UNIVERSIDAD POLITECNICA DE VALENCIA

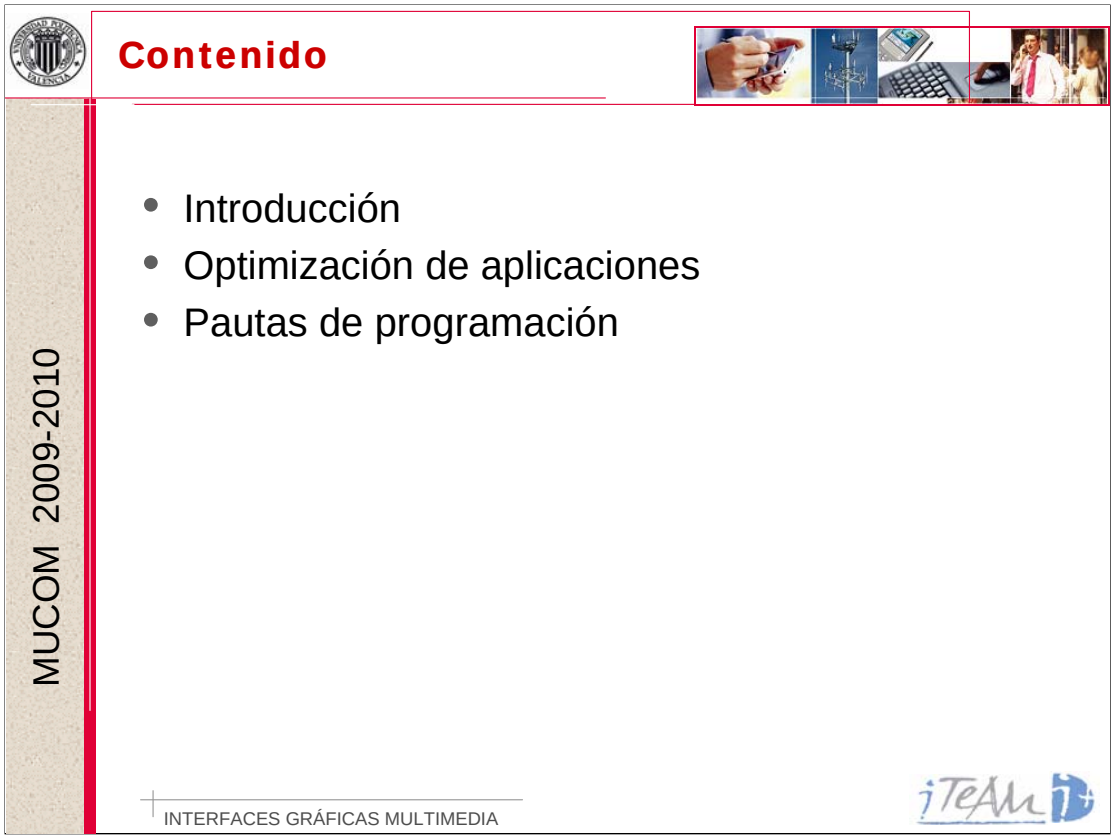
MASTER EN COMUNICACIONES Y DESARROLLO DE SERVICIOS MÓVILES

2009-2010

INTERFACES GRÁFICAS MULTIMEDIA

Optimización de aplicaciones

iTeAM Instituto de Telecomunicaciones y Aplicaciones Multimedia



En este tema se mostrarán las *best practices* para el desarrollo de aplicaciones en dispositivos móviles con Flash Lite y ActionScript. Se presentarán unas pautas de programación y optimización que servirán de ayuda a la hora de mejorar el rendimiento de la aplicación, puesto que en el mundo de los dispositivos móviles cada bit cuenta.

Contenido

- Introducción ►
- Optimización de aplicaciones
- Pautas de programación

MUCOM 2009-2010

Pese a que disponemos del emulador de dispositivos en Flash Lite, realmente no existe mejor sustituto para probar el hardware que no sea el propio dispositivo.

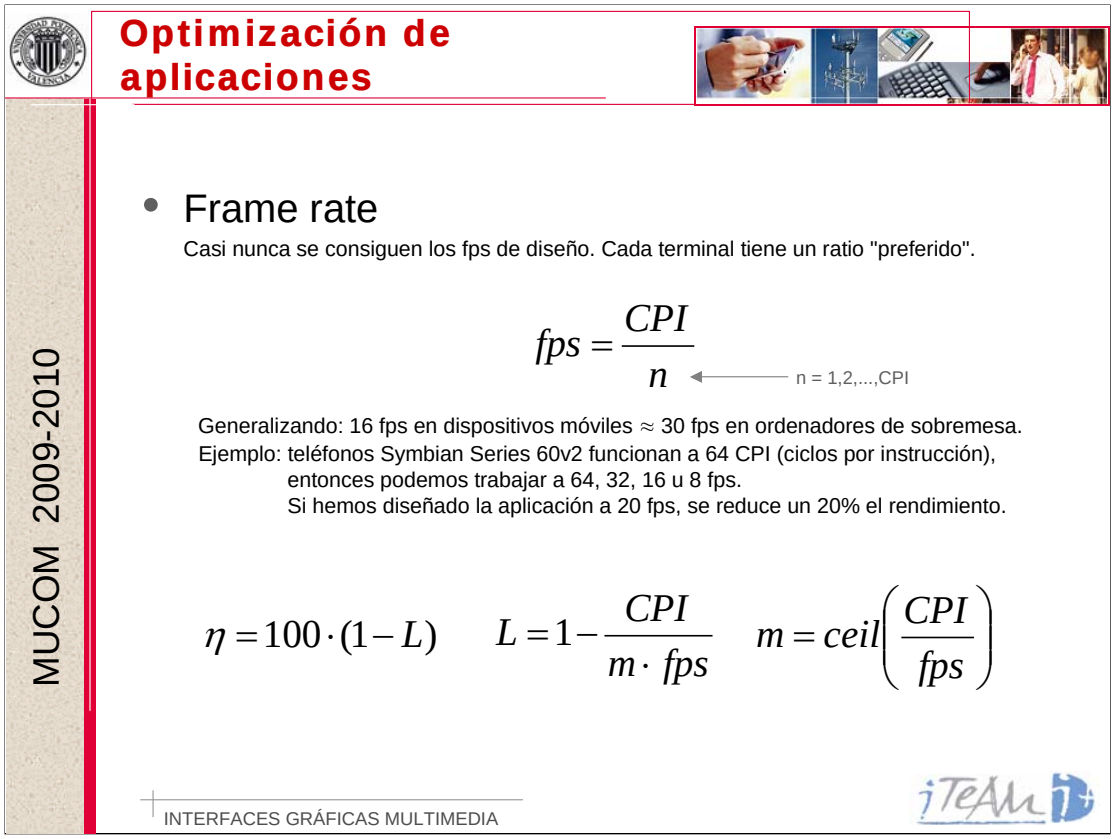
Ya hemos visto que los dispositivos móviles tienen mucha menos memoria y capacidad de procesado que los ordenadores tradicionales de sobremesa y portátiles, aunque poco a poco se van estrechando diferencias.

Muchas características, como la resolución de pantalla o la velocidad de la CPU, varían considerablemente entre los diferentes modelos de dispositivos. Por lo tanto, es importante considerar varios parámetros de optimización y rendimiento en toda aplicación para dispositivos móviles.

Contenido

MUCOM 2009-2010

- Introducción
- Optimización de aplicaciones ►
- Pautas de programación



Frame Rate

El mayor problema que afectará a nuestra aplicación es determinar el *frame rate*, esto es, los fotogramas por segundo (fps) a los que se ejecutará el programa. En muy pocas ocasiones se consiguen los fps de diseño, dado que cada terminal tiene un ratio nativo. Por ejemplo, en los teléfonos de Symbian Series 60v2 el número de instrucciones por ciclo es de 1/64. Esto significa que las funciones que usemos pueden llamarse a intervalos de 1/64, 2/64, 3/64, 4/64, 5/64, y así sucesivamente. De esta forma, si hemos preparado nuestra aplicación para funcionar a 20 fps, teóricamente los eventos deberían ser ejecutados por el dispositivo cada 1/20 segundos. Para los teléfonos de este ejemplo, 1/20 se encuentra entre los intervalos de 3/64 y 4/64, que son los que proporciona dicho terminal. Dado que la ejecución de instrucciones es conservativa, nuestra animación se ejecutará a una velocidad de 4/64 segundos, es decir, a 16 fps. Este “ajuste” del 20% se percibe como una pérdida de rendimiento, haciendo por ejemplo (y exagerando un poco) que una transición que habíamos creado se convierta en algo más que un mero refresco de pantalla.

La tasa de fotogramas por segundo se ve modificada además, por supuesto, según el tipo y cantidad de contenido mostrado en pantalla.

Como regla general, la tasa de 16 fps es la ideal, equivalente a los 30 fps tan usados en las aplicaciones de escritorio. Se recomienda no pasar de 25 fps, pese a que existen dispositivos que soportan incluso mayores ratios pero no son mayoría.

- Tiempo de CPU

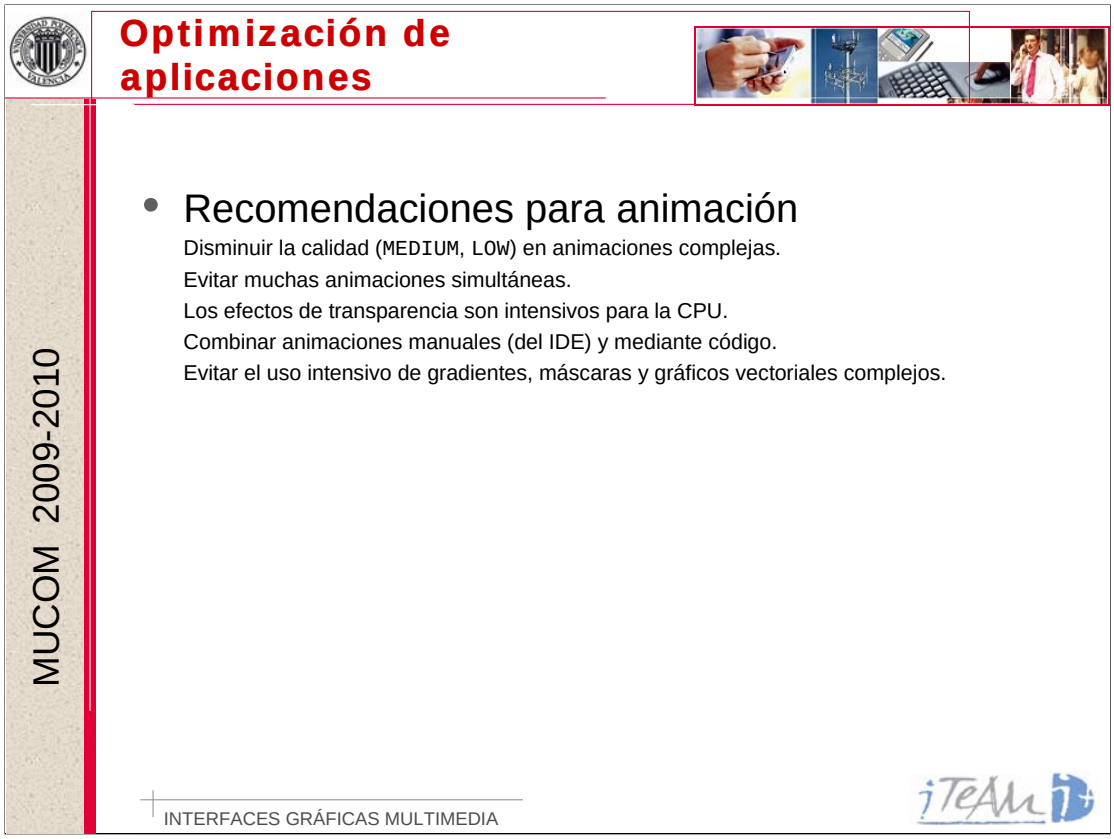
  Parámetro objetivo para comparar el tiempo de procesado de instrucciones máquina.

$$t = I \cdot CPI \cdot T \qquad T = \frac{1}{f}$$

COMPARATIVA · RISC 32 bit	Teléfono	PDA	Ordenador
Procesador	ARM7	ARM9	AMD K8
CPI (ciclos por instrucción)	64	4	1/9
Frecuencia Reloj	200 MHz	168 MHz	2 GHz
Tiempo en procesar 1 instrucción	320 ns	23 ns	0.05 ns

Tiempo de CPU

Es un parámetro objetivo para comparar los tiempos de procesado de instrucciones máquina (no de código). La inclusión de esta ecuación y comparativa es puramente anecdótica, si bien ilustra los órdenes de magnitud en los que nos movemos al trabajar con dispositivos móviles.



Recomendaciones para animación

Flash Lite puede renderizar los gráficos vectoriales en tres niveles de calidad: baja, media y alta. A mayor calidad, mayor suavizado y precisión y por lo tanto mayor trabajo para el procesador. Es importante poder modificar la calidad mediante ActionScript en aquellas animaciones que sean muy complejas.

Se recomienda no usar muchas animaciones simultáneamente. Siempre podremos jugar con la subdivisión en animaciones más sencillas, o de tal forma que cuando acabe una empiece otra.

Los efectos de transparencia (`_alpha`) son intensivos para la CPU, por lo que es recomendable limitarlos en aquellas escenas donde intervienen muchos símbolos gráficos.

Las animaciones demasiado largas pueden reducir el rendimiento de la aplicación, mostrando a saltos una animación que se veía muy suave en nuestro ordenador, por ejemplo. Para mejorar esto podemos combinar animaciones realizadas en la línea de tiempo con código ActionScript.

Otros efectos visuales que requieren normalmente más potencia de CPU son las máscaras, gradientes y gráficos vectoriales muy complejos.

Optimización de aplicaciones

MUCOM 2009-2010

- **Tamaño del escenario**
  Al tener que escalar todo el contenido, el rendimiento decrece.
- **Variables**
  Usar nombres descriptivos pero cortos.
  Mejor variables locales.
- **Funciones**
  Evitar el abuso de funciones anónimas.
- **Liberación de memoria**
  Los `addListener()` y `setInterval()` dentro de un MovieClip no desaparecen al borrar el movieClip mediante `unloadMovie()` o `removeMovieClip()`.
  Variables y objetos: `delete` o asignarles el valor `null`.

Tamaño de la aplicación

Pese a que las aplicaciones realizadas con Flash Lite pueden escalar su contenido, normalmente existe una pérdida de rendimiento cuando el reproductor tiene que ampliar mucho el tamaño de la aplicación.

Variables

Los nombres de las variables que usemos deben ser descriptivos, sí, pero lo más cortos posibles. Sobre todo si vamos a acceder a ellas muchas veces a lo largo del código.

Es conveniente registrar las variables en modo local en vez de global, pues son eliminadas de la memoria del dispositivo de forma automática cuando ya no se necesitan en la aplicación.

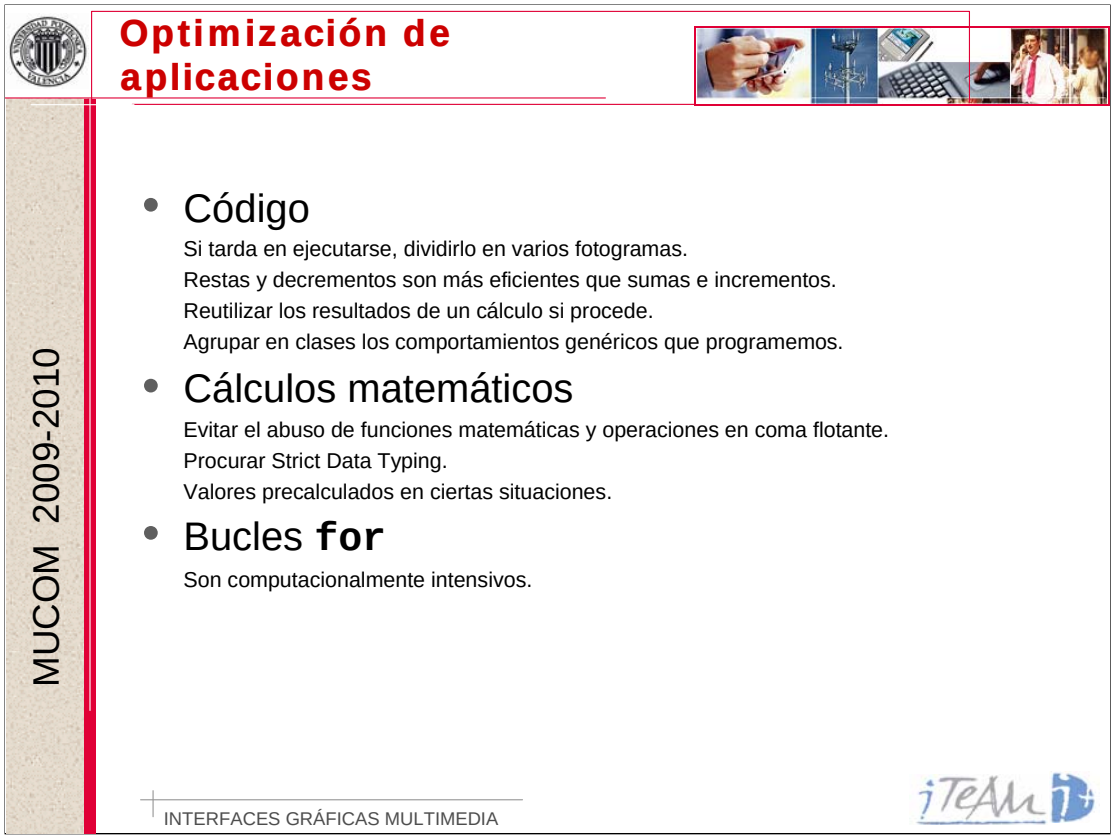
Funciones

Evitar el abuso de funciones anónimas (ej: en callbacks). En su lugar, es preferible emplear listeners o definir primero la función y llamarla después.

Liberación de memoria

Cuando usemos listeners o funciones `setInterval()` en un MovieClip, los datos no se liberarán de la memoria del dispositivo mediante `unloadMovie()` o `removeMovieClip()`. Tendremos que hacerlo en nuestro código mediante un `removeListener()` o mediante la función `clearInterval()`, respectivamente.

Podemos eliminar variables y objetos mediante la sentencia `delete` o asignarles el valor `null`.



Código

Si un código tarda más de 1 segundo en ejecutarse, se recomienda dividirlo en múltiples instantes de ejecución, acordes a la tasa de refresco del dispositivo.

En general, la operación de sustracción (resta) es más rápida computacionalmente que la de adición (suma). Por esta misma razón, los decrementos de variables son mejores que los incrementos.

En ocasiones podemos reutilizar los resultados de un cálculo, si procede, en lugar de volver a recalcularlo todo.

Es conveniente agrupar en clases o librerías las acciones que van a realizar los objetos, teniendo así ordenada de forma sistemática nuestra aplicación.

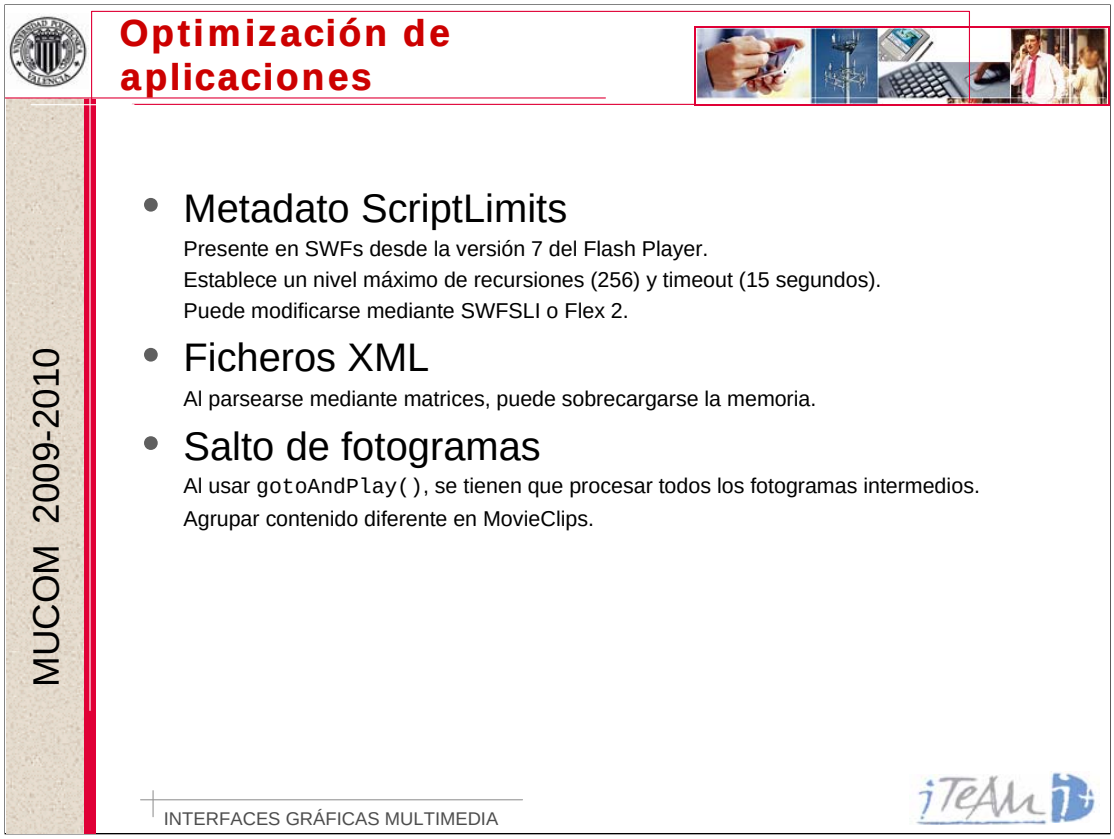
Cálculos matemáticos

El abuso de funciones matemáticas y números en coma flotante disminuyen notablemente el rendimiento de la aplicación. Empleando Strict Data Typing, el reproductor Flash Lite hará las operaciones más eficientemente, ya que el tipo de las variable no cambiará en tiempo de ejecución.

Para ciertas operaciones es mejor tener valores ya precalculados y almacenarlos en una lista o en variables separadas.

Bucles `for`

Los bucles `for` son intensivos para el procesador, ya que la condición de repetición se tiene que calcular en cada iteración. Por eso hay que evitar un número elevado de repeticiones simultáneas, véase aquellas que supongan más de 1 segundo de espera como orden de magnitud en Flash Lite; aunque todo dependerá del procesador y del número de sentencias que haya que ejecutar en cada iteración.



Metadato ScriptLimits

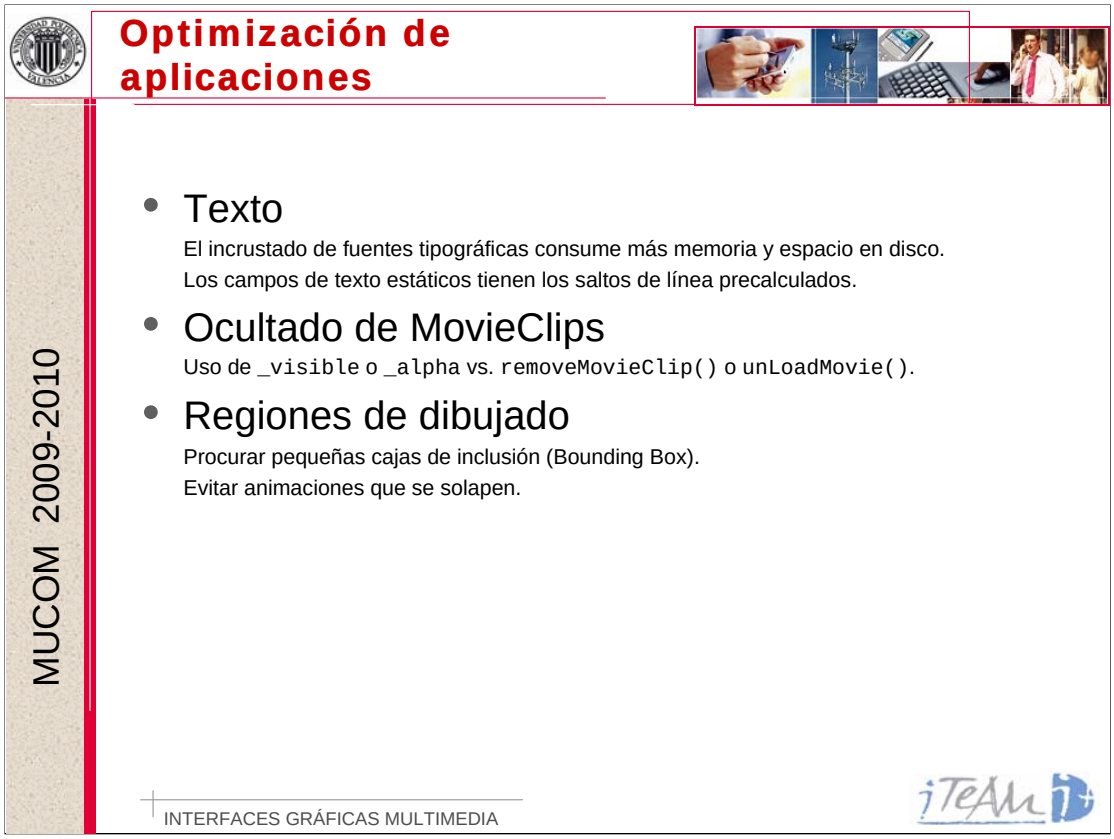
Los ficheros SWF a partir de la versión 7 llevan un metadato llamado ScriptLimits en el que por defecto se establece un nivel máximo de 256 recursiones y un tiempo de espera (*TimeOut*) de 15 segundos. Podemos cambiar estos límites con el programa SWFSLI, de momento sólo disponible para la consola de comandos de Win32. También puede emplearse el compilador Flex2 – por defecto propone 1000 recursiones y 1 minuto de TimeOut.

XML

La carga y parseo de archivos XML puede sobrecargar la memoria del dispositivo, dependiendo claro está del tamaño del XML y del parsing de variables que hagamos. Esto es debido a que al procesar un fichero XML estamos almacenando sus valores en matrices, lo cual no es muy eficiente para la memoria pero de momento es la única forma de hacerlo sin usar APIs externas. En general, la carga de cualquier contenido de gran tamaño puede ralentizar nuestra aplicación.

Salto de fotogramas

Al usar la función `gotoAndPlay()`, todos los frames entre el actual y el frame requerido necesitan ser procesados antes de mostrar el contenido de dicho frame requerido. Si estos frames intermedios tienen un contenido muy diferente, es mejor usar diferentes MovieClips en lugar de la línea de tiempo principal.



Texto

Cuando incrustamos fuentes tipográficas (*embed fonts*) estamos creando un gráfico vectorial para cada carácter, lo cual consume más memoria y hace que la aplicación ocupe más espacio. Por ello, debemos emplear las fuentes del dispositivo en la medida de lo posible, las cuales son tratadas como entidades del sistema operativo.

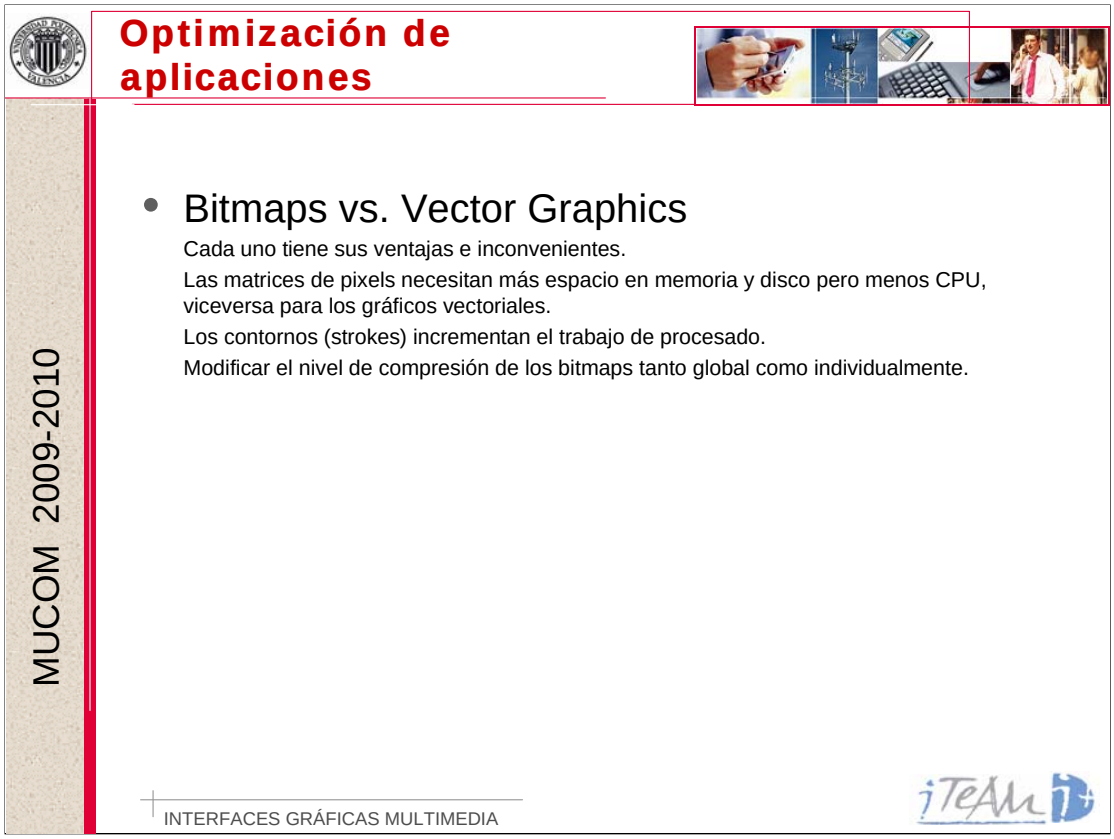
Los campos de texto estáticos no son tan problemáticos porque los saltos de línea están precalculados. Sin embargo, en los dinámicos y de entrada el salto de línea se calcula cada vez que el campo de texto necesita ser redibujado.

Ocultado de movieClips

Si queremos eliminar un MovieClip, debemos usar `removeMovieClip()` o `unloadMovie()`. Si empleamos `_visible = false` o `_alpha = 0`, el MovieClip seguirá en la memoria del dispositivo, así como en el caso de taparlo con otro MovieClip.

Regiones de dibujado

Flash Player calcula las animaciones de los gráficos en base a un rectángulo ficticio llamado *Bounding Box* o caja de inclusión. Cuanto menor sea dicho rectángulo, más eficiencia conseguiremos. Si evitamos que varias animaciones se solapen entre sí, estaremos disminuyendo el área del Bounding Box. Podemos ver dicho área mediante el comando `showRedrawRegions(true)` al abrir nuestra aplicación desde el escritorio – no disponible en el simulador de dispositivos ni en Flash Lite.



Bitmaps vs. Gráficos Vectoriales

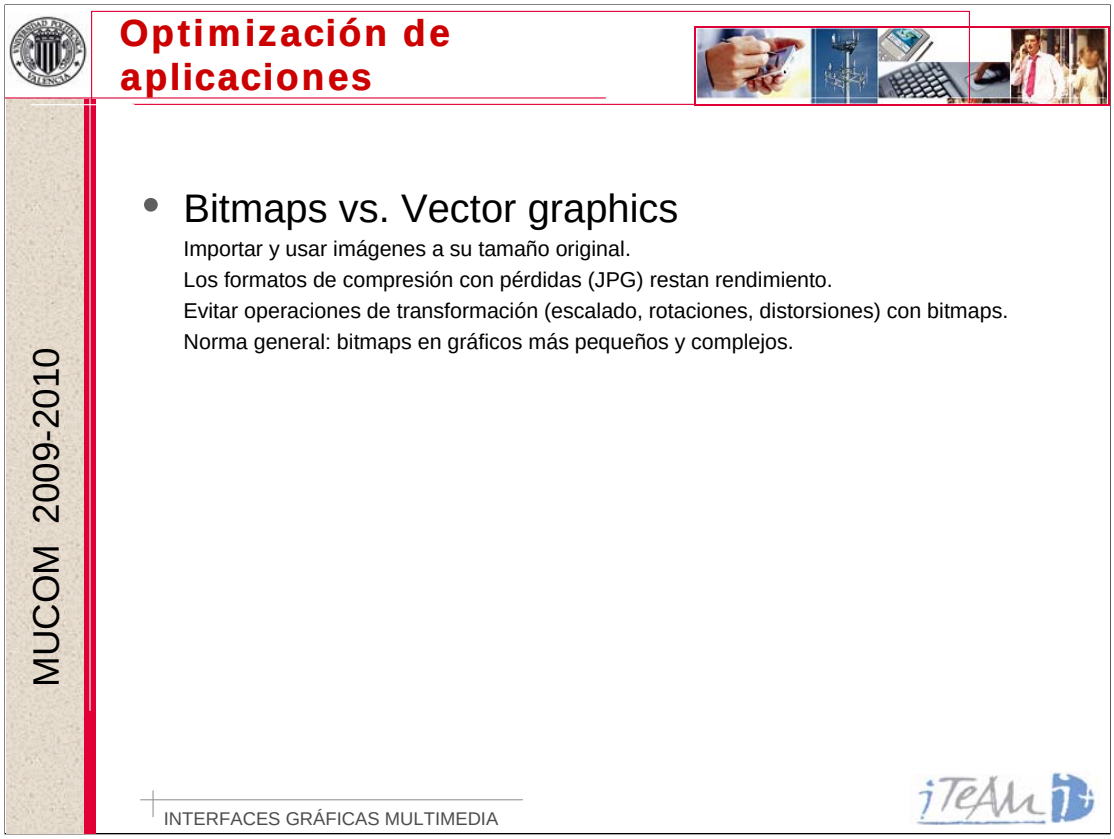
Flash Player puede renderizar gráficos tanto de mapa de bits como vectoriales. Este es un tema bastante controvertido para discutir, ya que cada formato gráfico tiene sus ventajas e inconvenientes. Así, cuándo usar un tipo de gráficos u otro no es una decisión totalmente clara.

Los gráficos vectoriales se almacenan como ecuaciones matemáticas en lugar de matrices de pixels, las cuales necesitan normalmente más bytes de datos. Por eso al usar gráficos vectoriales nos aseguramos de que el fichero SWF ocupará poco espacio en disco y memoria. Al escalar un gráfico vectorial, debido a su naturaleza matemática, su calidad no se ve afectada. Las ecuaciones que lo describen son recalculadas automáticamente para luego mostrarlo en pantalla.

Debido nuevamente a su naturaleza matemática, los gráficos vectoriales necesitan mayor potencia de CPU para ser renderizados. Por ello, a mayor complejidad de trazos, contornos y rellenos, menor rendimiento de la aplicación. Los gráficos de mapa de bits no necesitan apenas potencia de CPU para ser dibujados en pantalla.

A mayor número de contornos (*strokes*) en los dibujos vectoriales, mayor número de elementos a renderizar y por lo tanto mayor uso de procesador. Por ello se recomienda no abusar de su empleo en las aplicaciones.

Podemos modificar el nivel de compresión de los bitmaps tanto individual como globalmente, ajustando así nuestras necesidades de rendimiento-calidad.

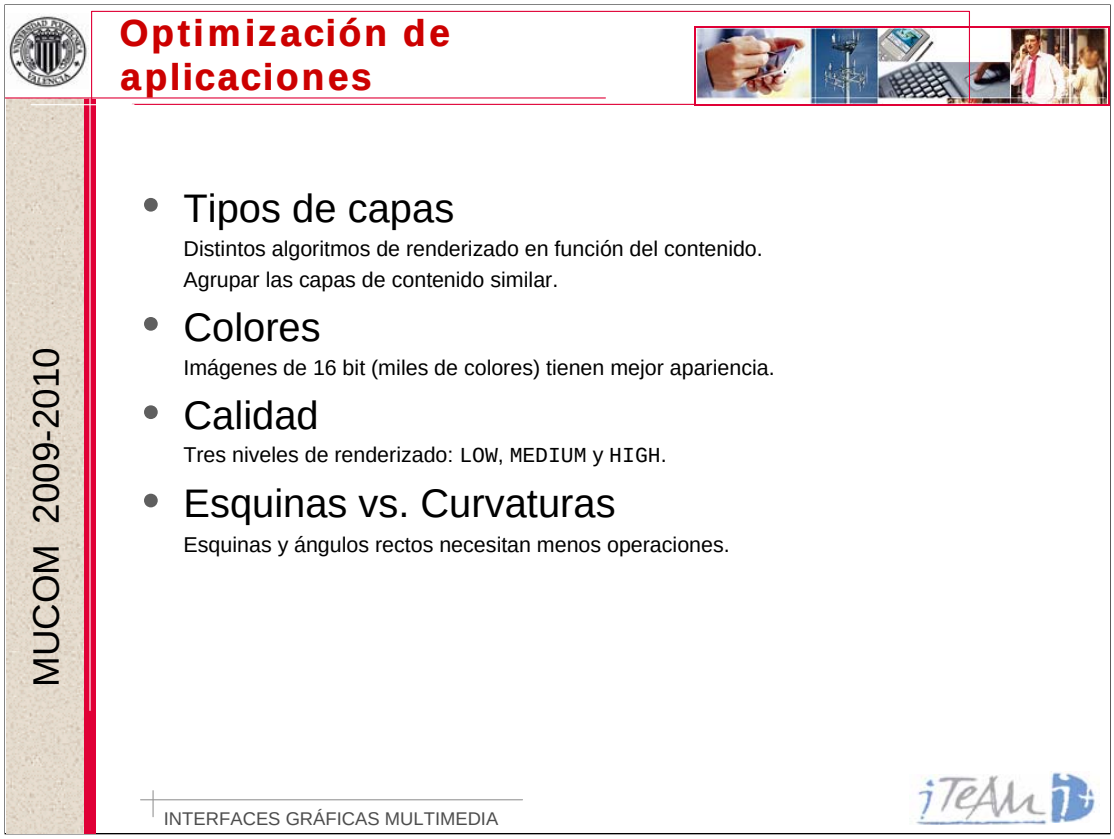


Se aconseja importar las imágenes de mapa de bits al tamaño con el que serán usadas en la aplicación. Si aumentamos su tamaño en el IDE seguramente apreciemos una pérdida de calidad, pero si las importamos a un tamaño superior al de la aplicación y luego reducimos su escala hasta que quepan en pantalla, estaremos desperdiciando la memoria del dispositivo y aumentando el tamaño del fichero SWF.

Los formatos de imagen que usan compresión con pérdidas (por ejemplo JPG) restan rendimiento: se recomienda emplear en su lugar el formato PNG. Si queremos realizar una animación con un bitmap – desplazándolo, escalándolo o rotándolo, por ejemplo – deberíamos intentar usar un gráfico vectorial en su lugar.

Por otra parte, si pensamos incluir en la aplicación muchos gráficos vectoriales estáticos, es mejor usar en su lugar gráficos de mapa de bits. Otra situación en la que es deseable emplear un bitmap en lugar de un gráfico vectorial es en el caso de dibujar pequeños iconos en nuestra aplicación.

Como norma general, el uso de bitmaps es preferible para gráficos pequeños y complejos, y el uso de gráficos vectoriales para aquellos más grandes y simples.



Tipos de capas

Flash Player utiliza diferentes algoritmos de renderizado dependiendo del tipo de gráficos, entre los que va alternando durante el tiempo de ejecución. Se recomienda colocar capas de contenido similar cerca unas de otras, para que esta alternancia entre motores de rendering sea la menor posible.

Colores

La gran mayoría de los teléfonos móviles actuales no soportan profundidades de color de más de 16 bit (65536 colores). Por lo tanto es recomendable diseñar nuestra aplicación como mucho a esta profundidad de color.

Calidad

El motor de render Flash Player puede emplear tres niveles de calidad diferente para presentar los objetos (`LOW`, `MEDIUM` y `HIGH` quality). Dependiendo de la situación, podemos ir alternando entre uno y otro nivel, mediante la propiedad global `_quality`. Los MovieClips aceptan además el nivel de calidad `BEST`, pero, al tratarse de una propiedad global, afecta a toda la aplicación.

Esquinas vs. curvaturas

Los gráficos vectoriales realizados con líneas y ángulos rectos consumen menos memoria y se renderizan más rápido, debido al menor número de operaciones necesarias para representarlos en la memoria. Por ello, y sobre todo en formas pequeñas, se recomienda emplear líneas rectas y ángulos en lugar de curvas.

Optimización de aplicaciones

MUCOM 2009-2010

- **Gradientes**
  Costosos de representar en gráficos vectoriales. Considerar rasterizarlos.
  Gradientes lineales mejor que radiales.
- **Otros recursos gráficos intensivos**
  Transparencias, máscaras y contornos.
- **Precargas**
  Almacenan en un buffer el contenido antes de mostrarlo en pantalla.
- **Ficheros fuente**
  Claridad y limpieza.
  Procurar el menor tamaño posible: cada bit (de la aplicación SWF) cuenta.

Gradientes

Los gradientes en gráficos vectoriales son computacionalmente costosos de calcular. Si se puede, es mejor rasterizar la imagen e importarla a Flash. Los gradientes lineales se renderizan más rápido que los radiales.

Otros recursos gráficos intensivos

Evitar el abuso de transparencias y máscaras; son operaciones que hacen trabajar mucho al procesador. Las líneas de contorno (*strokes*) incrementan notablemente el número de vectores a renderizar.

Precargas

En las aplicaciones de escritorio a veces se carga todo el contenido de la aplicación en el primer fotograma y después se va usando según sea necesario. En muchas aplicaciones para dispositivos móviles no tiene sentido; lo habitual es ir inicializando los objetos a medida que se van necesitando. Recordemos que existen los preloaders, y que están para eso.

Ficheros fuente

Hay que tratar de mantener el código lo más limpio y claro posible, con vistas a una mejor comprensión (sobre todo pasado cierto tiempo), para trabajos en grupo, etc. Lo mismo se aplica a los ficheros FLA y las clases que hayamos creado. Debemos procurar aplicaciones de tamaño tan pequeño como sea posible, eliminando todo código y frames redundantes así como símbolos sin usar. De esta forma, al compilar al formato de publicación SWF, nuestra aplicación sólo contendrá los datos que necesita realmente.

- Gestión de memoria

  Al cargar datos externos muy pesados puede saturarse la memoria.

  Recolección de basura cada minuto o cada sobrecarga mayor o igual al 20%.

  Detallado en http://www.adobe.com/devnet/devices/articles/memory_management.html

Gestión de memoria

Al cargar datos externos de gran tamaño – como una foto en alta resolución o un XML de muchas líneas – puede resultar en un fallo de memoria, aunque el dispositivo tuviera espacio suficiente.

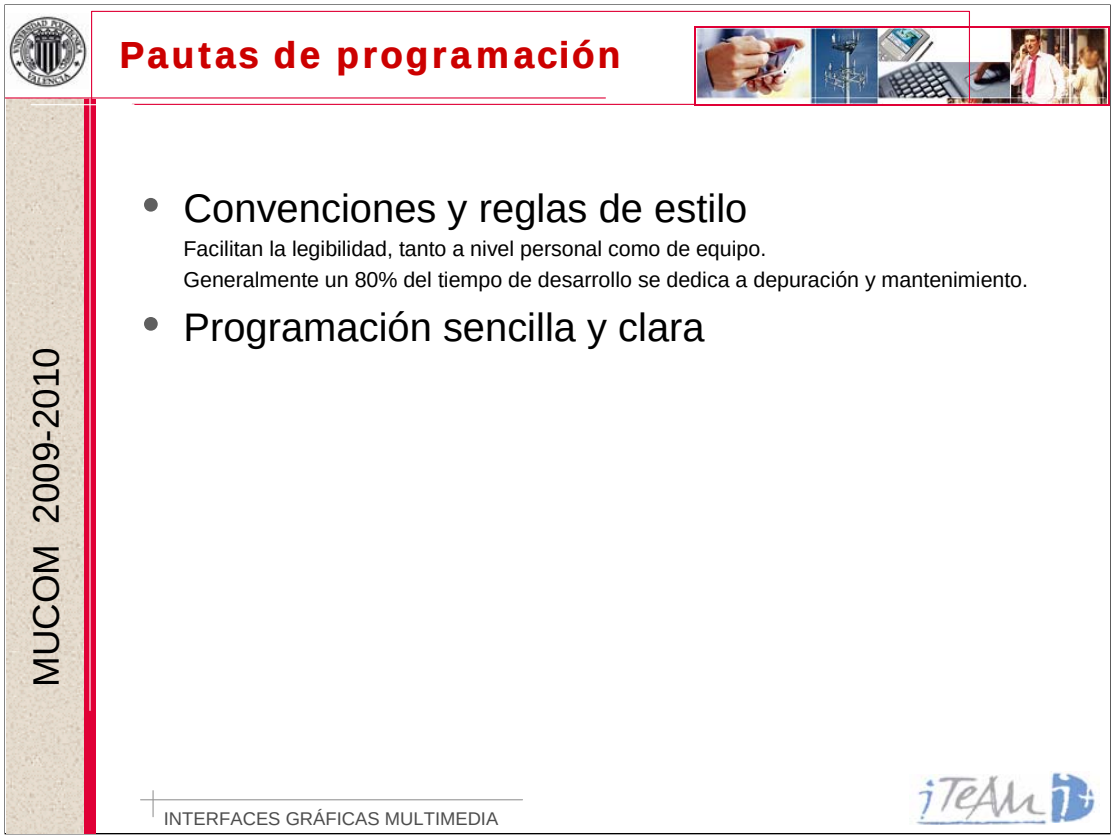
Mediante la recolección de basura, Flash Lite va eliminando de la memoria del dispositivo las variables y objetos no referenciados. Este proceso se ejecuta de forma transparente para el usuario cada 60 segundos o cada vez que la memoria disponible disminuye repentinamente un 20% o más. Dicho proceso de liberación de memoria ocurre en 2 fases: la de marcado (*mark*) y la de barrido (*sweep*). En la primera el recolector de basura marca los objetos no referenciados, y en la segunda los elimina definitivamente.

Para más información: http://www.adobe.com/devnet/devices/articles/memory_management.html.

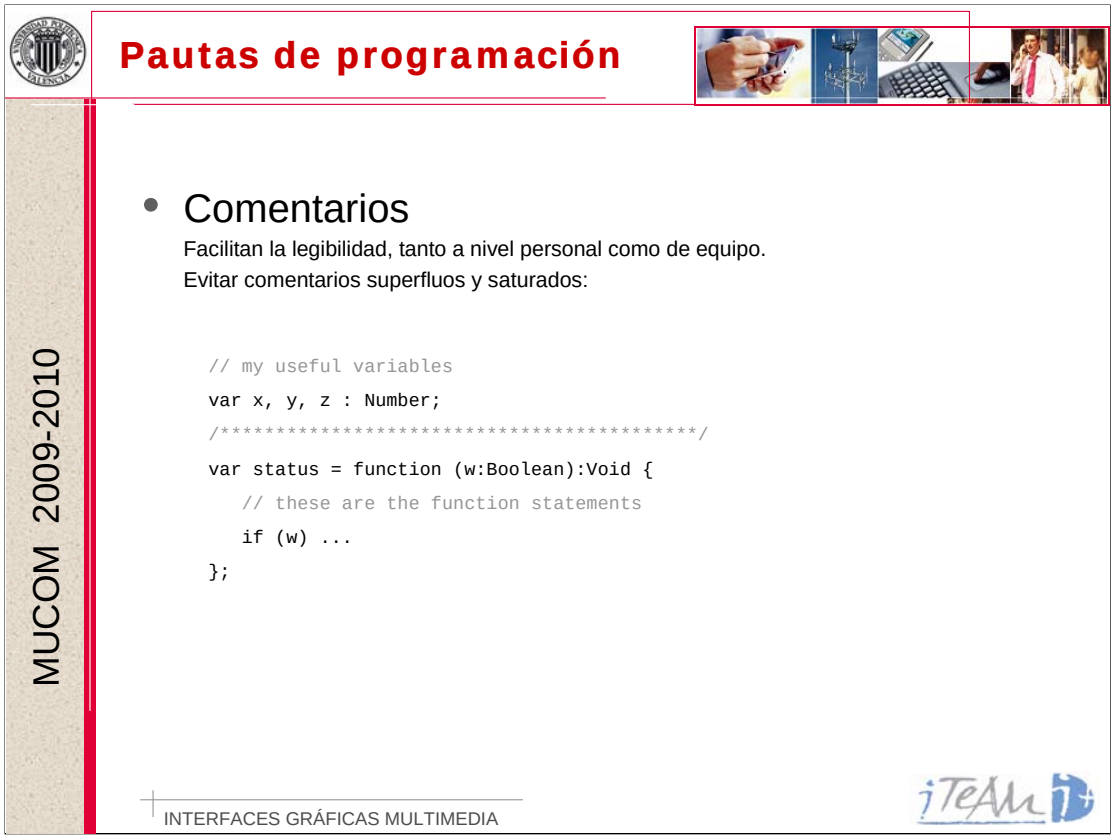
Contenido

- Introducción
- Optimización de aplicaciones
- Pautas de programación ►

MUCOM 2009-2010



Es recomendable seguir ciertas reglas a la hora de escribir nuestro código, con vistas a facilitar la búsqueda de errores o en el caso de que nuestro trabajo deba ser continuado por otros. Los acuerdos en la codificación de ActionScript son muy importantes tanto para desarrolladores como para los diseñadores. Generalmente se emplea un 80% del tiempo de desarrollo de una aplicación depurando, buscando errores, y realizando mantenimiento general, especialmente en los grandes proyectos. A continuación se recopilan las reglas de estilo más empleadas en el lenguaje ActionScript, extraídas en gran parte de la documentación oficial de ActionScript y Flash Lite.



Comentarios

Hay que usar comentarios para nuestro código mientras escribimos. Esto nos ayudará a ordenar y encontrar rápida y efectivamente los elementos a la hora de retomar un proyecto.

Sin embargo, debemos evitar comentarios superfluos del código, como "definición de las variables x e y" u otros comentarios que resulten obvios de manera inmediata para otros desarrolladores. Si tenemos que incluir muchos comentarios sobre el funcionamiento del código, ello será indicativo de que nuestro ActionScript no es ni elegante ni intuitivo.

Se recomienda evitar los comentarios saturados. Un ejemplo de comentario saturado sería el uso de varias líneas de signos igual (=) o asteriscos (*) para crear un bloque de separación alrededor de los comentarios. En lugar de eso, se recomienda usar una línea en blanco para separar los comentarios del código ActionScript.

Pautas de programación

MUCOM 2009-2010

- Nombres e identificadores

Cada objeto debe tener un nombre único.
Empezar variables y funciones con minúsculas, y clases con mayúsculas.
ActionScript es case-sensitive (AS1 no).
Sufijos para los elementos más usados:

Clase	Sufijo	Clase	Sufijo
Array	_array	SharedObject	_so
Button	_btn	Sound	_sound
Color	_color	String	_str
Date	_date	TextField	_txt
LoadVars	_lv	TextFormat	_fmt
MovieClip	_mc	Video	_video
MovieClipLoader	_mcl	XML	_xml

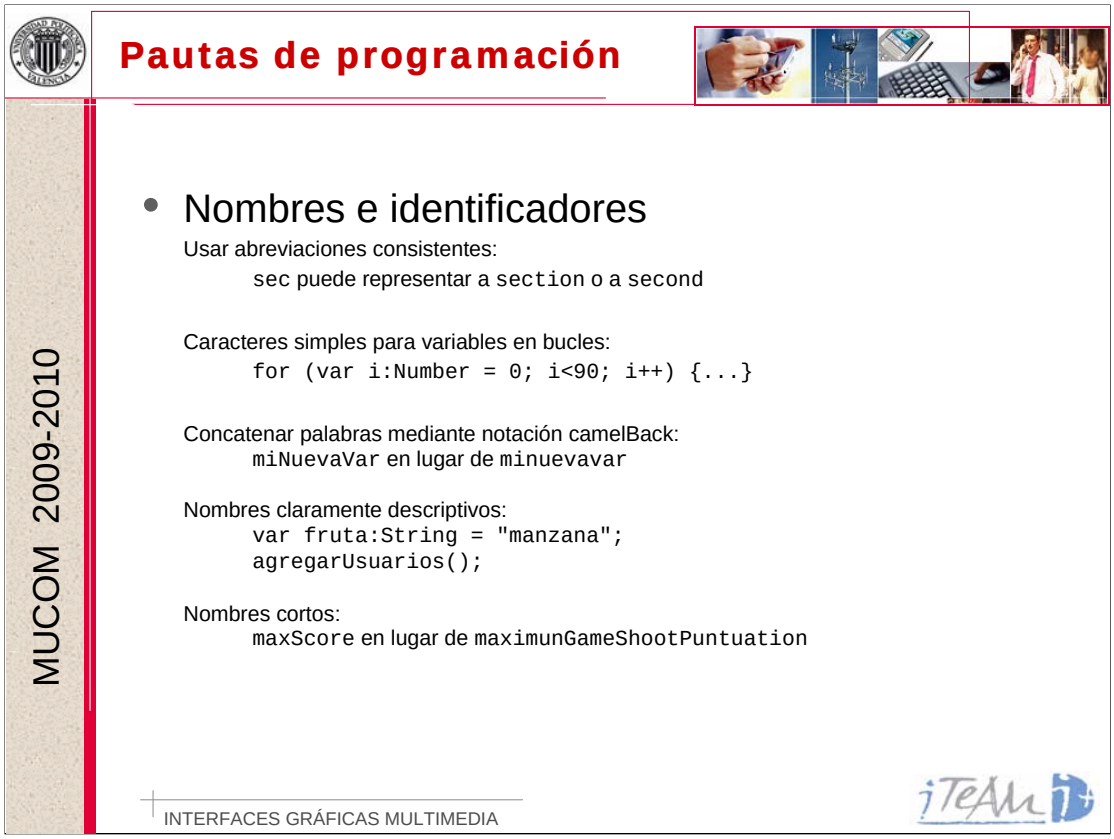
Nombres e identificadores

Todas las variables y funciones deben tener un nombre único. Se recomienda comenzar las variables con una letra minúscula y las clases con una mayúscula.

No usar el mismo nombre con distinta capitalización ya que ActionScript es case-sensitive y podríamos confundirnos – AS1 no lo es pero es buena práctica tratarlo como si lo fuera.

Emplear los sufijos correspondientes para cada uno de los elementos que añadamos a nuestro proyecto. El uso de sufijos nos permitirá trabajar de una manera muy rápida, accediendo directamente a las propiedades de las clases a medida que vamos escribiendo el código. Los elementos más usados tienen su sufijo:

Array (`_array`)
Button (`_btn`)
Color (`_color`)
Date (`_date`)
LoadVars (`_lv`)
MovieClip (`_mc`)
MovieClipLoader (`_mcl`)
SharedObject (`_so`)
Sound (`_sound`)
String (`_str`)
TextField (`_txt`)
TextFormat (`_fmt`)
Video (`_video`)
XML (`_xml`)



Nombres e identificadores

Utilizar abreviaciones consistentes. Una abreviación debe representar claramente una sola cosa.

`sec` puede representar a `section` o a `second`

Utilizar caracteres simples para representar variables en bucles. Esto ayuda a optimizar el rendimiento y la velocidad de procesado.

```
for (var i:Number = 0; i < 90; ++i) {...}
```

Concatenar palabras para crear nombres. En caso de nombres largos o compuestos, alternar capitalización de letras para facilitar el fin y el comienzo de palabras. Esta notación se denomina camelBack.

`miNuevaVar` en lugar de `minuevavar`

Es recomendable utilizar nombres que identifiquen claramente a las variables y funciones, porque de esa manera con sólo leerla en pantalla sabremos qué rol cumple en nuestro proyecto o qué datos almacena.

```
var fruta:String = "manzana";
actualizarUsuarios();
```

Los nombres deben cortos pero consistentes y descriptivos, a menos que se considere un factor de riesgo de ilegibilidad.

`maxScore` en lugar de `maximunGameShootPuntuation`

- Nombres e identificadores

Definición de variables de forma grupal:

```
var minHeight:Number = 100;
var maxHeight:Number = 300;
```

Las constantes se suelen escribir en mayúsculas:

```
var BASE_URL:String = "http://www.miservidor.com";
```

Nombres de paths compactos:

```
this.loadMovie("../src/imgs/2006/12/08/DSC00675.jpg");
```


Nombres e identificadores

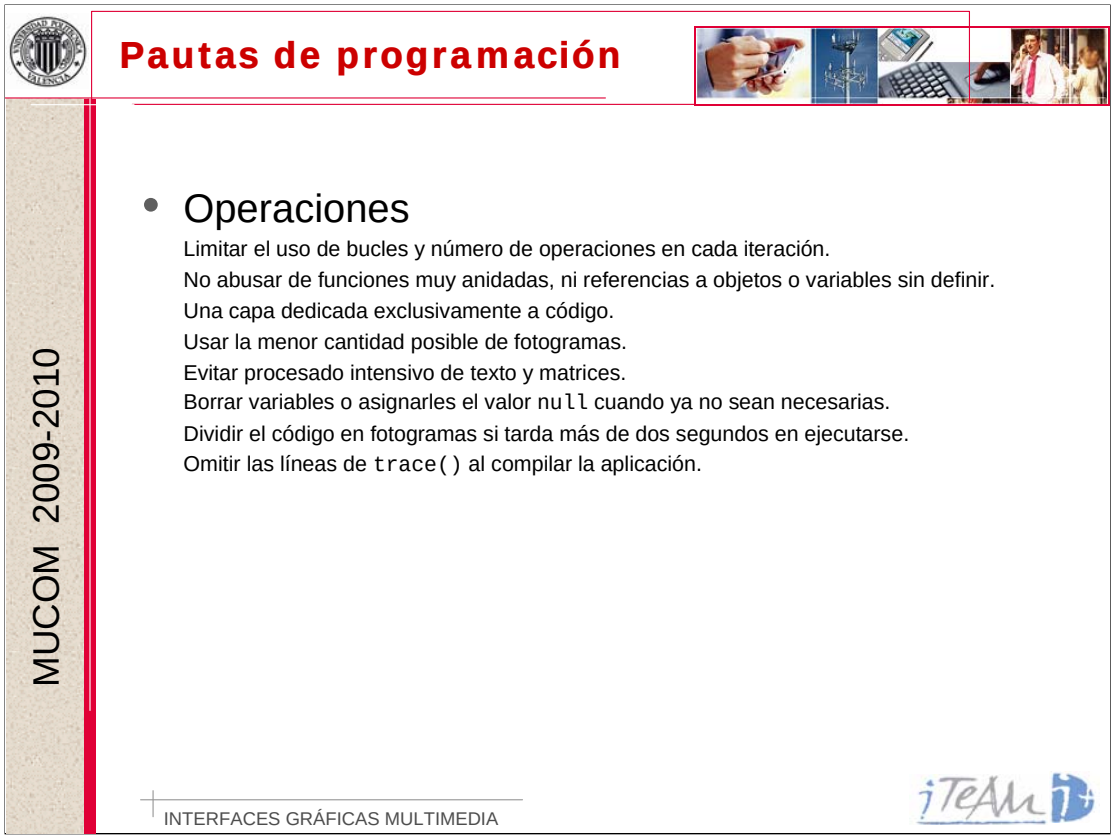
Usar la definición de variables de manera grupal cuando se refieren a un mismo proceso.

```
var minHeight:Number = 100;
var maxHeight:Number = 300;
```

A pesar de que se recomienda alternar la capitalización de letras en los nombres de variables, en las constantes existe una convención de utilizar todas las letras en mayúscula. En caso de ser palabras compuestas se usa el guión bajo:

```
var BASE_URL:String = "http://www.miservidor.com";
```

El nombre de los paths debe ser lo más compacto posible. Al instanciar paquetes de clases, por ejemplo, se crea internamente un nuevo objeto por cada path y una sentencia condicional `if`. Algunos desarrolladores utilizan software de preprocesado para asignar identificadores únicos, de tal forma que la sentencia `com.nombre.xyz.namespace.package.functionName()` queda `ab123cd4.functionName()` antes de compilar el SWF.



Operaciones

Limitar el número de bucles y el número de operaciones en cada bucle. En caso de realizar bucles mediante la línea de tiempo, detenerlos en cuanto no se vayan a utilizar más.

No abusar de funciones muy anidadas ni referencias a objetos o variables sin definir.

Emplear exclusivamente una capa de nuestra línea de tiempo para escribir código, tratando, siempre en lo posible, de agrupar en ella todas las funciones y variables o cualquier bloque de instrucciones importante.

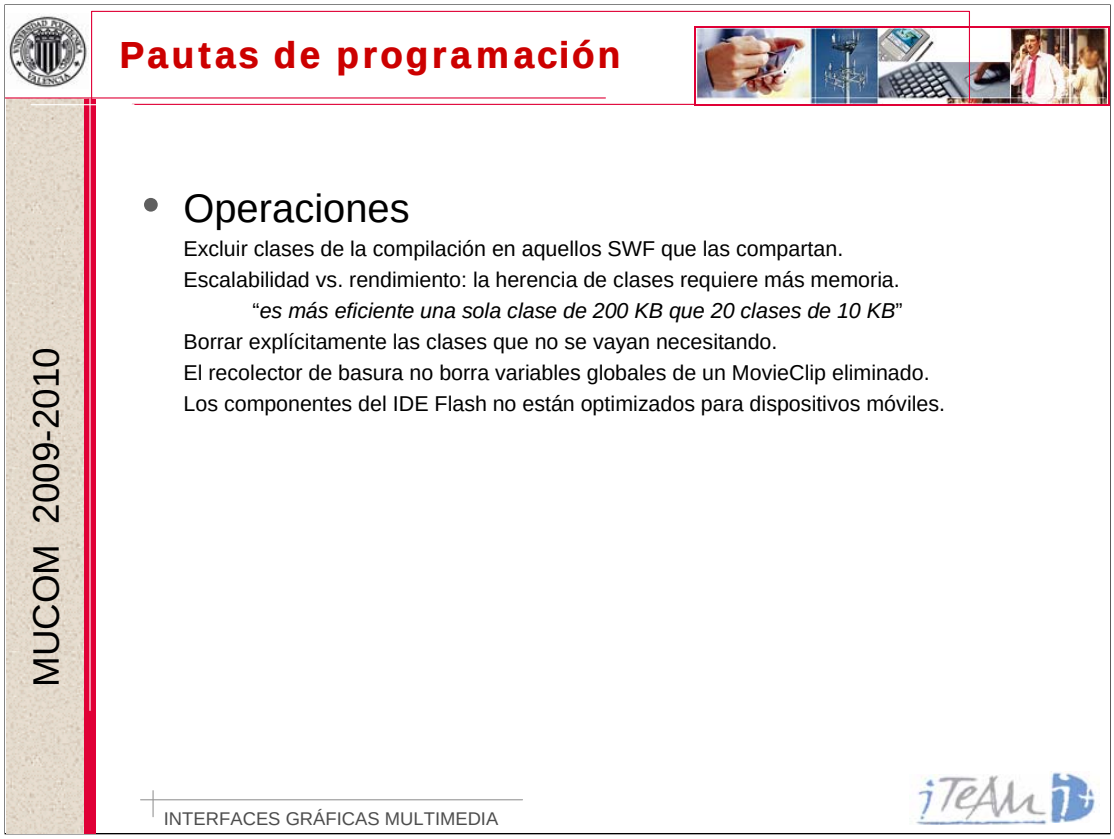
Utilizando la menor cantidad de fotogramas ayudará a mejorar nuestro código, exprimiéndolo al máximo y reduciendo notablemente el peso de los archivos finales.

El procesado intensivo de texto y matrices disminuye notablemente el rendimiento de la aplicación.

Borrar variables o asignarles el valor `null` cuando ya no sean necesarias. Así, el recolector de basura tendrá menos operaciones que realizar. Los listeners se eliminan directamente con la función `removeListener()`. Los intervalos de ejecución, mediante `clearInterval()`.

Si un código tarda más de un segundo en ejecutarse, es recomendable dividirlo en varios fotogramas o emplear signos visuales de qué está sucediendo en la aplicación.

Para ahorrar espacio, se aconseja omitir las líneas de `trace()`, a la hora de publicar el SWF.



Operaciones

Si nuestra aplicación está formada por muchos ficheros SWF que comparten las mismas clases, podemos excluirlas de la compilación de los SWF que especifiquemos.

La herencia de clases incrementa el número de llamadas a métodos y requiere más memoria. Una clase que incluye toda la funcionalidad que necesita ocupa más líneas de código pero es más eficiente que otra de pocas líneas que hereda de varias superclases. Esto supone un dilema entre escalabilidad y rendimiento.

Una vez que se llama a una clase personalizada desde un SWF, ésta queda en memoria incluso si se borra el SWF desde el cual se hizo la llamada. Hasta que no se borre explícitamente mediante la orden `delete` estará consumiendo recursos innecesariamente.

Si en un MovieClip se definen variables globales y luego este es borrado de la memoria, el recolector de basura no podrá borrar dichas variables globales.

Los componentes del IDE Flash se crearon para las aplicaciones de escritorio, no para dispositivos móviles. Por eso no están optimizados para este tipo de aparatos, de ahí que no se recomiende su uso, salvo contadas excepciones (ej. los que son exclusivos de Flash Lite).