se comenzó a realizar el mismo en Mayo de 2003 y se finalizó en Abril de 2004.

3.2 Tareas realizadas

Las tareas llevadas a cabo para la realización del presente proyecto han sido las siguientes:

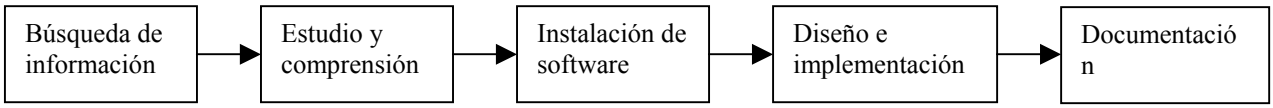


Fig.1 Tareas realizadas en el presente proyecto

3.3 Coste temporal

En la siguiente tabla se muestra una comparativa entre las horas planificadas para la realización de las tareas del proyecto y las horas realmente utilizadas para la finalización de las mismas. Además, se muestra también el computo total de horas necesarias invertidas para llevar a cabo dicho proyecto.

TAREA	HORAS PLANIFICADAS	HORAS REALES
Búsqueda de información	20	30
Estudio y comprensión	35	70
Instalación de software	2	5
Diseño e implementación	200	250
Documentación	25	55
TOTAL	**282**	**410**

Fig.2 Coste temporal en horas para cada una de las tareas

La búsqueda de información hace referencia al hecho de tener que realizar entrevistas para determinar qué se pretende realizar en el proyecto y que documentación es necesaria. Así, en esta tarea hay que tener en cuenta el tiempo invertido en las entrevistas con el director, junto con el tiempo necesario para buscar documentación (bibliográfica o

mediante buscadores de páginas web) de apoyo para llevar a cabo el proyecto. En el presente proyecto se ha partido de la utilización de información aportada por el director del proyecto, aunque posteriormente ha sido necesaria información referente a la librería Ming, la librería STL, incluso teoría matemática (ecuación de la recta que pasa por un punto) necesaria para representar las "flechas" de flujo óptico. Algunas de estas informaciones han sido buscadas en Internet y se especifican en la bibliografía.

La tarea "estudio y comprensión" se refiere al hecho de que, una vez que se dispone de la información suficiente para llevar a cabo el proyecto, es necesario llevar a cabo un estudio exhaustivo de dicha información. Hay que determinar aquello que es determinante para el presente proyecto y aquello que puede ser irrelevante. Además, se deben comprender perfectamente los conceptos a desarrollar en el proyecto, así como reflexionar sobre qué herramientas van a ser necesarias para la realización de este.

La instalación de software es una tarea que no es muy costosa, en cuanto a tiempo se refiere, aunque si que es importante. Esto se debe a que una buena elección en cuanto al software a utilizar puede ser muy determinante en la consecución de los objetivos fijados al comienzo de la realización del proyecto.

La tarea "diseño e implementación" hace referencia a todas las acciones necesarias para definir las distintas partes de las cuales se compondrá el proyecto. Esto se refiere a determinar, cómo se lleva a cabo la lectura de información de flujo óptico calculado a partir de experimentos anteriores y de la que se dispone en ficheros .uji, almacenamiento de dicha información en algún tipo de estructura de datos, uso de métodos de la librería Ming para representar el flujo óptico en un fichero SWF y la generación de un documento HTML para mostrar dicha representación de flujo óptico. En esta tarea, además de determinar cuáles son todas estas partes, se lleva a cabo la implementación necesaria para llevar a cabo cada una de las fases del proyecto descritas anteriormente.

En cuanto a la documentación, esta tarea simplemente hace referencia al hecho de redactar la presente memoria, para así detallar qué se ha hecho en el presente proyecto y facilitar la comprensión del mismo.

El coste temporal de realización del proyecto ha sido de 410 horas sobre una estimación inicial de casi 300 horas. Esto se debe a que, en la mayoría de partes de las que se compone el proyecto, el tiempo utilizado realmente ha sido mayor que el estimado inicialmente. Esto es algo normal, ya que durante el desarrollo del proyecto aparecen nuevos problemas que dificultan la realización de ciertas tareas.

3.4 Recursos utilizados

El hardware y software utilizado en la realización de este proyecto ha sido:

- Desknote ECS i-Buddie-XP Athlon XP 1800+
- Sistema Operativo Linux
- Librería Ming
- Librería STL
- Lenguaje de programación C++
- Editor KWrite de Linux
- Navegador web Mozilla

4. LIBRERÍA MING

Introducción

En este apartado, se comenta qué herramienta se ha usado para la creación de ficheros SWF que representen el flujo óptico. En este caso, se ha optado por el uso de la librería Ming, capaz de generar ficheros SWF mediante el desarrollo de código C++, PHP, u otros lenguajes. También se muestran las ventajas que ofrece el uso de esta herramienta para representar flujo óptico, frente al uso de ficheros postscript, utilizados hasta este momento. Además, se añade un breve tutorial con los comandos básicos de la librería Ming necesarios para generar ficheros SWF.

Ming es una librería C de código abierto (LGPL) que permite escribir películas SWF ("Flash"). También es un juego de envoltorios para usar la librería de C++ y lenguajes de script como PHP, Python, y Ruby.

La librería Ming soporta casi todas las características de Flash 4, incluyendo: formas, bitmaps (pngs y jpegs), gradientes, transformaciones ("cambio de forma"), texto, botones, acciones, símbolos gráficos ("clips de película"), soporte audio mp3, y las transformaciones de color. La única característica que todavía no soporta son los eventos de sonido. Una interesante página web sobre la librería Ming es la conocida como Mingdocs[2].

La librería Ming puede descargarse directamente desde su página oficial[3]. En está página, se encuentra un enlace "Installation Guide" mediante el cual se accede a una página donde se explican con detalle los pasos necesarios para instalarla y compilarla.

Ming ofrece una serie de ventajas que han hecho muy interesante su uso en las tareas realizadas en este proyecto. Permite usar el lenguaje C++ , lenguaje muy extendido entre los programadores, y que permite fácilmente generar las películas flash sin usar el Macromedia Flash. También da la posibilidad de trabajar con distintos tipos de objetos que permiten representar de manera muy eficiente los conceptos planteados en este proyecto.

Otra ventaja es la facilidad de uso de los distintos métodos que ofrece, tanto por su simplicidad como por el hecho de contar con tutoriales y manuales de referencia que resuelven cualquier duda que pueda surgir.

[2] Ver *Bibliografía [3]*
[3] *Para visitar la página principal de Ming, ver Bibliografía [1]*

Hasta el momento se ha utilizado el "código de Barron, Fleet y Beauchemin"[4] para generar ficheros postscript en los cuales se pintaba el flujo óptico. En la siguiente figura se puede observar un ejemplo de fichero de este tipo:

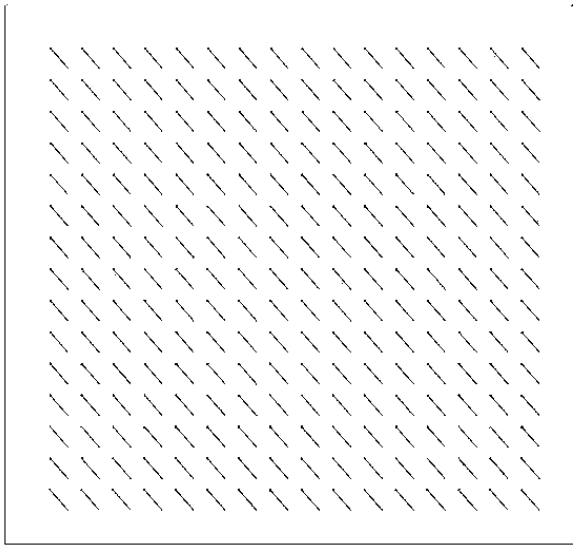


Fig.3 Ejemplo de flujo óptico pintado en fichero postscript

La representación de flujo óptico hasta este momento parecía ser suficiente. Sin embargo, mediante la utilización de ficheros flash que representen el flujo óptico se pueden conseguir ventajas que no se obtienen con los ficheros de tipo postscript. El problema está en que cuando se realizan multitud de experimentos, es inviable que para cada uno se tenga que crear manualmente, mediante una herramienta visual, la animación flash correspondiente a un flujo óptico. Por lo tanto, dichas animaciones se han de crear desde algún programa, mediante una librería capaz de escribir directamente ficheros SWF.

Esto se puede llevar a cabo mediante la librería Ming. En primer lugar, la utilización de ficheros flash permite mostrar los ficheros de este tipo en cualquier sistema, ya que normalmente todos cuentan con un navegador de Internet que permita la visualizar objetos de este tipo, mientras que no todos los sistemas cuentan con algún programa que permita abrir ficheros de tipo postscript.

Además, los ficheros de tipo flash permiten crear animaciones. Esto ofrece una gran ventaja frente a los ficheros postscript ya que es posible crear películas flash en las que se muestren secuencias de imágenes que tienen superpuestos los flujos ópticos de cada una de las imágenes de la secuencia. De esta forma, se puede representar "el movimiento producido en un cierto intervalo de tiempo". También se pueden crear ficheros flash que contengan sólo el flujo óptico o sólo las imágenes.

A parte de la utilización de la librería Ming en lo referente al flujo óptico, también se pueden encontrar otras aplicaciones interesantes debido a la posibilidad de generar películas flash con secuencias de imágenes. En este sentido, se pueden utilizar imágenes resultantes de experimentos iterativos, para incluirlas en películas y mostrar la evolución

[4] *Ver Bibliografía [4]*

de un cierto fenómeno como, por ejemplo, en el caso del algoritmo Sieve (ver fig. 29), se puede observa la "evolución de la segmentación".

En definitiva, la utilización de la librería Ming para generar ficheros de tipo flash que representen el flujo óptico ofrece más ventajas que la utilización de ficheros postscript.

A continuación se explica brevemente cómo construir animaciones flash simples con la librería Ming usando el envoltorio de C++.

4.1 Comandos básicos de dibujo.

El objeto fundamental en una película flash es una forma. En Ming, se refiere a un objeto *SWFShape*. En C++, se instancia un objeto *SWFShape* con la siguiente expresión:

```
SWFShape *shape = new SWFShape();
```

Aunque la librería Ming está escrita en código C plano, está diseñada para ser utilizada en un ambiente orientado a objetos; por tanto, en C++ se puede acceder a las funciones de dibujo de Ming mediante el objeto *SWFShape*.

Para dibujar un rectángulo amarillo de 100 unidades en cada lado con una anchura de línea de 10 unidades se debe hacer:

```
SWFShape *shape = new SWFShape();

shape->setLine(10, 200, 200, 0);
shape->movePenTo(180,180);
shape->drawLine(100,0);
shape->drawLine(0,100);
shape->drawLine(-100,0);
shape->drawLine(0,-100);
```

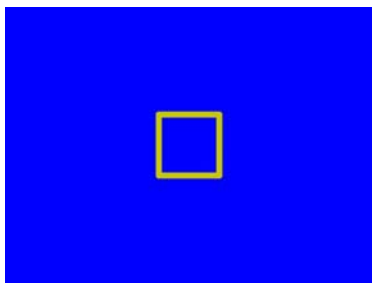


Fig.4 Rectángulo[5]

El dibujo en Ming está basado en pluma, es decir, el objeto forma guarda una referencia de la ubicación de la pluma imaginaria en la superficie del dibujo, y los comandos de dibujo referencian y afectan implícitamente la ubicación de la pluma. Se pueden utilizar dos funciones distintas para dibujar líneas con Ming. En el ejemplo anterior se ha utilizado la función *drawLine* que usa posicionamiento relativo pero también se podría haber utilizado la función *drawLineTo* que usa coordenadas absolutas (x,y) para dibujar. Si la hubiéramos utilizado, las funciones de dibujo anteriores podrían reemplazarse por

```
shape->drawLineTo(400, 0);
shape->drawLineTo(0, 400);
shape->drawLineTo(-400, 0);
shape->drawLineTo(0, -400);
```

[5] *Para ver este flash en funcionamiento visitar la página: ver Bibliografía [2]. Código en apéndice VII*

obteniendo el mismo rectángulo amarillo.

El método *setLine* recibe cuatro (o cinco) argumentos: el ancho de la línea, y el rojo, verde, y azul (y opcionalmente la transparencia), componentes del color de la línea. Cuando se fija el estilo de línea, este se aplica a todos los comandos de dibujo que se utilicen a continuación hasta que se cambie de nuevo el estilo de línea con *setLine*. Para no utilizar estilo de línea (que es el valor por defecto para las formas), simplemente hay que fijar el ancho en cero.

Dos de los comandos de dibujo más útiles son:

```
shape->movePenTo(x,y);
```

el cual mueve la pluma a las coordenadas (x,y) sin dibujar línea (*movePen* se mueve usando posicionamiento relativo), mientras que

```
shape->drawCurveTo(cx, cy, ax, ay);
```

dibuja una curva cuadrática de Bezier simple desde la ubicación actual de la pluma hasta el punto (ax,ay) usando el punto (cx,cy) como punto de control. Mediante *drawCurve* se hace lo mismo, excepto que usa coordenadas relativas a la posición actual de la pluma.

4.2 Estilos De Relleno

El formato SWF está diseñado para ser compacto y fácil de utilizar, esto significa que las optimizaciones en los atributos de formas se realizan, en este caso, mientras se está programando y no por el reproductor. Por tanto, cuando se está definiendo una forma se debe indicar al reproductor qué porción del espacio entre las líneas debe quedar rellena. Esto se debe a que el reproductor flash dibuja la forma mientras verifica las líneas, y como este recorre cada borde en su definición de forma, revisa los registros de relleno del lado izquierdo y del lado derecho para ver que estilo de relleno se está aplicando sobre la línea. Si el borde es más denso al norte que al sur, el explorador de línea se esta moviendo al lado derecho del borde mientras cruza sobre la línea desde la derecha hacia la izquierda, así empieza rellenando con el estilo de relleno del borde del lado derecho. Si el borde fuera más denso al sur que al norte, usaría el estilo relleno del lado izquierdo.

Para fijar el relleno del lado izquierdo y del lado derecho se utilizan los métodos *setRightFill* y *setLeftFill*. Éstos, simplemente trabajan como *setLine*, se aplican a todos los bordes que se utilicen (tras llamar a *setRightFill* y/o *setLeftFill*) hasta que cambie el estilo de relleno de nuevo. *setRightFill* y *setLeftFill* reciben un argumento, un objeto *SWFFill*.

SWFFill permite crear rellenos sólidos, gradientes, y mapas de bits. En lugar de instanciar un *SWFFill* directamente, se solicita un *SWFShape* para que se cree uno para el

programador utilizando el método *addFill* de *SWFShape*. Se debe realizar así ya que el relleno sólo tiene significado dentro del contexto de la forma. El método *addFill* tiene una variedad de formatos, uno para cada uno de los tipos de relleno que puede crear: colores sólidos, mapas de bits, y gradientes.

Para definir un relleno de color sólido con los componentes especificados rojo, verde, y azul (y opcionalmente transparencia) se debería escribir al menos el siguiente código:

```
SWFShape *shape = new SWFShape();
shape->setRightFill(shape->addFill(r, g, b [,a]));
```

En el siguiente ejemplo se crea un rectángulo con un relleno de color rojo sólido:

```
SWFShape *shape = new SWFShape();
shape->setRightFill(shape->addFill(255, 0, 0));

shape->movePenTo(-200, -200);
shape->drawLine(100, 0);
shape->drawLine(0, 100);
shape->drawLine(-100, 0);
shape->drawLine(0, -100);
```

En el ejemplo anterior se usa *setRightFill* porque se está dibujando el contorno del rectángulo en el sentido de las agujas del reloj, así, el interior del rectángulo se rellena desde el borde del lado derecho. Si se hubiera dibujado el rectángulo en sentido contrario a las agujas del reloj, se habría usado el método *setLeftFill* para rellenar por la izquierda de la línea pintada.

A continuación se muestra una tabla como resumen de los principales métodos para un objeto de tipo forma:

Método	Descripción
*SWFShape *shape = new SWFShape();*	Crea un objeto de tipo forma.
shape->setLine	Determina el estilo de la línea a pintar.
shape->addFill	Añade relleno sólido a la forma
shape->setLeftFill	Añade el relleno por la izquierda de la línea pintada.
shape->setRightFill	Añade el relleno por la izquierda de la línea pintada.
shape->movePenTo	Mueve la pluma de la forma (posición absoluta).
shape->movePen	Mueve la pluma de la forma (posición relativa).
shape->drawLineTo	Dibuja una línea mediante posiciones absolutas.
shape->drawLine	Dibuja una línea mediante posiciones relativas.
shape->drawCurveTo	Dibuja una curva mediante posiciones absolutas.
shape->drawCurve	Dibuja una curva mediante posiciones relativas.

Fig. 5 - Principales métodos de un objeto “shape” (forma)

Otra forma de crear rellenos para formas consiste en utilizar una imagen, en lugar de un color sólido. Para agregar un relleno de mapa de bits a una forma, usamos

```
shape->addBitmapFill(b));
```

donde el mapa de bits es un objeto SWFBitmap, creado con

```
SWFBitmap *b = new SWFBitmap(filename );
```

filename es el nombre de un jpeg o un archivo dbl. Un archivo "dbl" es un fichero png convertido mediante la utilidad "png2dbl" el cual Ming puede procesar más fácilmente.

SWFBitmap también tiene métodos para obtener las dimensiones de su imagen

```
b->getWidth ();
b->getHeight ();
```

que le permite dibujar un rectángulo ,del mismo tamaño de su mapa de bits, que rodee a la imagen.

En la figura que se muestra a continuación, se puede observar un código de ejemplo que genera una película en la que se muestra una imagen jpeg:

```
#include "../../mingpp.h"

int main()
{
  Ming_init();
  SWFDisplayItem  *jpg;
  SWFMovie *movie=new SWFMovie();

  movie->setDimension(500,400);
  movie->setBackground(255,255,255);

  SWFShape *shape = new SWFShape();
  SWFBitmap *b = new SWFBitmap("imagen1.jpg");
  shape->setRightFill(shape->addBitmapFill(b)); //asignar imagen
a "shape"

  shape->setLine(1,0,0,255);
  shape->drawLine(b->getWidth(),0);  //recuadro que rodeara a la
imagen
  shape->drawLine(0,b->getHeight());
  shape->drawLine(-b->getWidth(),0);
  shape->drawLine(0,-b->getHeight());

  jpg = movie->add(shape);
  movie->save("./imagen.swf");
}
```

Fig.6 Ejemplo de relleno de mapa de bits[6]

[6] *Para ver este flash en funcionamiento visitar: ver Bibliografía [2] Código en apéndice IX*

4.3 Transformación de un relleno

Se pueden realizar modificaciones en los rellenos usando los siguientes métodos: (suponiendo que tenemos un objeto f de tipo *SWFFill*)

```
f->moveTo(x,y);
```

mueve el relleno a las coordenadas (x,y),

```
f->rotateTo(deg);
```

gira el relleno deg grados, y

```
f->scaleTo(xscale [, yscale]);
```

fija la escala de relleno en xscale en dirección x y yscale en dirección y. Si sólo se proporciona un argumento, ambas dimensiones son escaladas por xscale. Por último,

```
f->skewXTo(s);

f->skewYTo(s);
```

sesga el relleno por s en el eje x o en el eje y. Una s de 0 no es sesgada, una s de 1.0 es sesgada a un ángulo de 45 grados.

4.4 Animación

Cuando se ha definido una forma, para poder verla necesitamos un lienzo en el cual dibujar. Ese lienzo se puede generar mediante

```
SWFMovie *movie = new SWFMovie ();
```

mediante el cual se crea un nuevo objeto *SWFMovie*, base para la animación. Posteriormente se puede usar

```
movie->setBackground(r, g, b);
```

para fijar el color del fondo, así como

```
movie->setRate(rate);   rate ∈ [1.0,20.0]
```

para fijar el “frame rate” (en frames por segundo), y

```
movie->setDimension(anchura, altura);
```

para fijar el tamaño de la película.

Una forma no se dibujará en una película mientras no se cree una instancia de ella llamando

```
SWFDisplayItem *i = new SWFDisplayItem();
```

y se añada a la película mediante

```
i = movie->add(shape);
```

Este método crea una instancia de la forma en la película y retorna una referencia con la que puede alterar la apariencia de la instancia. De hecho, se puede agregar la misma forma a una película tantas veces como se desee (siendo razonable, claro) y usar los apuntadores devueltos para referenciar cada uno individualmente.

También se puede quitar un objeto de la película llamando al método

```
movie->remove(i);
```

Usar Ming para hacer películas flash es análogo a usar una cámara de movimiento. Se deben poner las formas en su lienzo, colocarlos como se desee, pulsar entonces el obturador para grabar las modificaciones que se hayan realizado y avanzar al siguiente "frame". En Ming, para avanzar al siguiente "frame" se llama al método:

```
movie->nextFrame();
```

Las modificaciones llevadas a cabo se realizan llamando a métodos proporcionados por el apuntador a la instancia que se generó con el método *add* de *SWFMovie* (arriba lo hemos definido como apuntador i). De esta manera

```
i->moveTo(x,y);
```

mueve el objeto a las coordenadas (x,y), mientras

```
i->move(x,y);
```

desplaza el objeto mediante el vector (x,y). El resto de los métodos son:

i->rotateTo(deg);	i->skewXTo(s);
i->rotate(deg);	i->skewX(s);
i->scaleTo(xscale [, yscale]);	i->skewYTo(s);
i->scale(xscale [, yscale]);	i->skewY(s);

Estos métodos son similares a los comentados en la sección de transformaciones de rellenos.

Cuando se tiene todo listo, se puede guardar la película en un fichero SWF llamando al método

```
movie->save("archivo.swf");
```

A continuación se muestra un ejemplo de animación en el que un rectángulo se desplaza horizontalmente sobre un fondo rojo:

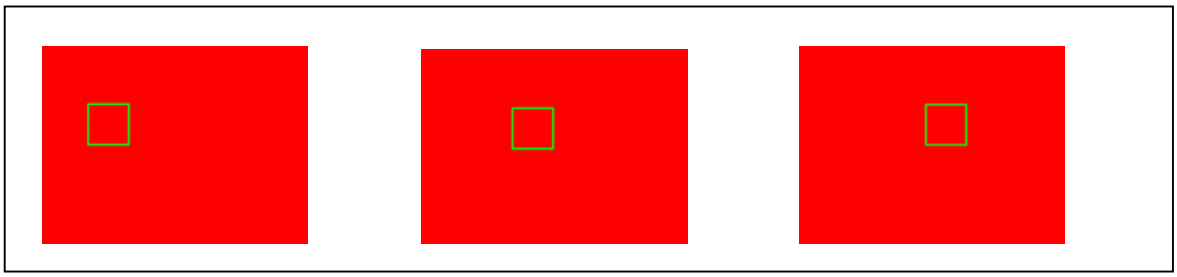


Fig.7 Rectángulo animado[7]

El código necesario para generar esta película podría ser, por ejemplo, el siguiente:

```
int main()
{
  Ming_init();

  // Define la animacion

  SWFMovie *Movie=new SWFMovie();
  Movie->setDimension(500,500);
  Movie->setBackground(255,0,0);
  Movie->setRate(20.0);

  // Elemento animado
  SWFShape *Shape1=new SWFShape();

  Shape1->setLine(5,0,255,0);

  Shape1->movePen(-50,-50);
  Shape1->drawLine(100,0);
  Shape1->drawLine(0,100);
  Shape1->drawLine(-100,0);
  Shape1->drawLine(0,-100);

  // Objeto para mover el cuadrado
  SWFDisplayItem *MovimientoCuadrado;

  MovimientoCuadrado=Movie->add(Shape1);
  MovimientoCuadrado->moveTo(50,200);

  for (int i=0; i<50; i++) {

      Movie->nextFrame();
      MovimientoCuadrado->move(5,0);
  }

  Movie->save("./rectangulo_animado.swf");
```

[7] Para ver este flash en funcionamiento visitar: *ver Bibliografía [2]*
Código en apéndice VIII

```
}
```

Un ejemplo más complejo podría consistir en hacer una película en la que aparezca una imagen y sea sustituida por otra. Por ejemplo:

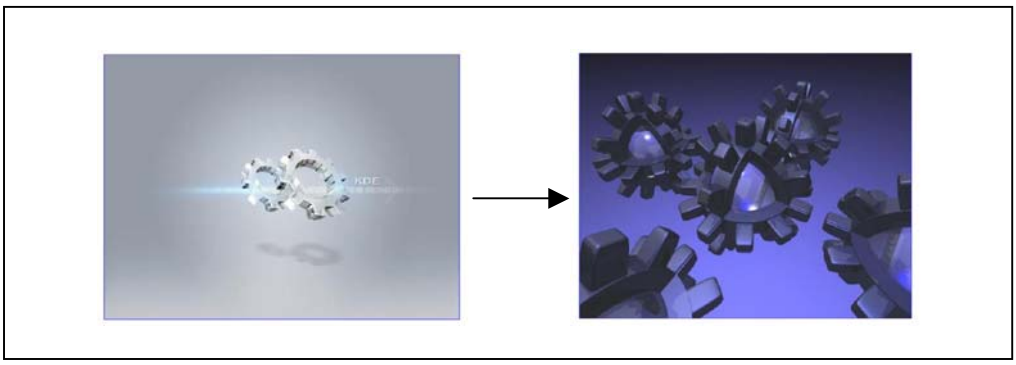


Fig. 8 - Película que cambia de imagen[8]

Para generar una película como la del ejemplo anterior se podría generar el código siguiente:

```
int main()
{
  Ming_init();

  SWFMovie *movie=new SWFMovie();

  movie->setDimension(500,400);
  movie->setBackground(255,255,255);

  movie->setRate(2.0);


      SWFShape *shape, *shape_ant,*shapejpg,*shapejpgant;
      SWFDisplayItem *anterior,*actual,*jpg,*jpgant;

      SWFBitmap *b;

      shape_ant = new SWFShape();
      shapejpgant = new SWFShape();

      anterior = movie->add(shape_ant);  // inicializacion
      jpgant = movie->add(shapejpgant);

      for (int i=0; i<2; i++)
      {
      SWFShape *shapeaux = new SWFShape();
      shape_ant = new SWFShape();
      shapejpg = new SWFShape();
      shapejpgant = new SWFShape();

            movie->remove(anterior);     // borra el shape anterior
            // prepara el nuevo jpg para mostrarlo en shapejpg
            movie->remove(jpgant);

            if (i == 0 )
```

[8] Para ver este flash en funcionamiento visita: *ver Bibliografía [2] Código en apéndice X*

```
               b = new SWFBitmap("imagen1.jpg");
            else b = new SWFBitmap("imagen2.jpg");

            shapeaux->setRightFill(shapeaux->addBitmapFill(b)); //asignar
imagen a "shape"
            shapeaux->setLine(1,0,0,255);

//recuadro que rodeara a la imagen

            shapeaux->drawLine(b->getWidth(),0);
            shapeaux->drawLine(0,b->getHeight());
            shapeaux->drawLine(-b->getWidth(),0);
            shapeaux->drawLine(0,-b->getHeight());

            shapejpg = shapeaux;
            shape = shapejpg;
            jpg = movie->add(shapejpg);


  actual = movie->add(shape);  //muestra el nuevo shape

            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();

            shape_ant = shape;
            shapejpgant = shapejpg
            anterior = actual;
            jpgant = jpg;

      }

  movie->save("./imagen_animada.swf");
}
```

A continuación se muestra una tabla como resumen de los principales métodos para un objeto de tipo película:

Método	Descripción
*SWFMovie *movie = new SWFMovie();*	Crea un objeto de tipo película.
movie->save	Graba la película en un fichero.
movie->add	Añade cualquier tipo de dato a la película.
movie->remove	Quita un objeto de la película .
movie->setbackground	Determina el color de fondo de la película .
movie->setrate	Fija los “frames” por segundo de la película.
movie->setdimension	Determina el tamaño de la película (anchura, altura).
movie->nextframe	Mueve la película al siguiente “frame” de animación.

Fig. 9 - Principales métodos para un objeto de tipo “movie” (película)

5. TRABAJO REALIZADO

Introducción

En esta sección se comenta con detalle cómo se han utilizado las distintas herramientas con las que se ha desarrollado el presente proyecto. Se explican los pasos seguidos para, a partir de ficheros de flujo óptico, generar animaciones flash que representen a estos. También se comenta el trabajo realizado, que ha consistido en el desarrollo de dos aplicaciones capaces de generar ficheros SWF. Para llevar a cabo estas aplicaciones se han desarrollado dos clases SecOpticFlow y OpticFlow, fundamentales para llevar a cabo este proyecto. Al final de esta sección se muestran dos tablas en las que se detallan los métodos y miembros de cada clase, así como una descripción de cada uno de ellos.

El eje central de este proyecto ha consistido en desarrollar código C++ mediante el cual generar ficheros SWF que representen flujo óptico. Para ello, se parte de ficheros .uji que contienen la información de flujo óptico a representar. Dicha información se lee de ficheros .uji y se almacena en estructuras de datos con las que, mediante la librería Ming y la aplicación de varios métodos, se crea un objeto flash. Una vez creado dicho objeto flash, se genera un documento HTML en el cual se inserta la película flash, para así poder ser visualizada.

Los pasos seguidos en este proyecto para generar un fichero .SWF y mostrarlo en un documento HTML se pueden observar en el siguiente esquema:

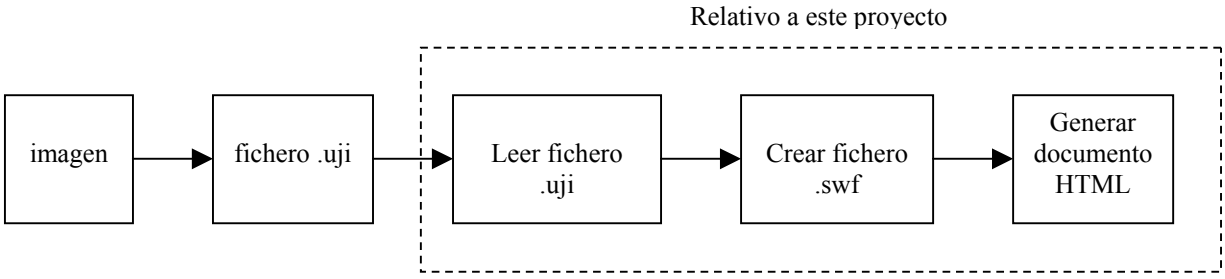
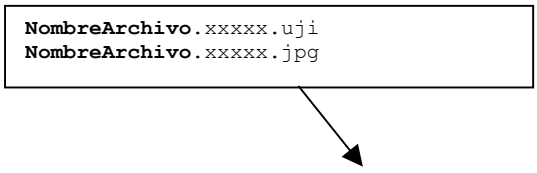


Fig. 10 Esquema de generación de una animación flash e inserción en una página web.

En este proyecto se ha llevado a cabo el desarrollo de dos programas que realizan las funciones descritas en el párrafo anterior. Estos programas son el *tSecOpticFlow* y el *HazAnimacion*. A continuación, se describe su funcionamiento y los parámetros que necesita cada uno:

tSecOpticFlow



Ejecución mediante:

```
./tSecOpticFlow Tipo_SWF Cuantos EscalaImagen EscalaVector Espaciado NombreArchivo
fichero.swf fichero.html
```

Este programa genera un fichero .SWF que contiene la representación del flujo óptico de una serie de ficheros .uji. Dicha representación puede mostrar el flujo óptico únicamente o éste superpuesto sobre las imágenes asociadas a él. A este programa se le tienen que indicar cuantos ficheros .uji se van a representar en la animación flash que se desea generar. También hay que indicarle otros parámetros cuyos valores se muestran en la siguiente tabla:

Parámetro	Valor
Tipo_SWF	Puede tomar los valores: 1 → Muestra sólo el flujo óptico 2 → Muestra el flujo óptico superpuesto a imágenes.
Cuantos	Número de ficheros .uji a representar
EscalaImagen	Modifica la escala del flujo óptico del fichero flash generado (normalmente vale 1).
EscalaVector	Modifica la longitud del vector del flujo óptico del fichero flash generado (2 o 3 suelen ser valores aceptables).
Espaciado	Modifica el número de "flechas" de flujo óptico que se mostrarán (valores entre 4 y 8 suelen mostrar buenas representaciones).
NombreArchivo	Nombre del archivo .uji y archivo .jpg a representar. Solo representa el comienzo del archivo. Por convenio, los ficheros deben llamarse NombreArchivo.xxxxx.uji y NombreArchivo.xxxxx.jpg (en caso de representarse con imágenes) y comenzando desde **NombreArchivo.00000**.
fichero.swf	Nombre del fichero de animación flash generado
fichero.html	Nombre del documento HTML que mostrará la animación flash.

Fig. 11 - Parámetros de la aplicación tSecOpticFlow

El código principal de esta aplicación es el contenido en el fichero *tSecOpticFlow.cpp*[9]. En la figura siguiente se muestra, en pseudocódigo, el funcionamiento básico de este programa.

```
      Crear objeto de flujo óptico;

      LeerFichero de flujo óptico;
      EsribirFicheroSWF;
      GenerarHtml;
}
```

Fig. 12 Pseudocódigo con funcionamiento básico del programa *tSecOpticFlow*

[9] *ver Apéndice I*

En este pseudocódigo se puede observar que la aplicación *tSecOpticFlow* tiene como tareas básicas crear un objeto SecOpticFlow (de flujo óptico), que será el que contenga los métodos necesarios para :

1. Leer el flujo óptico de un fichero .uji
2. Escribir el fichero de animación flash (SWF) que represente el flujo óptico.
3. Escribir un documento HTML que será el encargado de mostrar en un navegador la animación flash creada.

Para generar el fichero tSecOpticFlow ejecutable, se compilan los ficheros con el código desarrollado usando un fichero makefile, cuyo contenido es el siguiente:

```
MODULES=OpticFlow.o SecOpticFlow.o

#ESTE ES EL PRIMERO QUE SE EJECUTA AL HACER MAKE
all: tSecOpticFlow

tSecOpticFlow: $(MODULES) tSecOpticFlow.o
      g++ -o tSecOpticFlow tSecOpticFlow.o $(MODULES) -lming

# CREAR LOS .O PARA LAS CLASES
OpticFlow.o: OpticFlow.cpp OpticFlow.H
      g++ -c -o OpticFlow.o OpticFlow.cpp -I.

SecOpticFlow.o: SecOpticFlow.cpp SecOpticFlow.H
      g++ -c -o SecOpticFlow.o SecOpticFlow.cpp -I.

tSecOpticFlow.o: tSecOpticFlow.cpp
      g++ -c -o tSecOpticFlow.o tSecOpticFlow.cpp -I.

#BORRAR LOS .O Y LOS EJECUTABLES
clean:
      rm -f *.o
            rm -f tSecOpticFlow
```

Anteriormente, se ha comentado que para generar las animaciones flash se parte de la información de flujo óptico contenida en ficheros de texto del tipo "**nombre.uji**". A grandes rasgos, podría decirse que la información que contiene un fichero ".uji" es, entre otras cosas, el desplazamiento que han sufrido los píxeles de una imagen actual, a la que se refiere el fichero ".uji", a partir de una imagen anterior. Hay que considerar que ambas forman parte de una secuencia de imágenes de la cual se está estimando el movimiento producido.

El formato de un fichero de flujo óptico (.uji) de 100x100 (columnas,filas) es el siguiente:

```
Número de columnas
Número de filas
0       0       BUENO       DX     DY
0       1       BUENO       DX     DY
.....................................................
.....................................................
99      99      BUENO       DX     DY
```

Fig. 13 Formato de un fichero .uji de 100x100

Las dos primeras líneas indican el número de columnas y filas de la imagen asociada al flujo óptico que detalla el fichero ".uji".

Cada una de las líneas siguientes del fichero se refiere a un píxel de la imagen e indica tres valores:

- BUENO, que puede ser 0 ó 1:
  - 1 indica que el flujo de ese píxel ha de ser tenido en cuenta
  - 0 indica que el flujo de ese píxel **NO** ha de ser tenido en cuenta.

- DX es la componente x del vector desplazamiento (flujo óptico obtenido).
- DY es la componente y del vector desplazamiento (flujo óptico obtenido).

Existe un caso especial en el que DX=DY=100. En este caso se indica que el flujo óptico no se pudo calcular correctamente para el píxel en cuestión.

Un ejemplo de fichero .uji sería el mostrado en la figura.

```
256
256
0 0 1 5.04795 0.331378
0 1 1 5.04897 0.331221
0 2 1 5.04998 0.331064
0 3 1 5.051 0.330907
0 4 1 5.05202 0.33075
0 5 1 5.05303 0.330593
0 6 1 5.05405 0.330436
0 7 1 5.05506 0.330279
0 8 1 5.05608 0.330122
0 9 1 5.0571 0.329965
0 10 1 5.05811 0.329808
0 11 1 5.05913 0.329651
0 12 1 5.06015 0.329494
0 13 1 5.06116 0.329337
```

Fig. 14 Fragmento de un fichero de flujo óptico

5.1 Método de lectura de un fichero de flujo óptico

Para leer los datos de un fichero de flujo óptico se llama desde un objeto de tipo SecOpticFlow[10] al método LeerFichero. Este método crea un objeto de tipo OpticFlow[11] (de flujo óptico) y realiza una serie de acciones, tantas veces como ficheros de flujo óptico se tengan que leer. Las acciones a llevar a cabo son las siguientes:

Desde el objeto de tipo OpticFlow se llama a un método ReadFlowUJI, que es el encargado de leer del fichero de flujo óptico los valores de los atributos, descritos en el apartado anterior, para cada píxel y los guarda en vectores bidimensionales para utilizarlos posteriormente.

Una vez leídos estos valores y guardados en estructuras de datos pertenecientes al objeto de tipo OpticFlow, se almacena este objeto en una "lista de flujos ópticos".

El pseudocódigo que representa las acciones llevadas a cabo mediante el método ReadFlowUJI es el siguiente:

```
Abrir fichero .uji para lectura;

SI (Apertura es errónea) ENTONCES
     Informar del error;
SINO
     Leer número de columnas del fichero .uji;
     Leer número de filas del fichero .uji;

     Reservar memoria para almacenar datos de cada píxel;
     Leer línea del fichero .uji;

     MIENTRAS ( No Fin de fichero ) HACER
          Determinar si el valor del píxel leído es correcto;
          Leer componente x del flujo óptico;
          Leer componente y del flujo óptico;

          Leer línea del fichero .uji;
     FINMIENTRAS
     Cerrar fichero .uji;
FINSI
```

Fig. 15 Pseudocódigo con acciones para leer un fichero de flujo óptico

Una vez leído el fichero de flujo óptico, se tiene que generar la animación flash correspondiente. Esto se realiza mediante la llamada al método WriteFlash (ver Fig.) perteneciente a un objeto OpticFlow[11] (de flujo óptico). En la siguiente figura se pueden observar en pseudocódigo las principales acciones que tiene que llevar a cabo el método WriteFlash para generar un fichero SWF que contenga una representación de flujo óptico y, si corresponde, imágenes asociadas a éste.

[10] *Ver Apéndice III*
[11] *Ver Apéndice V*

```
Situarse al inicio de objeto "lista de flujos ópticos";
Crear objeto película;
Fijar dimensiones de la película;
Fijar fondo de la película;
Fijar velocidad de muestreo de la película;

PARA i=Primero_de_lista HASTA Ultimo_de_lista HACER

     Objeto "shape actual" = Representación flujo óptico i-ésimo de la lista (GetShape);

     Borrar shape anterior;
     Borrar imagen jpeg anterior;

     Añadir imagen a objeto shape actual; //si procede

     Mostrar objeto shape en película;
     Avanzar uno o varios frames;
FINPARA
Generar fichero SWF;
```

Fig. 16 Pseudocódigo con principales acciones para generar una animación flash

Para asignar a un objeto "shape" una representación de flujo óptico, en la aplicación tSecOpticFlow[12] se llama al método *GetShape*, que es el encargado de pintar cada una de las flechas que representan el flujo óptico. Todo esto puede observarse en el código del fichero OpticFlow.cpp[13]. Un ejemplo en pseudocódigo de las acciones llevadas a cabo por el método GetShape se muestra en la siguiente figura.

```
    Definir anchura de la línea a pintar
    Situarse en la posición inicial de la forma (shape)
    PARA columnas= 1 HASTA NumeroColumnas HACER
           PARA filas=1 HASTA NumeroFilas HACER
                  SI Valor leido es bueno ENTONCES
                           Pinta flecha de flujo optico
                  SINO
                           Pintar punto rojo
                 Cambiar de fila
           Cambiar de columna
    Devuelve shape con OF pintado
```

Fig. 17 Pseudocódigo con principales acciones del método GetShape para pintar el flujo óptico

Una vez leído el fichero de flujo óptico y generada la animación flash, resta simplemente generar un documento HTML mediante el cual se podrá mostrar la animación flash en un navegador. El código necesario para crear dicho documento HTML se muestra, a continuación, en el apartado "Insertar un fichero SWF en un documento HTML".

[12] *Ver Apéndice I*
[13] *Ver Apéndice V*

5.2 Insertar un fichero SWF en un documento HTML

Una de las formas de visualizar una película flash es insertar ésta en un documento HTML para mostrarla en una navegador de Internet. Hay que tener en cuenta que aún haciendo esto, el usuario debe tener instalado el plugin de flash para poder visualizar dicha película.

Es importante tener en cuenta que el tamaño de la presentación establecido en un documento HTML determina el tamaño real de la película. Las dimensiones de la película fijadas en el documento HTML determinan la escala en la que sus formas serán dibujadas. Si se fija el tamaño de la película en 3200x2400 y se empotra en una página HTML de 320x240, las formas serán 1/10 del tamaño que se especificó en los comandos de dibujo (métodos que proporciona Ming para generar el fichero SWF).

En la figura siguiente se puede observar un ejemplo de documento HTML en el cual se ha insertado un archivo SWF:

```
<html>
  <head>
    <title>Bitmap test</title>
  </head>
  <body>
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
      codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#vers
ion=4,0,0,0"
        ID=objects
        WIDTH=640
        HEIGHT=480>
<PARAM NAME=movie VALUE="rectangulo_animado.swf">
<EMBED src="./rectangulo_animado.swf"
       WIDTH=640
       HEIGHT=480
       TYPE="application/x-shockwave-flash"
      PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi
?P1_Prod_Version=ShockwaveFlash">
</OBJECT>
</table>
  </body>
</html>
```

La etiqueta **<object>** puede utilizarse para insertar diferentes objetos dentro de la página, como pueden ser archivos de audio, archivos de vídeo, imágenes, etc.

El objetivo del uso de la etiqueta **<object>** es el de que no se produzcan incompatibilidades por las distintas etiquetas soportadas por distintos navegadores.

Las animaciones flash se insertan del mismo modo que los archivos de audio y de vídeo, a través de la etiqueta **<embed>,** pero debido a que tiene más posibilidades de que se produzcan incompatibilidades entre los distintos navegadores, necesita también de la

etiqueta **<object>.** La etiqueta **<param>** se utiliza para especificar los valores necesarios para la inicialización de un objeto.

A continuación se especifica con más detalle el significado de las distintas etiquetas y los atributos de estas:

Etiqueta	Atributos
<embed>	• **pluginspage** : especifica la página desde la que se podrá descargar el plugin necesario para reproducir la animación flash, para que si algún usuario no lo tiene aún instalado en su ordenador pueda descargarlo. • **type** : especifica el tipo de fichero, para que el navegador pueda saber qué tipo de programa necesita ejecutar para reproducir la animación. • **src** : determina la ruta en la que se encuentra el fichero .swf a insertar. • **width y height** : especifican la anchura y altura del objeto flash
<object>	• **classid** : identifica al objeto. Cuando el objeto es una animación flash, el atributo classid debe valer ***clsid:D27CDB6E-AE6D-11cf-96B8-444553540000***. • **codebase** : especifica la dirección en la que se encuentran los componentes externos necesarios para reproducir la animación. • **width** y **height** : se utilizan del mismo modo, y deben tener el mismo valor, que en la etiqueta **<embed>**.
<param>	• **name** : indica el nombre de la característica que va a ser definida • **value** : indica su valor.

Fig. 18 Etiquetas y atributos para insertar un fichero SWF en un documento HTML

HazAnimacion

La otra aplicación creada en este proyecto se denomina *HazAnimacion*. Esta aplicación es una versión más reducida de la anterior, en la que simplemente se da la posibilidad de crear ficheros SWF (animaciones flash) compuestas por secuencias de imágenes.

Ejecución mediante:

```
./HazAnimacion NombreImagen cuantos fichero.swf fichero.html
```

Los posibles valores de cada uno de los parámetros son los siguientes:

Parámetro	Valor
NombreImagen	Nombre del fichero de imagen. Por convenio, se sigue el formato NombreImagen.xxxxx.jpg comenzando desde NombreImagen.00000.jpg
Cuantos	Número de imágenes a representar en la animación flash
fichero.swf	Nombre del fichero SWF a generar
fichero.html	Nombre del documento HTML que contendrá la animación flash.

Fig.19 Parámetros de la aplicación *HazAnimación*

El código mediante el cual se generan animaciones flash compuestas de secuencias de imágenes es muy similar al de la aplicación tSecOpticFlow. La única diferencia es que *HazAnimación* no tiene que hacer nada con objetos de flujo óptico, sino que únicamente tiene que asociar imágenes a objetos de tipo "shape" (forma) e ir insertando estos objetos en la película flash.

El código para generar una animación flash con *HazAnimación* está contenido en el fichero HazAnimacion.cpp[14]. A continuación, se muestra un ejemplo en pseudocódigo con los pasos principales para crear una animación flash mediante la aplicación *HazAnimación*:

```
Crear objeto película;
Fijar fondo;
Fijar velocidad muestreo;

PARA i=1 HASTA cuantas_imágenes HACER

     Borrar el shape anterior;
     Borrar el jpeg anterior;

     Obtener información imagen;
     Definir dimensiones de película a partir tamaño imagen;
     Añadir imagen a shape;
     Mostrar imagen jpeg en el shape actual;

     Avanzar varios frames;

FINPARA
Generar fichero SWF;
Generar documento HTML;
```

Fig. 20 Pseudocódigo con las principales acciones de la aplicación *HazAnimación*

[14] *Ver Apéndice VI*

A continuación se muestran dos tablas en las que detallan los métodos y miembros de las clases desarrolladas en el presente proyecto para crear las dos aplicaciones descritas anteriormente.

SecOpticFlow[15]

Componente	Descripción
Lista de objetos de tipo OpticFlow	Lista que contiene objetos de tipo OpticFlow (contienen la verdadera información de flujo óptico leída de un fichero .uji).
LeerFichero	Método que lee la información de flujo óptico de uno o varios archivos .uji. Crea un objeto de tipo OpticFlow Para cada fichero de flujo óptico: Lee la información del fichero .uji. Almacena dicha información en el objeto de tipo OpticFlow. Inserta el objeto de tipo OpticFlow en una lista.
WriteFlash	Genera el fichero SWF a partir de la información de cada uno de los objetos OpticFlow de la lista y/o de imágenes, si procede.
GeneraHtml	Genera el documento HTML en el que se mostrará la animación flash generada mediante WriteFlash.

Fig. 21 Tabla con descripción de los elementos que componen la clase SecOpticFlow

OpticFlow[16]

Parámetro	Descripción
_ncols	Número de columnas de una imagen representada en un fichero .uji
_nrows	Número de filas de una imagen representada en un fichero .uji
_buenOF	Valor de un determinado píxel leído del fichero .uji y que determina si el flujo óptico, para ese píxel, ha de ser tenido en cuenta.
_vx	Valor de un determinado píxel leído del fichero .uji y que representa el desplazamiento en el eje x para ese píxel.
_vy	Valor de un determinado píxel leído del fichero .uji y que representa el desplazamiento en el eje y para ese píxel.
AllocMemory	Método para gestionar las estructuras de datos de un objeto OpticFlow.
FreeMemory	Método para gestionar las estructuras de datos de un objeto OpticFlow.
CopyData	Método para gestionar las estructuras de datos de un objeto OpticFlow.
ReadFlowUJI	Método utilizado por “LeerFichero” para leer la información de flujo óptico de uno o varios ficheros .uji.
GetShape	Método que, para cada objeto OpticFlow de la lista, se encarga de pintarlo en la animación flash a generar.

Fig. 22 Tabla con descripción de los elementos que componen la clase OpticFlow

[15] *Ver Apéndice III*
[16] *Ver Apéndice V*

6. INSERTAR UN FICHERO SWF EN POWERPOINT

Las animaciones flash ofrecen la posibilidad de ser insertadas en presentaciones de Powerpoint para posteriormente ser utilizadas en contextos como la docencia o la investigación. Por ello, en esta sección se comentan los pasos necesarios para insertar un fichero SWF en una presentación de Powerpoint. En el presente proyecto, se ha seguido el procedimiento que se detalla a continuación para insertar animaciones flash que representen el flujo óptico.

Los pasos necesarios para insertar una película flash en una presentación de Powerpoint son los siguientes:

1. Abrir el Powerpoint y crear una presentación nueva con una diapositiva en blanco en la cual se creará una animación.

2. En la barra de menú principal, seleccionar *Ver* | *Barra de herramientas* | *Visual Basic*.

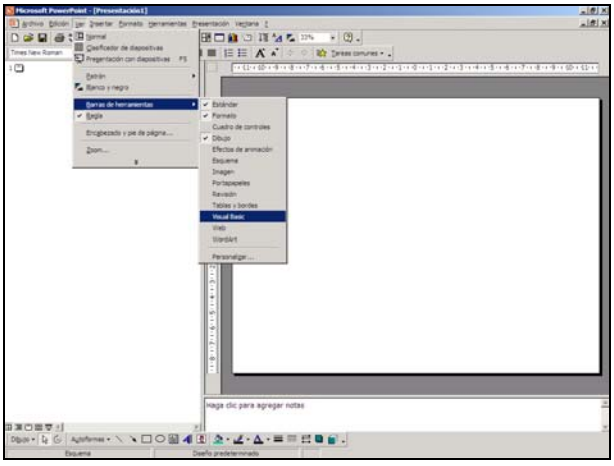


Fig. 23 Mostrar la barra de herramientas Visual Basic

Se mostrará la barra de herramientas de Visual Basic.

Fig. 24 Barra de herramientas Visual Basic

3. Seleccionar la opción "*Cuadro de controles*". En la barra que se muestra (ver Fig.22) seleccionar la opción "*Más controles*" y en la lista que aparece seleccionar "*Shockwave Flash Object*". Hasta aquí lo que se está haciendo es insertar un control Active X.

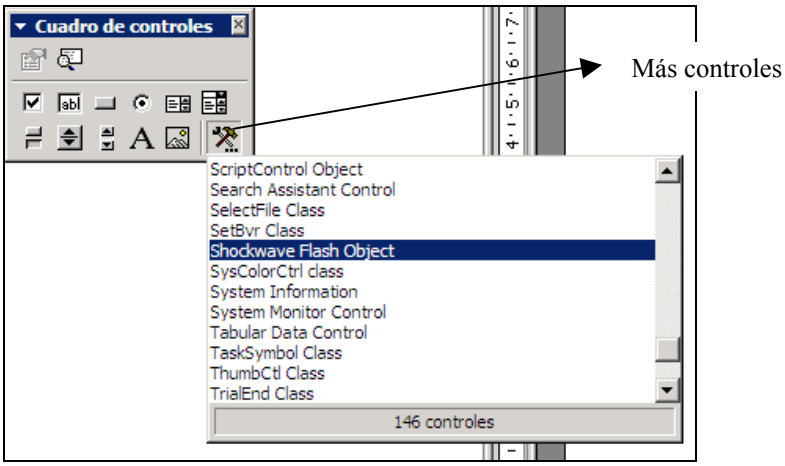


Fig. 25 Controles para insertar un Objeto Flash

4. El puntero del ratón se cambiará a una cruz. Hay que dibujar un rectángulo en la animación (diapositiva). El tamaño puede ser reajustado posteriormente.

5. Hacer clic con el botón derecho sobre el rectángulo que se acaba de dibujar y seleccionar la opción "*Propiedades*". Se abrirá una lista de propiedades de la película que se va a insertar (Ver Fig.23).

6. Para insertar la película hay que realizar los siguientes cambios en las propiedades:
   - *EmbedMovie* : Valor = True.
   - *Loop*: Valor = False.
   - *Movie*: Escribir la ruta absoluta de la película .swf

Una vez hecho esto ya tenemos la animación en el Powerpoint. Para visualizarla basta con seleccionar *Presentación* | *Ver presentación* o bien pulsar la tecla F5.

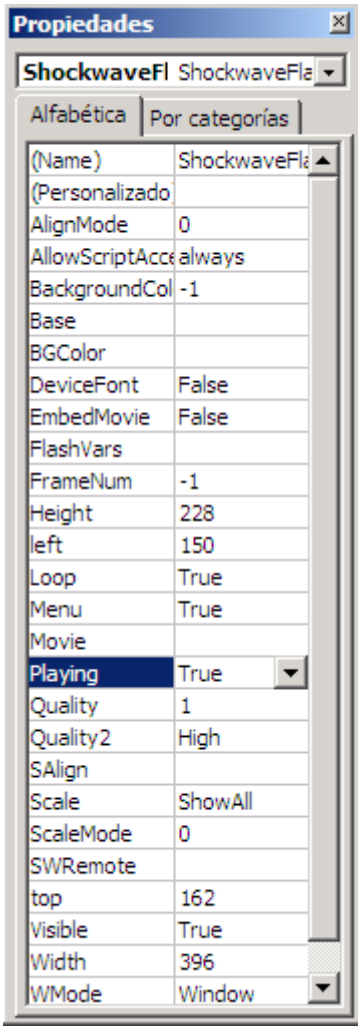


Fig. 26 Propiedades de la película Flash

7. RESULTADOS

Los resultados obtenidos a la finalización de este proyecto han sido los esperados al comienzo del mismo. Se han obtenido dos aplicaciones desarrolladas mediante el lenguaje de programación C++, junto con la librería STL y la librería Ming. Ambas aplicaciones permiten generar ficheros SWF los cuáles podemos utilizar para distintos fines. La primera de estas aplicaciones es tSecOpticFlow, cuyo objetivo es generar animaciones flash mediante las cuáles se representa el flujo óptico, calculado anteriormente y almacenado en ficheros .uji (ficheros de flujo óptico).

La otra aplicación se denomina HazAnimacion y su función es generar animaciones flash compuestas de secuencias de imágenes de las que se dispone en formato JPEG. Ambas aplicaciones generan, además de los ficheros de animaciones flash, documentos HTML en los que se inserta el objeto de animación flash. Este documento HTML permite mostrar la animación flash en cualquier navegador web que tenga el plugin para mostrar animaciones flash.

El objetivo de la creación de estas animaciones es utilizarlas posteriormente en áreas de investigación o de docencia, ya que mediante el uso de animaciones es mucho más fácil, por ejemplo, representar conceptos que evolucionan en el tiempo o procesos iterativos. Las animaciones flash también ofrecen ventajas como su posibilidad de uso en distintos tipos de sistemas, ya que el formato SWF está bastante extendido.

7.1 Ejemplos de animaciones flash creadas con la librería Ming

En el presente proyecto se han creado varias animaciones flash con el objetivo principal de representar el flujo óptico de una manera más interactiva de lo que se hacía hasta ahora mediante ficheros postscript. Los resultados obtenidos son muy interesantes, ya que dan la posibilidad de mostrar animaciones flash en las que se muestre, sólo el flujo óptico o éste superpuesto a la imagen de la cual se calcula dicho flujo óptico. También da la posibilidad de mostrar una secuencia formada por varias imágenes con el flujo óptico superpuesto a éstas, cosa que aporta mucha claridad en el estudio de dicho concepto. Además, estas animaciones flash que representan el flujo óptico disponen de una interactividad de la que los ficheros postscript carecen.

A continuación se muestran ejemplos de ficheros de tipo SWF (animaciones flash) generados, en el presente proyecto, mediante la librería Ming para representar el flujo óptico:

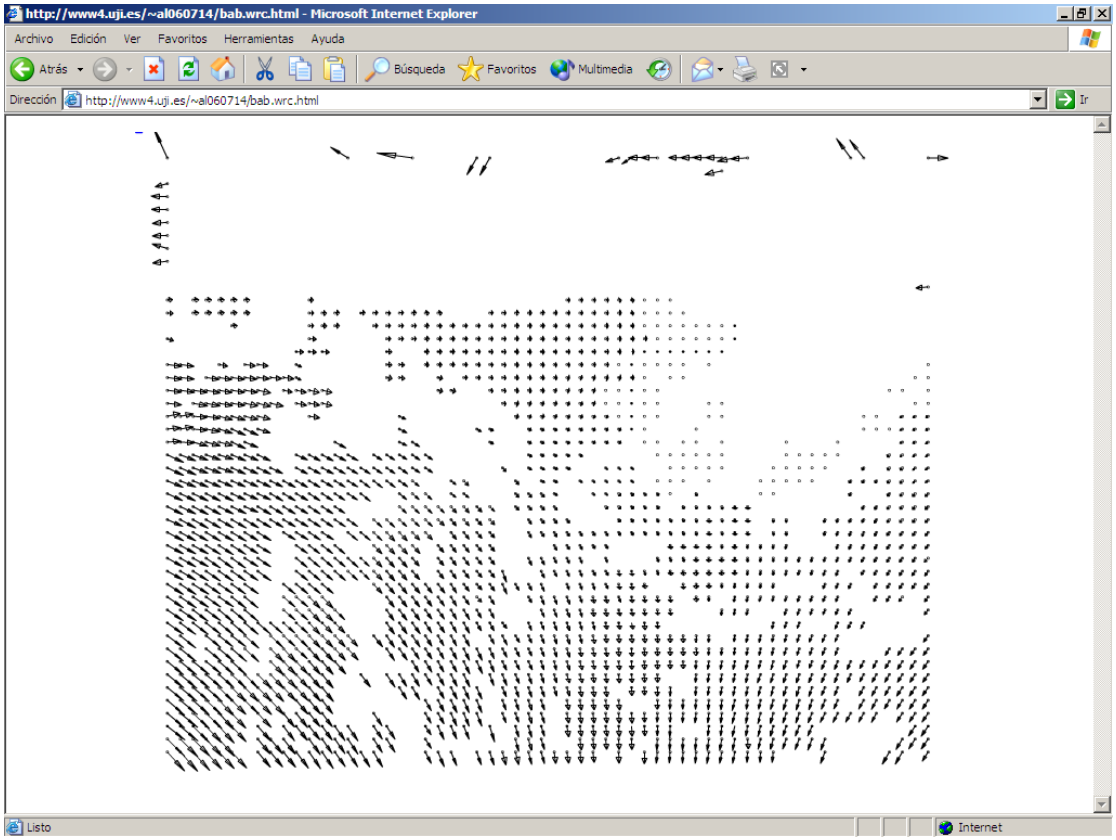


Fig. 27 Ejemplo de bab.wrc.00000.uji

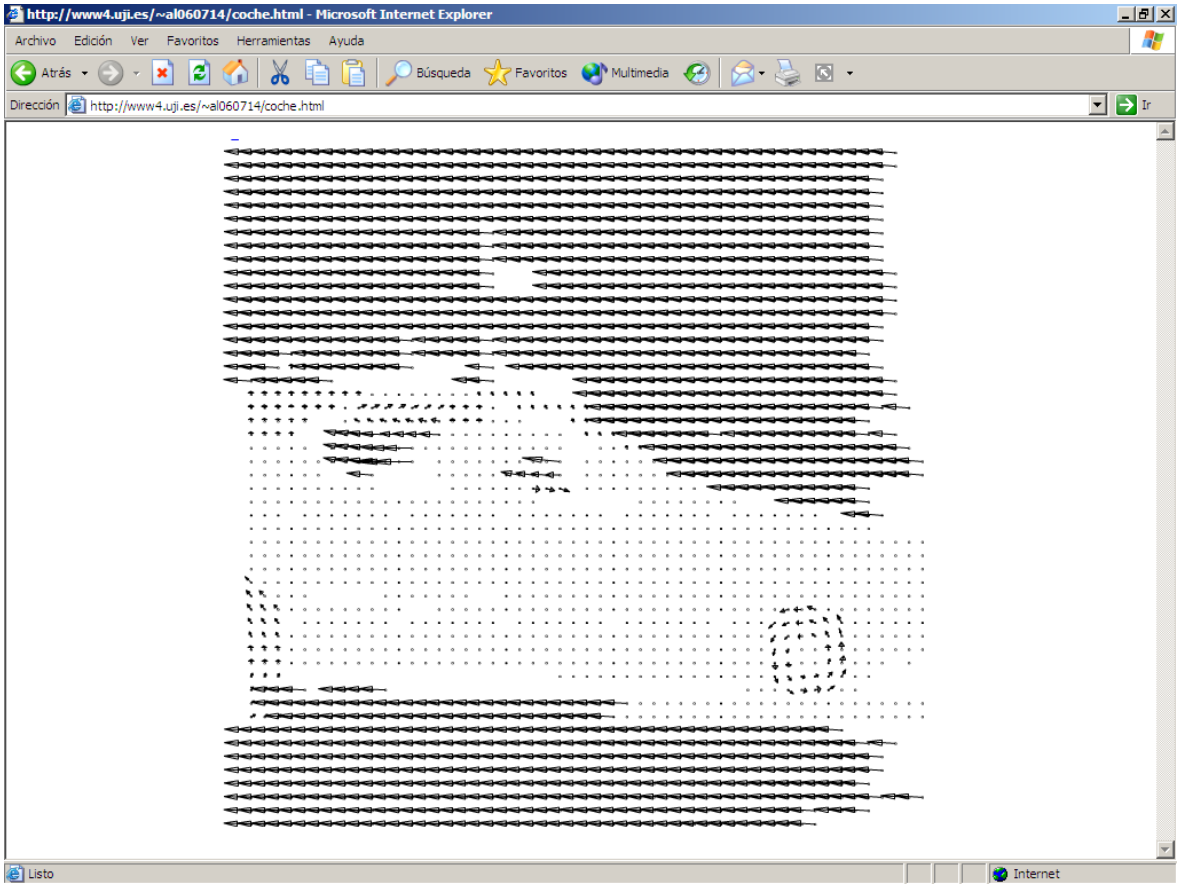


Fig . 28 Ejemplo de coche.00000.uji

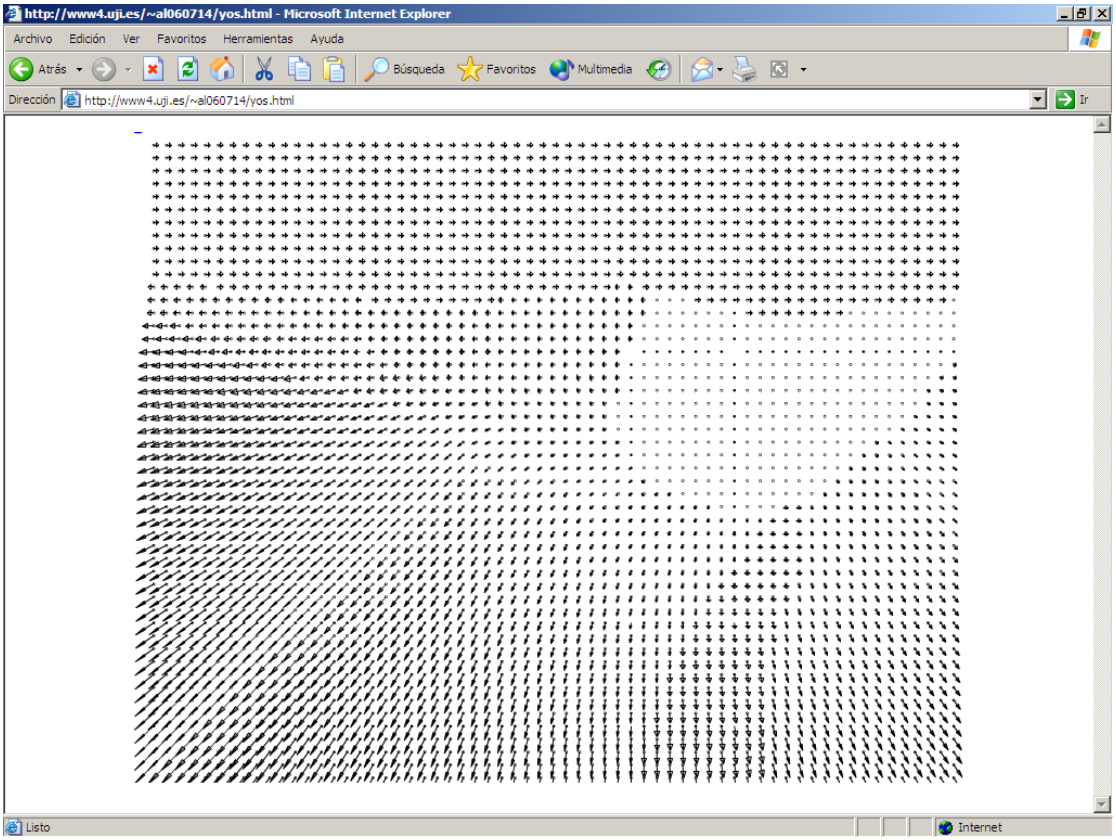


Fig. 29 Ejemplo de yos.00000.uji (yosemite) únicamente con flujo óptico.

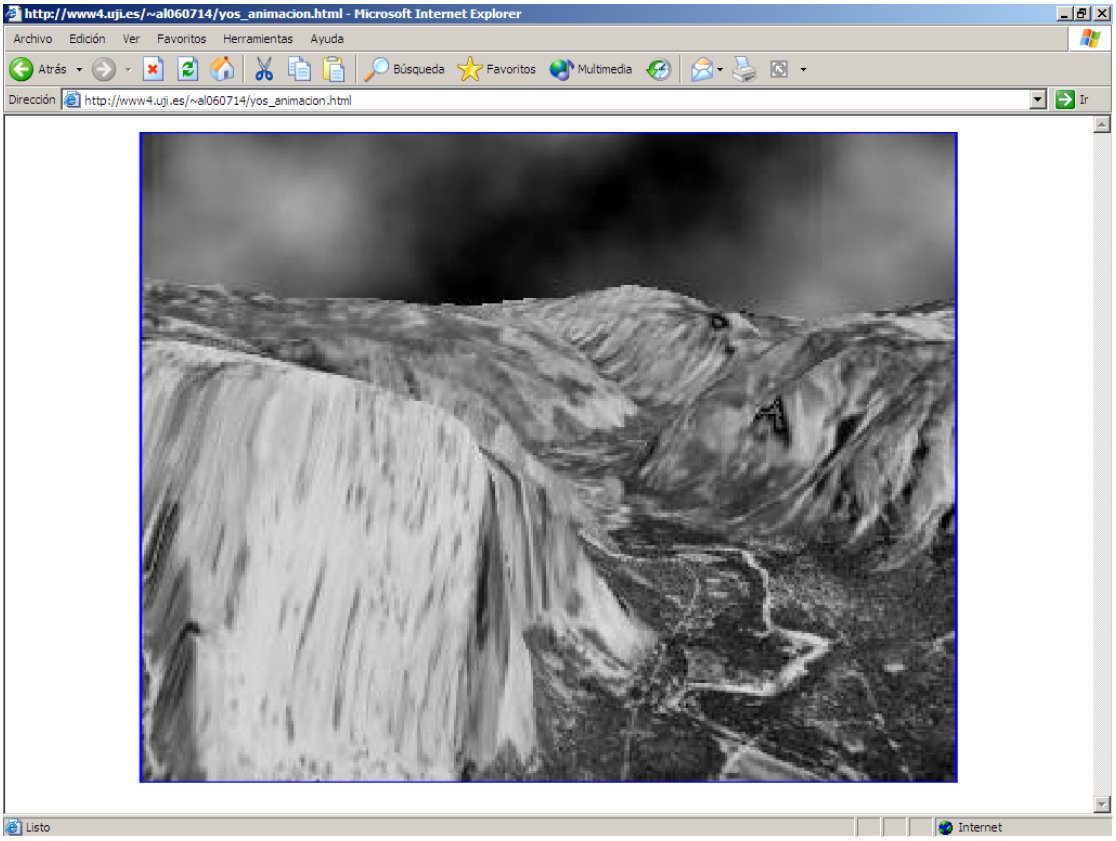


Fig. 30 Ejemplo extraído de una secuencia de imágenes de yosemite creado mediante HazAnimacion

Fig. 31 Ejemplo de una secuencia de imágenes, con el flujo óptico sobre éstas.

Como se ha citado anteriormente, también se pueden generar películas flash para mostrar de forma animada la evolución de procesos iterativos en cierto tipo de experimentos. A continuación, se muestra una secuencia de imágenes que representa las fusiones de regiones que se van produciendo con la ejecución del algoritmo de segmentación Sieve para una imagen en escala de grises.

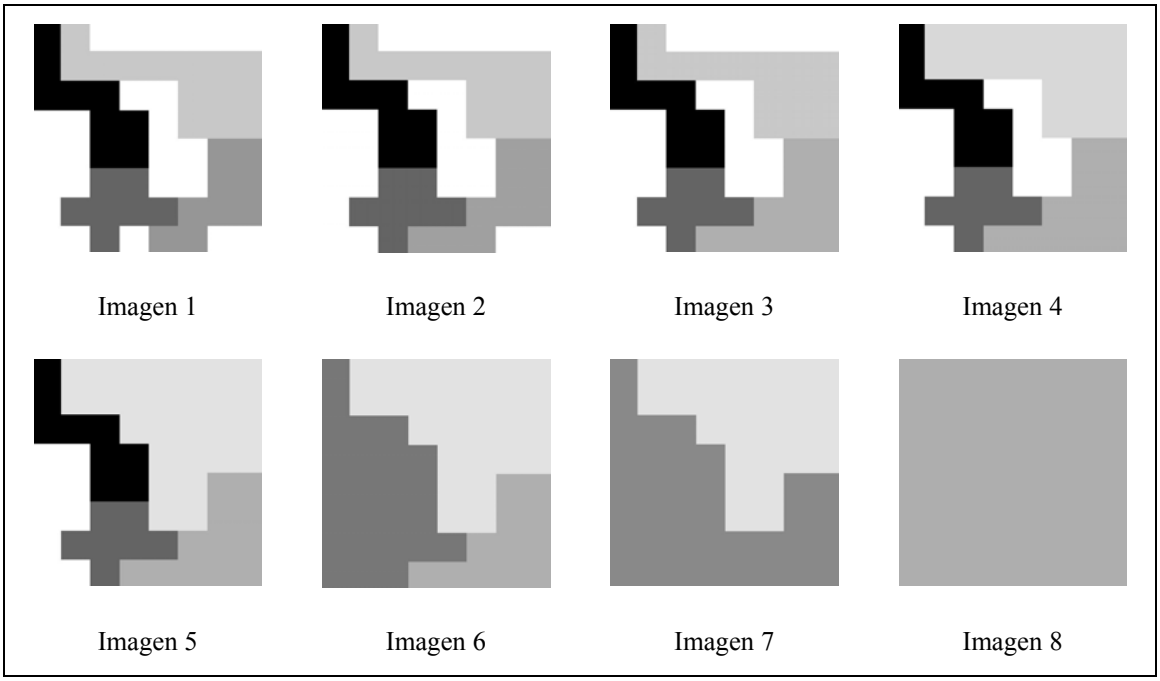


Fig.32 Imágenes de una película flash que muestra un ejemplo de evolución del algoritmo Sieve.

La siguiente figura muestra cómo, si se pulsa el botón derecho del ratón sobre una animación flash que se ha generado mediante el uso de la librería Ming, aparece un menú con opciones para controlar dicha animación. Como ya se ha citado anteriormente, esta es una de las ventajas que aporta el uso de ficheros SWF, frente a los ficheros postscript, para representar flujo óptico.

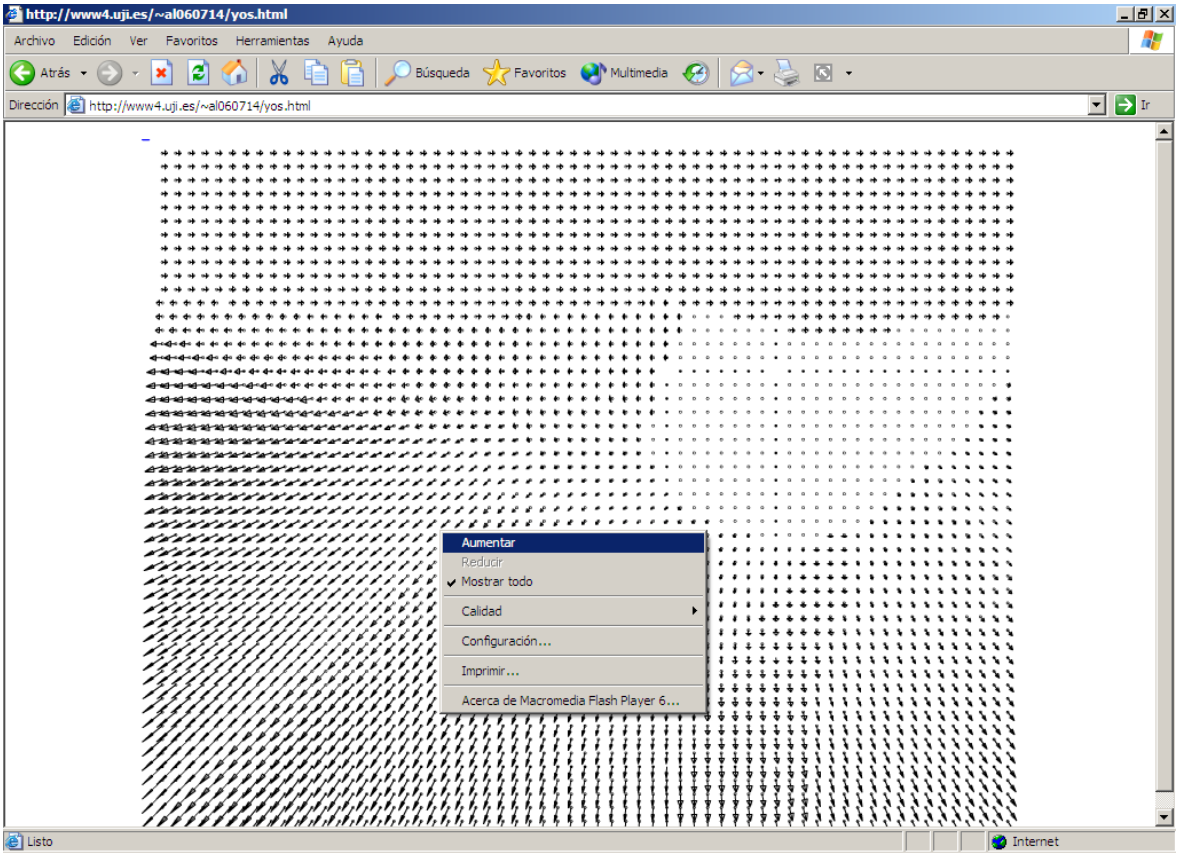


Fig.33 Menú con opciones para controlar la animación flash

Gracias a la posibilidad que ofrecen los ficheros SWF de llevar a cabo un "zoom" de cualquier zona de la animación flash, se pueden observar en estas ciertos detalles que podrían pasar inadvertidos si se usara otro método de representación, como podrían ser los ficheros postscript.

Un ejemplo significativo de esto lo encontramos en la animación flash que se muestra en la figura 25. En ella, se muestra el flujo óptico de una imagen en la que aparece un coche. Gracias a las opciones de zoom, en este ejemplo se puede observar el leve movimiento de rotación de la rueda del coche, algo que mediante ficheros postscript podría no ser observado debido a la carencia de opciones de "zoom" de las que no disponen este tipo de ficheros.

En la figura siguiente, se muestra el resultado de aplicar un "zoom" sobre la zona de la animación flash en la que se encuentra la rueda.

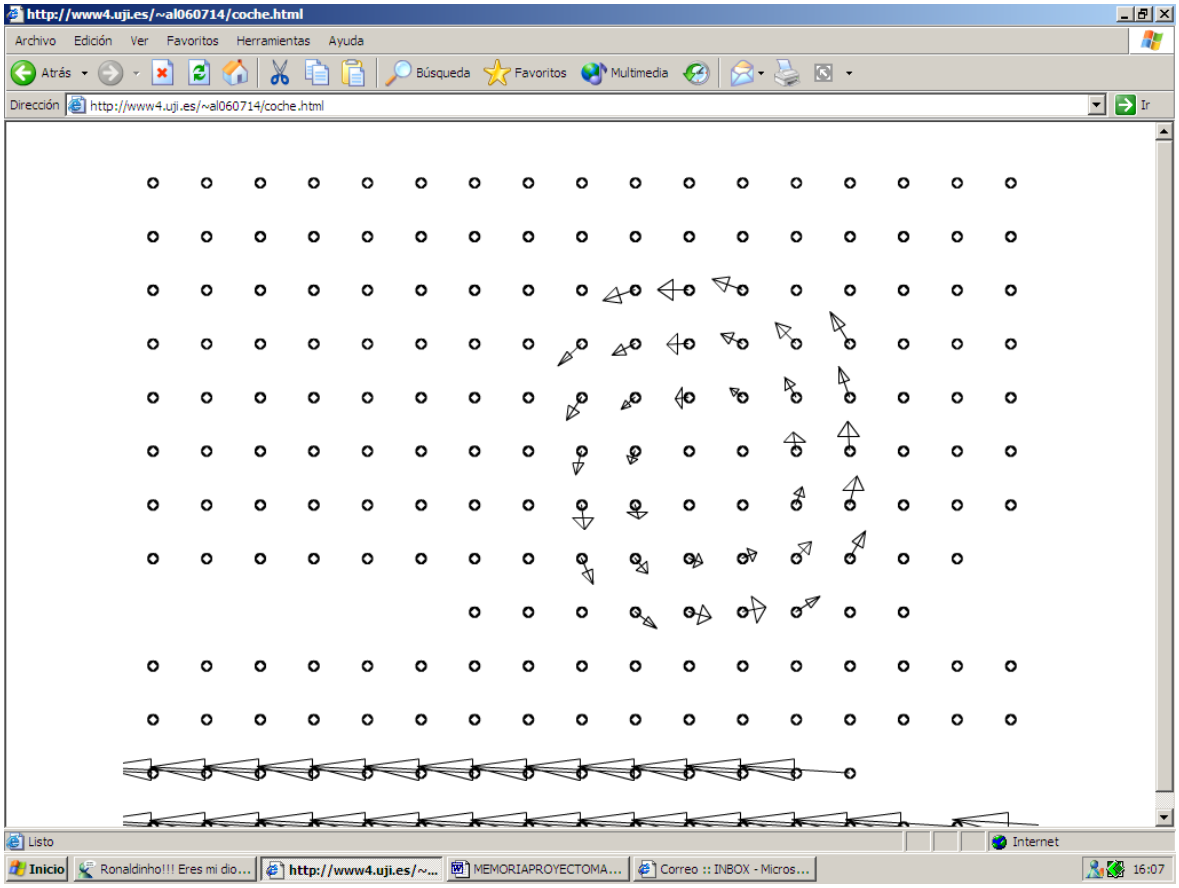


Fig.34 Animación flash en la que se ha hecho un zoom

8. CONCLUSIONES

Tras llevar a cabo este proyecto se han implementado dos aplicaciones capaces de generar ficheros SWF que representen el flujo óptico. Con una de ellas, generamos ficheros SWF con los que conseguimos mostrar el flujo óptico de manera interactiva, ya que en las animaciones flash se pueden realizar tareas como acercamiento/alejamiento de una zona de la animación, crear animaciones que muestren solo el flujo óptico, o éste superpuesto a las imágenes asociadas a él, avanzar o rebobinar la animación, etc...

La otra aplicación nos permite crear animaciones flash compuestas de secuencias de imágenes de las que disponemos en ficheros JPEG.

A titulo personal, he aprendido la idea básica sobre en qué consiste el flujo óptico. También he podido comprobar que la librería Ming ofrece ciertas ventajas para generar ficheros SWF. Algunas de estas ventajas son la posibilidad de generar animaciones flash a partir de código en un determinado lenguaje de programación , en lugar de tener que usar una aplicación comercial como, por ejemplo, Macromedia Flash.

Esta característica es muy importante ya que, si se quiere generar una animación flash a partir de los resultados de ciertos experimentos, puede darse el caso de que el número de elementos a representar en la animación sea muy elevado. Así, crear un fichero SWF donde se tengan tantos elementos puede resultar un trabajo muy costoso de realizar mediante una aplicaciones comerciales, ya que la mayoría de estas suelen tener un funcionamiento más bien “manual” (hay que trabajar elemento a elemento). Sin embargo, la utilización de lenguajes de programación (con sus estructuras de control) y una serie de métodos, como los que ofrece la librería Ming, convierten una tarea como esta algo muy factible. Por todo esto, veremos que el uso de la librería Ming puede ser muy recomendable, no solo en las tareas en las que se ha utilizado en el presente proyecto sino en otras tareas futuras enmarcadas dentro de áreas como la investigación, la docencia o incluso a nivel de usuario doméstico.

En cuanto a la investigación, se pueden utilizar animaciones flash para representar secuencias que simulen procesos iterativos, aportando mayor interactividad a la presentación de dichos procesos. En la docencia, se pueden las animaciones flash, como herramientas para explicar conceptos cuya representación temporal sea compleja, sin mecanismos que permitan mostrar una secuencia de imágenes. Por último, a nivel doméstico se puede utilizar para generar animaciones flash de lo que uno desee como, por ejemplo, para publicar en el web.

La librería Ming ofrece otra ventaja en la creación de animaciones flash, como es la facilidad de uso de los distintos métodos que ofrece, tanto por su simplicidad como por el hecho de contar con tutoriales y manuales de referencia que resuelven cualquier duda que pueda surgir.

La utilización de ficheros flash para representar flujo óptico tiene la ventaja de que, generalmente, cualquier sistema, suele contar con un navegador de internet que permita la representación de objetos flash insertados en documentos HTML, mientras que no todos los sistemas cuentan con algún programa que permita abrir ficheros de tipo postscript.

En definitiva, el uso de la librería Ming para generar ficheros SWF es muy aconsejable, no sólo en la generación de animaciones flash que representen el flujo óptico, sino también para otras finalidades o contextos, todo ello debido a las ventajas que aporta el uso de dicha librería como se ha comentado a lo largo de este documento.

9. BIBLIOGRAFÍA

[1] Web principal de Ming, http://ming.sourceforge.net/

[2] Web con ejemplos de animaciones flash, http://www4.uji.es/~al060714/index.html

[3] Web con lecciones sobre el funcionamiento de la librería Ming y ejemplos de animaciones flash, http://www.dataturn.com/mingdocs/

[4] "Performance of Optical Flow Techniques" Barron, J.L. and Fleet, D.J. and Beauchemin, S.S. "International Journal of Computer Vision", v12, n1. pp.43-77. 1994

[5] "Curso Práctico de programación en C y C++". Jorge Badenas, José Luis Llopis, Oscar Coltell. Col·lecció Manuals/8. Publicacions de la Universitat Jaume I. 1997

10. APÉNDICE

APÉNDICE I

tSecOpticFlow.cpp

```
#include <stdio.h>
#include <SecOpticFlow.H>

//tSecOpticFlow Tipo_SWF Cuantos EscalaImagen EscalaVector Espaciado
  archivo fichero.swf fichero.html
// archivo ( .xxxxx.uji  y .xxxxx.jpg )
//Tipo_SWF :  1 --> OF ; 2 --> OF y JPG ;

int main(int argc,char *argv[])
{
      SecOpticFlow sof;

      sof.LeerFichero(argv[6],atoi(argv[2]));
      int EscalaImagen = atoi(argv[3]);
      int EscalaVector = atoi(argv[4]);
      int Espaciado    = atoi(argv[5]);
      sof.WriteFlash(atoi(argv[1]),argv[7],argv[6],EscalaImagen,
                     EscalaVector,Espaciado);
      sof.GeneraHtml(argv[8],argv[7]);
}
```

APÉNDICE II

SecOpticFlow.H

```
#ifndef _SECOPTICFLOW_H_
#define _SECOPTICFLOW_H_
#include <SecOpticFlow.H>
#include <OpticFlow.H>
#include <string>
#include <list>

using namespace std;

class SecOpticFlow
{
private:

  list<OpticFlow> _l_of;
  int _ncolumnas, _nfilas;

public:
  SecOpticFlow();

  void LeerFichero(const string &name,const int cuantos);
  void WriteFlash(const int TipoSWF,const char *name,const char
                  *jpgname,int EscalaImagen,int EscalaVector, int
                   Espaciado);
  void GeneraHtml(const string &htmlname,const char *name) const;
};
#endif
```

APÉNDICE III

SecOpticFlow.cpp

```
#include <SecOpticFlow.H>
#include <OpticFlow.H>
#include <string.h>
#include <iostream>
#include <strstream>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265

SecOpticFlow::SecOpticFlow()
{
  _ncolumnas = 0;
  _nfilas = 0;
}

void SecOpticFlow::LeerFichero(const string &name,const int cuantos)
{
      OpticFlow of;
      string aux;

      for (int i=0; i< cuantos; i++)
      {
            ostrstream ficherouji;
            int longitud;
            ficherouji << name.substr(0,longitud-9) << "."<< setw(5) <<
                        setfill('0') << i << ".uji" << ends;
            of.ReadFlowUJI(ficherouji.str());
            _l_of.push_back(of); //mete of en la lista
      }
}


void SecOpticFlow::WriteFlash(const int TipoSWF,const char *name,const
char *jpgname,int EscalaImagen,int EscalaVector, int Espaciado)
{
      list<OpticFlow>::iterator indice = _l_of.begin();

      // para poder poner el numero de filas y columnas en el fichero
         html que se genera posteriormente.
      this->_ncolumnas=indice->GetCols();
      this->_nfilas=indice->GetRows();

      Ming_init();
      SWFMovie *movie = new SWFMovie();
      movie->setDimension(EscalaImagen*(indice->GetCols()),
                          EscalaImagen*(indice->GetRows()));
      movie->setBackground(255,255,255);
      movie->setRate(16.0);

      // en cada frame se dibuja cada OF
      SWFShape *shape, *shape_ant,*shapejpg,*shapejpgant,*shapeaux;
      SWFDisplayItem *anterior,*actual,*jpg,*jpgant;
      list<OpticFlow>::iterator it;
      SWFBitmap *b;
```

```
	shape_ant = new SWFShape();
	if (TipoSWF == 2) shapejpgant = new SWFShape();

	anterior = movie->add(shape_ant);  // inicializacion
	if (TipoSWF == 2) jpgant = movie->add(shapejpgant);

	int vez=0;
	for (it=_l_of.begin(); it!=_l_of.end(); it++)
	{
		shapeaux = new SWFShape();
		shape_ant = new SWFShape();
		if (TipoSWF == 2) shapejpg = new SWFShape();
		if (TipoSWF == 2) shapejpgant = new SWFShape();

		//************* calcula el nuevo shape ******************
		shape = it->GetShape(EscalaImagen,EscalaVector,Espaciado);
		movie->remove(anterior);     // borra el shape anterior

		// prepara el nuevo jpg para mostrarlo en shapejpg
		if (TipoSWF == 2) movie->remove(jpgant);

		ostrstream ficherojpg;
		if (TipoSWF == 2)
		{
			int longitud;
			const string nombre =(string ) jpgname;
			ficherojpg << nombre.substr(0,longitud-9) << "."<<
			 setw(5) << setfill('0') << vez << ".jpg" << ends;
		}
		b = new SWFBitmap(ficherojpg.str());

		//asignar imagen a "shape"

		shapeaux->setRightFill(shapeaux->addBitmapFill(b));
		shapeaux->setLine(1,0,0,255);

		//recuadro que rodeara a la imagen
		//pongo un 2 porque sino no se visualiza bien el recuadro
		shapeaux->drawLine((b->getWidth()-2)*EscalaImagen,0);
		shapeaux->drawLine(0,(b->getHeight()-2)*EscalaImagen);
		shapeaux->drawLine((-b->getWidth()+2)*EscalaImagen,0);
		shapeaux->drawLine(0,(-b->getHeight()+2)*EscalaImagen);

		shapejpg = shapeaux;
		jpg = movie->add(shapejpg);
		actual = movie->add(shape);  //muestra el nuevo shape

		movie->nextFrame();

		shape_ant = shape;
		if (TipoSWF == 2) shapejpgant = shapejpg;
		anterior = actual;
		if (TipoSWF == 2) jpgant = jpg;
		vez++;
	}//for
	movie->save(name);
}
```

```
void SecOpticFlow::GeneraHtml(const string &htmlname, const char *name)
const
{
  ofstream salida (htmlname.c_str());
  if (!salida) {
     cerr << "Fallo en apertura de fichero" << htmlname << "\n";
     exit(1);
  }
  else
  {
      salida << "<html>" << endl;
      salida << "   <head>" << endl;
      salida << "   </head>" << endl;
      salida << "   <body>" << endl;
      salida << "      <center>"<< endl;
      salida << "<OBJECT classid=\"clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000\"" << endl;//
      salida <<
"codebase=\"http://active.macromedia.com/flash2/cabs/swflash.cab#version=
4,0,0,0\""<<endl;//
      salida << "ID=objects"<< endl;
      salida << "WIDTH=" << _ncolumnas <<endl;
      salida << "HEIGHT=" << _nfilas << endl;
      salida << "<PARAM NAME=movie VALUE=\"" << name << "\">"<<endl;//
      salida << "<EMBED src=\""<< name << "\""<<endl;//
      salida << "TYPE=\"application/x-shockwave-flash\""<<endl;//
      salida <<
"PLUGINSPAGE=\"http://www.macromedia.com/shockwave/download/index.cgi?P1_
Prod_Version=ShockwaveFlash\""             << endl;
      salida << "WIDTH=" << _ncolumnas << endl;
      salida << "HEIGHT=" << _nfilas << endl;
      salida << "</object>" << endl;
      salida << "</center>" << endl;
      salida << "</body>" << endl;
      salida << "</html>" << endl;
  }
  salida.close();
}
```

APÉNDICE IV

OpticFlow.H

```
#ifndef _OPTICFLOW_H_
#define _OPTICFLOW_H_
#include <OpticFlow.H>
#include <string>
#include <mingpp.h>

using namespace std;

class OpticFlow
{
private:
  double **_vx;
  double **_vy;
  int      _ncols,
           _nrows;
  bool   **_buenOF;

  void AllocMemory();
  void FreeMemory();
  void CopyData(const OpticFlow &of);

public:

  OpticFlow();
  OpticFlow(const int mc, const int mf);
  OpticFlow(const OpticFlow &of);
  OpticFlow & operator=(const OpticFlow &of);
  ~OpticFlow();

  void ReadFlowUJI(const string &name);
  void WriteFlowUJI(const string &name) const;
  SWFShape * GetShape(int EscalaImagen, int EscalaVector,int Espaciado);
  int GetCols();
  int GetRows();
};
#endif
```

APÉNDICE V

OpticFlow.cpp

```
#include <OpticFlow.H>
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <math.h>

OpticFlow::OpticFlow()
{
 _vx=NULL;
 _vy=NULL;
 _buenOF =NULL;
 _ncols=0;
 _nrows=0;
}

OpticFlow::OpticFlow(const int mc, const int mf)
{
 _vx=NULL;
 _vy=NULL;
 _buenOF =NULL;
 _ncols=mc;
 _nrows=mf;
}

OpticFlow::OpticFlow(const OpticFlow &of)
{
  _ncols=of._ncols;
  _nrows=of._nrows;
  AllocMemory();
  CopyData(of);
}
```

```
OpticFlow & OpticFlow::operator=(const OpticFlow &of)
{
 FreeMemory();
 _nrows=of._nrows;
 _ncols=of._ncols;
 AllocMemory();
 CopyData(of);

 return *this;
}

OpticFlow::~OpticFlow()
{
 FreeMemory();
}

void OpticFlow::AllocMemory()
{
   _vx=new double *[_ncols];
   _vy=new double *[_ncols];
   _buenOF=new bool *[_ncols];

 for (int i=0; i<_ncols; i++)
 {
   _vx[i]=new double [_nrows];
   _vy[i]=new double [_nrows];
   _buenOF[i]=new bool[_nrows];
 }
}

void OpticFlow::FreeMemory()
{
  if (_vx != NULL) // si lo es _vx tambien lo sera _vy y _buenOF
  {
    for (int c=0; c<_ncols; c++)
    {
      delete [] _vx[c];
      delete [] _vy[c];
      delete [] _buenOF[c];
    }
```

```
    delete [] _vx;
    delete [] _vy;
    delete [] _buenOF;
  }
}

void OpticFlow::CopyData(const OpticFlow &of)
{
  for (int c=0; c<_ncols;c++)
      for (int r=0; r<_nrows; r++)
      {
            _vx[c][r]=of._vx[c][r];
            _vy[c][r]=of._vy[c][r];
            _buenOF[c][r]=of._buenOF[c][r];
      }
}

int OpticFlow::GetCols()
{
      return _ncols;
}

int OpticFlow::GetRows()
{
      return _nrows;
}


void OpticFlow::ReadFlowUJI(const string &name)
{
      int c,r,bueno;
      double vx,vy;
      //abrimos el fichero
      ifstream entrada(name.c_str(), ios::in);
      if (!entrada){
            cerr << "Fallo en apertura del fichero " << name << "\n";
            exit(1);
      }
      else
      {
```

```
            entrada >> _ncols;
            entrada >> _nrows;
            //reservamos memoria
            AllocMemory();
            entrada >> c >> r >> bueno >> vx >> vy;
            while(!entrada.eof())
            {
                  if ( bueno )
                  _buenOF[c][r]=true;
                  else
                  _buenOF[c][r]=false;
                  _vx[c][r]=vx;
                  _vy[c][r]=vy;
                  entrada >> c >> r >> bueno >> vx >> vy;
            }//while
            entrada.close();
      }//else
}//ReadFlowUJI

void OpticFlow::WriteFlowUJI(const string &name) const
{
      int esbueno;

      ofstream salida (name.c_str());
      if (!salida) {
            cerr << "Fallo en apertura de fichero" << name << "\n";
            exit(1);
      }
      else
      {
            salida << _ncols << endl;
            salida << _nrows << endl;
            for (int c=0; c<_ncols; c++)
                  for (int r=0; r<_nrows; r++)
                  {
                        if (_buenOF[c][r])
                              esbueno = 1;
                        else
                              esbueno =0;
                        salida << c << " " << r <<" "<<  esbueno<<" "<<
```

```
                                _vx[c][r] <<" "<< _vy[c][r]<<endl;
                    }
      }//else
      salida.close();
}//WriteFlowUJI

SWFShape *  OpticFlow::GetShape(int EscalaImagen, int EscalaVector,
                                int Espaciado)
{
// anchura del cuadrado "origen" de la flecha
  double lado=0.1;

  SWFShape *shape = new SWFShape();

  double x,y,xfinal,yfinal,incx,incy,m,mprima;
  double xintermedio,yintermedio,xintermedio_izq,xintermedio_der;
  double ynuevo_izq,ynuevo_der,dist;

  shape->movePenTo(Espaciado*EscalaImagen,Espaciado*EscalaImagen);

  x = Espaciado*EscalaImagen;
  y = Espaciado*EscalaImagen;
  for (int c=(Espaciado*EscalaImagen); c<_ncols; c=c+(Espaciado*EscalaImagen))
  {
      for (int r=(Espaciado*EscalaImagen); r<_nrows;
           r=r+(Espaciado*EscalaImagen))
      {
            if ( _buenOF[c][r]) // se debe dibujar
            {
                  shape->setLine(1, 0, 0, 0);
                  shape->drawLine(lado,0);shape->drawLine(0,lado);
                  shape->drawLine(-lado,0);shape->drawLine(0,-lado);

                  //desplazamiento
                  // positivas para ejemplo yosemite

                  incx=-_vx[c][r]*(double) EscalaVector;
                  incy=-_vy[c][r]*(double) EscalaVector;

                  xfinal = x + incx;
```

```
                    yfinal = y + incy;

                    dist = sqrt(pow(incy,2.0)+pow(incx,2.0));
                    if (dist > 1.0)
                    // se dibujan los que tengan
                    al menos una distancia minima
                    {
                      shape->setLine(lado, 0, 0, 0);
                      shape->drawLineTo(xfinal,yfinal);
                      m=incy/incx; //pendiente
                      if ( incx > 0.0 )  //derecha
                      {
/*****************   cuadrante arriba-derecha *************************/
                          if ( incy < 0.0 )
                          {
                                if ( m  > -0.2 ) // 0 -15 grados
                                {
                                  xintermedio = xfinal-(incx/EscalaVector);
                                  yintermedio = yfinal-(incy/EscalaVector);
                                   ynuevo_izq = yintermedio - 1.0;
                                   ynuevo_der = yintermedio + 1.0;

                                   if ((incy/EscalaVector)>-0.5)
                                      xintermedio_izq=xintermedio-0.5;
                                   else xintermedio_izq= xintermedio;

                                    xintermedio_der = xintermedio ;
                                       }
                                       else
                                       {
                                             if ( m > -1.0 ) //15-45 grados
                                             {
                                                   // punto de interseccion de las rectas
                                                   xintermedio = xfinal-(incx/EscalaVector);
                                                   yintermedio = (m*(xintermedio-x))+y;

                                                   // puntos xintermedios
                                                   xintermedio_izq = (xintermedio - 0.25);
                                                   xintermedio_der = (xintermedio + 0.25);
                                                   //pendiente prima
```

```
                                    mprima = (-1/m);
                                    ynuevo_izq = (mprima* (xintermedio_izq - xintermedio)) + yintermedio;
                                    ynuevo_der = (mprima* (xintermedio_der - xintermedio)) + yintermedio;
                              }
                              else
                              {
                                    if ( m > -3.75 ) // 45-75 grados
                                    {
                                          // punto de interseccion de las rectas
                                          xintermedio = xfinal -(incx/EscalaVector);
                                          yintermedio = (m*(xintermedio-x))+y;

                                          // puntos xintermedios
                                          xintermedio_izq = (xintermedio - 0.5);
                                          xintermedio_der = (xintermedio + 0.5);
                                          //pendiente prima
                                          mprima = (-1/m);
                                          ynuevo_izq = (mprima* (xintermedio_izq - xintermedio)) +
                                                       yintermedio;
                                          ynuevo_der = (mprima* (xintermedio_der - xintermedio)) +
                                                       yintermedio;
                                    }
                                    else // 75-90
                                    {
                                          xintermedio = xfinal - (incx/EscalaVector);
                                          yintermedio = yfinal - (incy/EscalaVector);
                                          xintermedio_izq = (xintermedio - 1.0);
                                          xintermedio_der = (xintermedio + 1.0);

                                          //cambio incx por incx/EscalaVector
                                          if ((incx/EscalaVector) > 0.5 ) ynuevo_der = yintermedio + 0.5;
                                          else ynuevo_der= yintermedio;
                                          ynuevo_izq = yintermedio ;
                                    }//else 75-90
                              } // else 45-75
                        }// else 0-15
                  }//incx > 0 ; incy < 0
                  else
                  {
/*********************************   cuadrante abajo-derecha ************************************/
```

```
if (incy > 0.0 ) // abajo
{
      if ( m < 0.2 ) // 345-360
      {
            xintermedio = xfinal -(incx/EscalaVector);
            yintermedio = yfinal -(incy/EscalaVector);
            ynuevo_izq = yintermedio + 1.0;
            ynuevo_der = yintermedio - 1.0;

            //camibo incy por incy/EscalaVector
            if ((incy/EscalaVector) > 0.5 ) xintermedio_izq = xintermedio - 0.5;
            else xintermedio_izq= xintermedio;
            xintermedio_der = xintermedio ;
      }
      else
      {
            if ( m < 1.0 )//285-315 grados
            {
                  // punto de interseccion de las rectas
                  xintermedio = xfinal -(incx/EscalaVector);
                  yintermedio = (m*(xintermedio-x))+y;

                  // puntos xintermedios
                  xintermedio_izq = (xintermedio - 0.25);
                  xintermedio_der = (xintermedio + 0.25);
                  //pendiente prima
                  mprima = (-1/m);
                  ynuevo_izq = (mprima* (xintermedio_izq - xintermedio)) +
                               yintermedio;
                  ynuevo_der = (mprima* (xintermedio_der - xintermedio)) +
                               yintermedio;
            }//if m < 1.0
            else
            {
                  if ( m < 3.75 ) // 315-345 grados
                  {
                        // punto de interseccion de las rectas
                        xintermedio = xfinal -(incx/EscalaVector);
                        yintermedio = (m*(xintermedio-x))+y;
```

```
                                                // puntos xintermedios
                                                xintermedio_izq = (xintermedio - 0.5);
                                                xintermedio_der = (xintermedio + 0.5);
                                                //pendiente prima
                                                mprima = (-1/m);
                                                ynuevo_izq = (mprima* (xintermedio_izq - xintermedio)) +
                                                            yintermedio;
                                                ynuevo_der = (mprima* (xintermedio_der - xintermedio)) +
                                                yintermedio;
                                          }//if m < 3.75
                                          else // 270-285
                                          {
                                                xintermedio = xfinal-(incx/EscalaVector);
                                                yintermedio = yfinal-(incy/EscalaVector);
                                                xintermedio_izq = (xintermedio - 1.0);
                                                xintermedio_der = (xintermedio + 1.0);

                                                if ((incx/EscalaVector)>0.5) ynuevo_der=yintermedio-0.5;
                                                else ynuevo_der= yintermedio;
                                                ynuevo_izq = yintermedio ;
                                          }//else 270-285
                                    }
                              }
                        }
                        else // incx > 0  ; incy = 0
                        {
                              xintermedio = xfinal - 1.0;
                              yintermedio = yfinal;
                              xintermedio_izq = xintermedio;
                              xintermedio_der = xintermedio;
                              ynuevo_izq = yintermedio - 1.0;
                              ynuevo_der = yintermedio + 1.0;
                        }
                  }//else incy <0
            }//if incx >0
            else
            {
                  if ( incx < 0.0 )
                  {
                        if ( incy < 0.0 )
```

```
                                    {
/*********************************** cuadrante arriba -izquierda ************************************/
                                        if ( m > 3.75 ) // 90-105
                                        {
                                              xintermedio = xfinal-(incx/EscalaVector);
                                              yintermedio = yfinal-(incy/EscalaVector);
                                              xintermedio_izq = (xintermedio - 1.0);
                                              xintermedio_der = (xintermedio + 1.0);

                                              if ((incx/EscalaVector) < -0.5 ) ynuevo_der = yintermedio - 0.5;
                                              else ynuevo_der= yintermedio;
                                              ynuevo_izq = yintermedio ;
                                        } // if m > 3.75
                                        else
                                        {
                                              if( m > 1.0 )//105-135 grados
                                              {
                                                    // punto de interseccion de las rectas
                                                    xintermedio = xfinal-(incx/EscalaVector);
                                                    yintermedio = (m*(xintermedio-x))+y;

                                                    // puntos xintermedios
                                                    xintermedio_izq = (xintermedio - 0.5);
                                                    xintermedio_der = (xintermedio + 0.5);
                                                    //pendiente prima
                                                    mprima = (-1/m);
                                                    ynuevo_izq = (mprima* (xintermedio_izq - xintermedio)) +
                                                                 yintermedio;
                                                    ynuevo_der = (mprima* (xintermedio_der - xintermedio)) +
                                                                 yintermedio;
                                              }//if m > 1.0
                                              else
                                              {
                                                    if ( m > 0.2 ) // 135-165
                                                    {
                                                          // punto de interseccion de las rectas
                                                          xintermedio = xfinal-(incx/EscalaVector);
                                                          yintermedio = (m*(xintermedio-x))+y;

                                                          // puntos xintermedios
```

```
                                                xintermedio_izq = (xintermedio - 0.25);
                                                xintermedio_der = (xintermedio + 0.25);
                                                //pendiente prima
                                                mprima = (-1/m);
                                                ynuevo_izq = (mprima* (xintermedio_izq - xintermedio)) +
                                                            yintermedio;
                                                ynuevo_der = (mprima* (xintermedio_der - xintermedio)) +
                                                            yintermedio;
                                          }//if m > 0.2
                                          else // 165-180
                                          {
                                                xintermedio = xfinal-(incx/EscalaVector);
                                                yintermedio = yfinal-(incy/EscalaVector);
                                                ynuevo_izq = yintermedio - 1.0;
                                                ynuevo_der = yintermedio + 1.0;

                                                //cambio incy por (incy/EscalaVector)
                                                if ((incy/EscalaVector) < -0.5 )
                                                      xintermedio_izq = xintermedio + 0.5;
                                                else xintermedio_izq= xintermedio;
                                                xintermedio_der = xintermedio ;
                                          }//else 165-180
                                    }
                              }
                        }
                        else
                        {
                              if ( incy > 0.0 )
                              {
/************************************ cuadrante abajo-izquierda ************************************/
                                    if ( m  > -0.2 ) // 180-195
                                    {
                                          xintermedio = xfinal-(incx/EscalaVector);
                                          yintermedio = yfinal-(incy/EscalaVector);
                                          ynuevo_izq = yintermedio-1.0;
                                          ynuevo_der = yintermedio+1.0;

                                          if ((incy/EscalaVector) > 0.5 )
                                                xintermedio_der = xintermedio+0.5;
                                          else  xintermedio_der = xintermedio;
```

```
                        xintermedio_izq = xintermedio;
                  }
                  else
                  {
                        if (m > -1.0) //195-225 grados
                        {
                              // punto de interseccion de las rectas
                              xintermedio = xfinal /*+ 1.0;*/-(incx/EscalaVector);
                              yintermedio = (m*(xintermedio-x))+y;

                              // puntos xintermedios
                              xintermedio_izq = (xintermedio - 0.25);
                              xintermedio_der = (xintermedio + 0.25);
                              //pendiente prima
                              mprima = (-1/m);
                              ynuevo_izq = (mprima* (xintermedio_izq - xintermedio)) +
                                           yintermedio;
                              ynuevo_der = (mprima* (xintermedio_der - xintermedio)) +
                                           yintermedio;
                        }
                        else
                        {
                              if ( m > -3.75 ) // 225-255
                              {
                                    // punto de interseccion de las rectas
                                    xintermedio = xfinal-(incx/EscalaVector);
                                    yintermedio = (m*(xintermedio-x))+y;

                                    // puntos xintermedios
                                    xintermedio_izq = (xintermedio - 0.5);
                                    xintermedio_der = (xintermedio + 0.5);
                                    //pendiente prima
                                    mprima = (-1/m);
                                    ynuevo_izq = (mprima* (xintermedio_izq -
                                                 xintermedio)) + yintermedio;
                                    ynuevo_der = (mprima* (xintermedio_der -
                                                 xintermedio)) + yintermedio;
                              }// if m > -3.75
                              else // 255-270
                              {
```

```
                                                xintermedio = xfinal-(incx/EscalaVector);
                                                yintermedio = yfinal-(incy/EscalaVector);
                                                xintermedio_izq = (xintermedio - 0.5);
                                                xintermedio_der = (xintermedio + 0.5);

                                                //cambio incx por (incx/EscalaVector)
                                                if ((incx/EscalaVector) < -0.5 )
                                                      ynuevo_der = yintermedio + 0.5;
                                                else ynuevo_der= yintermedio;
                                                ynuevo_izq = yintermedio ;
                                          }
                                    }
                              }
                        }
                        else // incx < 0 ; incy = 0
                        {
                              xintermedio = xfinal-(incx/EscalaVector);
                              yintermedio = yfinal;
                              xintermedio_izq = xintermedio;
                              xintermedio_der = xintermedio;
                              ynuevo_izq = yintermedio - 1.0;
                              ynuevo_der = yintermedio + 1.0;
                        }
                  }
            }
            else // incx = 0
            {
                  if ( incy > 0.0 )
                  {
                        xintermedio = xfinal;
                        yintermedio = yfinal-(incy/EscalaVector);
                        xintermedio_izq = xintermedio -1.0;
                        xintermedio_der = xintermedio + 1.0;
                        ynuevo_izq = yintermedio;
                        ynuevo_der = yintermedio;
                  }
                  else
                  {
                        xintermedio = xfinal;
                        yintermedio = yfinal-(incy/EscalaVector);
```

```
                                        xintermedio_izq = xintermedio -1.0;
                                        xintermedio_der = xintermedio + 1.0;
                                        ynuevo_izq = yintermedio;
                                        ynuevo_der = yintermedio;
                                  }
                            }
                      }
                      shape->setLine(lado, 0, 0, 0);
                      shape->drawLineTo(xintermedio_izq,ynuevo_izq);
                      shape->drawLineTo(xintermedio_der,ynuevo_der);
                      shape->drawLineTo(xfinal,yfinal);
                      shape->movePenTo(x,y);
                }
          }// ( si es un buenOF )
          else // no es un buenOF
          {
                // Se pinta de rojo para mostrar que es un error
                if ((_vx[c][r]==100) && (_vy[c][r]==100))
                {
                      shape->setLine(1, 255, 0, 0);
                      shape->drawLine(lado,0); shape->drawLine(0,lado);
                      shape->drawLine(-lado,0);shape->drawLine(0,-lado);
                }
          }
          shape->movePen(0,(double) (Espaciado*EscalaImagen)); // se cambia de fila
          y+= (double) Espaciado*EscalaImagen;
     }//for r
     x +=(double ) (Espaciado*EscalaImagen);
     y =((ceil((double)_nrows/(double)Espaciado))*((double)(Espaciado*EscalaImagen)))-(y-(Espaciado*EscalaImagen));
     shape->movePenTo(x,y); // se pone en el nuevo punto en el que dibujar
   }//for c
   return shape;
}
```

APÉNDICE VI

HazAnimacion.cpp

```
#include "../../mingpp.h"
#include <stdlib.h>
#include <strstream>
#include <iomanip>
#include <iostream>
#include <fstream>
#include <stdio.h>

// HazAnimacion NombreImagen cuantos fichero.swf fichero.html
// La imagen debe tener como nombre NombreImagen(.xxxxx.jpg) comenzando
desde 00000

using namespace std;

int main(int argc,char *argv[])
{
  int columnas,filas;
  int cuantos = atoi(argv[2]);

  const char *jpgname = argv[1];

  Ming_init();

  SWFMovie *movie=new SWFMovie();

  movie->setBackground(255,255,255);
  movie->setRate(5.0);

  SWFShape *shape, *shape_ant,*shapejpg,*shapejpgant;
  SWFDisplayItem *anterior,*actual,*jpg,*jpgant;

  SWFBitmap *b;

  shape_ant = new SWFShape();
  shapejpgant = new SWFShape();

  anterior = movie->add(shape_ant);  // anterior shape
  jpgant = movie->add(shapejpgant);  // anterior jpg mostrado

  for (int i=0; i<cuantos; i++)
  {
      SWFShape *shapeaux = new SWFShape();
      shape_ant = new SWFShape();
      shapejpg = new SWFShape();
      shapejpgant = new SWFShape();

      // borra el shape anterior
      movie->remove(anterior);
      // borra el anterior jpg
      movie->remove(jpgant);

      // preparamos nombre del fichero jpg
      ostrstream ficherojpg;
      int longitud;
      const string nombre = (string) jpgname;
      ficherojpg << nombre.substr(0,longitud-9) << "."<< setw(5) <<
                    setfill('0') << i << ".jpg" << ends;

      b = new SWFBitmap(ficherojpg.str());
```

```
      // define las dimensiones de la pelicula en x,y
      movie->setDimension(b->getWidth(),b->getHeight());

      columnas = (int) b->getWidth();
      filas = (int) b->getHeight();

      //asignar imagen jpg al "shape"

      shapeaux->setRightFill(shapeaux->addBitmapFill(b));
      shapeaux->setLine(1,0,0,255);

      //recuadro que rodeara a la imagen

      shapeaux->drawLine(b->getWidth(),0);
      shapeaux->drawLine(0,b->getHeight());
      shapeaux->drawLine(-b->getWidth(),0);
      shapeaux->drawLine(0,-b->getHeight());

      shapejpg = shapeaux;
      shape = shapejpg;
      //ponemos la nueva imagen jpg en la pelicula
      jpg = movie->add(shapejpg);
      //muestra el nuevo shape con el jpg puesto
      actual = movie->add(shape);

      movie->nextFrame();
      movie->nextFrame();
      movie->nextFrame();
      movie->nextFrame();
      movie->nextFrame();

      shape_ant = shape;
      shapejpgant = shapejpg;
      anterior = actual;
      jpgant = jpg;
      }
  movie->save(argv[3]);

  // Se genera el fichero html para mostrar la pelicula
  ofstream salida (argv[4]);
  if (!salida) {
     cerr << "Fallo en apertura de fichero" << argv[4] << "\n";
     exit(1);
  }
  else
  {
      salida << "<html>" << endl;
      salida << "   <head>" << endl;
      salida << "   </head>" << endl;
      salida << "   <body>" << endl;
      salida << "      <center>"<< endl;
      salida << "<OBJECT classid=\"clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000\"" << endl;//
      salida <<
"codebase=\"http://active.macromedia.com/flash2/cabs/swflash.cab#version=
4,0,0,0\""<<endl;//
      salida << "ID=objects"<< endl;
      salida << "WIDTH=" << columnas <<endl;
      salida << "HEIGHT=" << filas << endl;
      salida << "<PARAM NAME=movie VALUE=\"" << argv[3] << "\">"<<endl;//
      salida << "<EMBED src=\""<< argv[3] << "\""<<endl;//
      salida << "TYPE=\"application/x-shockwave-flash\""<<endl;//
```

```
      salida <<
"PLUGINSPAGE=\"http://www.macromedia.com/shockwave/download/index.cgi?P1_
Prod_Version=ShockwaveFlash\""             << endl;
      salida << "WIDTH=" << columnas << endl;
      salida << "HEIGHT=" << filas << endl;
      salida << "</object>" << endl;
      salida << "</center>" << endl;
      salida << "</body>" << endl;
      salida << "</html>" << endl;
  }
  salida.close();
}
```

APENDICE VII

rectangulo.cpp

```
#include "../../mingpp.h"

int main()
{
  Ming_init();

 SWFMovie *movie = new SWFMovie();

  movie->setDimension(460,460);
  movie->setBackground(0,0,255);

  SWFShape *shape = new SWFShape();

  shape->setLine(10, 200, 200, 0);
  shape->movePenTo(180,180);
  shape->drawLine(100,0);
  shape->drawLine(0,100);
  shape->drawLine(-100,0);
  shape->drawLine(0,-100);

  movie->add(shape);
  movie->save("./rectangulo.swf");
}
```

APENDICE VIII

rectangulo_animado.cpp

```
#include "../../mingpp.h"

int main()
{
  Ming_init();

  // Define la animacion

  SWFMovie *Movie=new SWFMovie();
  Movie->setDimension(500,500);
  Movie->setBackground(255,0,0);
  Movie->setRate(20.0);

  // Elemento animado
  SWFShape *Shape1=new SWFShape();

  Shape1->setLine(5,0,255,0);

  Shape1->movePen(-50,-50);
  Shape1->drawLine(100,0);
  Shape1->drawLine(0,100);
  Shape1->drawLine(-100,0);
  Shape1->drawLine(0,-100);


  // Objeto para mover el cuadrado
  SWFDisplayItem *MovimientoCuadrado;

  MovimientoCuadrado=Movie->add(Shape1);
  MovimientoCuadrado->moveTo(50,200);

  for (int i=0; i<50; i++) {

      Movie->nextFrame();
      MovimientoCuadrado->move(5,0);
  }

  Movie->save("./rectangulo_animado.swf");
}
```

APENDICE IX

imagen.cpp

```
#include "../../mingpp.h"

int main()
{
  Ming_init();

  SWFDisplayItem  *jpg;
  SWFMovie *movie=new SWFMovie();

  movie->setDimension(500,400);
  movie->setBackground(255,255,255);

  SWFShape *shape = new SWFShape();
  SWFBitmap *b = new SWFBitmap("imagen1.jpg");

  //asignar imagen a "shape"

  shape->setRightFill(shape->addBitmapFill(b));
  shape->setLine(1,0,0,255);
  shape->drawLine(b->getWidth(),0);  //recuadro que rodeara a la imagen
  shape->drawLine(0,b->getHeight());
  shape->drawLine(-b->getWidth(),0);
  shape->drawLine(0,-b->getHeight());

  jpg = movie->add(shape);

  movie->save("./imagen.swf");
}
```

APENDICE X

imagen_animada.cpp

```
#include "../../mingpp.h"

int main()
{
  Ming_init();

  SWFMovie *movie=new SWFMovie();

  movie->setDimension(500,400);
  movie->setBackground(255,255,255);

  movie->setRate(2.0);


      SWFShape *shape, *shape_ant,*shapejpg,*shapejpgant;
      SWFDisplayItem *anterior,*actual,*jpg,*jpgant;

      SWFBitmap *b;

      shape_ant = new SWFShape();
      shapejpgant = new SWFShape();

      anterior = movie->add(shape_ant);  // inicializacion
      jpgant = movie->add(shapejpgant);

      for (int i=0; i<2; i++)
      {
      SWFShape *shapeaux = new SWFShape();
      shape_ant = new SWFShape();
      shapejpg = new SWFShape();
      shapejpgant = new SWFShape();

            movie->remove(anterior);     // borra el shape anterior
            // prepara el nuevo jpg para mostrarlo en shapejpg
            movie->remove(jpgant);

            if (i == 0 )
               b = new SWFBitmap("imagen1.jpg");
            else b = new SWFBitmap("imagen2.jpg");

            //asignar imagen a "shape"

            shapeaux->setRightFill(shapeaux->addBitmapFill(b));
            shapeaux->setLine(1,0,0,255);

            //recuadro que rodeara a la imagen

            shapeaux->drawLine(b->getWidth(),0);
            shapeaux->drawLine(0,b->getHeight());
            shapeaux->drawLine(-b->getWidth(),0);
            shapeaux->drawLine(0,-b->getHeight());

            shapejpg = shapeaux;
            shape = shapejpg;
            jpg = movie->add(shapejpg);


            actual = movie->add(shape);  //muestra el nuevo shape
```

```
            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();
            movie->nextFrame();

            shape_ant = shape;
            shapejpgant = shapejpg
            anterior = actual;
            jpgant = jpg;

      }

  movie->save("./imagen_animada.swf");
}
```