DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Nº 3

"Un juego de niños"

Segundo cuatrimestre de 2015 — Métodos Numéricos

Integrante	LU	Correo electrónico
Frizzo, Franco	013/14	francofrizzo@gmail.com
Martínez, Manuela	160/14	martinez.manuela.22@gmail.com
Rabinowicz, Lucía	105/14	lu.rabinowicz@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Resumen

En el presente trabajo, se explora la aplicación de diferentes técnicas de interpolación segmentaria para la aplicación de un efecto de *slow motion* (cámara lenta) a secuencias de video. Se evalúan como alternativas la utilización de funciones constantes y lineales de a trozos y de *splines* cúbicos, y se realizan pruebas experimentales con cada una de ellas, con el fin de comparar su desempeño tanto desde el punto de vista de la eficiencia como de la calidad de los resultados arrojados, y de extraer conclusiones sobre la conveniencia de la utilización de una u otra en función de las características del video a procesar.

Palabras clave: interpolación, *splines*, *slow motion*, edición de video

Índice

1. Introducción

Un problema matemático que aparece repetidamente en diferentes contextos es el de la *interpolación*, que consiste en hallar una función que se ajuste a un determinado conjunto de datos. Entre las múltiples aplicaciones que tiene la resolución de este problema, se encuentra el de aplicar a videos el efecto de cámara lenta o *slow motion*, que resulta útil para la producción de peliculas, transmisiones deportivas y aplicaciones cientificas, entre otros.

Un video no es otra cosa que una sucesión de imágenes (cuadros o *frames*) en función del tiempo, cada una de las cuales consiste en una matriz de píxeles. El efecto de movimiento se consigue reproduciendo una determinada cantidad de estas imágenes por segundo; a esta cantidad se la denomina el *framerate* del video, y habitualmente es de entre 24 y 30 cuadros por segundo.

Para filmar videos en *slow motion*, se utilizan cámaras capaces de capturar una cantidad mucho mayor de cuadros por segundo (más de 100, en general), y luego se los reproduce a una velocidad normal. Si lo que se desea es obtener un video en cámara lenta a partir de uno normal, pueden intercalarse nuevos cuadros entre los ya existentes; para esto, es necesario utilizar alguna técnica para "inventar" la información correspondiente a los cuadros faltantes. Aquí es donde entra en juego la interpolación: si encontramos una función que interpole correctamente la información de los frames que conocemos, podemos calcular el valor de los nuevos evaluando esta función en instantes de tiempo intermedios.

En el transcurso de este trabajo, evaluaremos diferentes métodos que nos permitan lograr este objetivo, analizando su desempeño tanto desde el punto de vista de la performance como del de la calidad de los resultados obtenidos, y teniendo en cuenta estos datos para estudiar su aplicabilidad frente a diferentes contextos de uso.

2. Desarrollo

2.1. Conceptos teóricos

Conocemos como *interpolación* al proceso de hallar una función $f(x) = y$ que, dado un conjunto de puntos o pares de valores (x_k, y_k), cumpla $f(x_k) = y_k$ para todos estos puntos. En ciertos casos, además, es deseable que esta función cuente con determinadas propiedades, requeridas por el contexto de aplicación particular; por ejemplo, puede buscarse que sea continua o diferenciable.

Existen diversos enfoques para la resolución de este problema. Uno de los más comunes es la denominada *interpolación polinómica*, donde se busca encontrar un polinomio que se ajuste a los puntos dados, procurando, en general, que sea del menor grado posible. Como se detalla extensamente en [1, sección 3.1], dado un conjunto de n pares de valores reales, se puede demostrar la existencia de un único polinomio interpolador para todos ellos de grado menor o igual a n, conocido como el *polinomio de Lagrange*.

No obstante, la interpolación polinómica presenta algunas desventajas. En particular, cuando el conjunto de puntos de entrada tiene un tamaño considerable (como es el caso en el contexto de aplicación de este trabajo), la función hallada tiende a presentar fuertes oscilaciones, que puede llevar a obtener resultados no deseados (en [1, p. 158] puede verse un ejemplo de este comportamiento).

Como un enfoque alternativo ante este inconveniente, se presenta la *interpolación segmentaria*. Al igual que antes, se busca interpolar utilizando polinomios, pero en vez de pretender hallar un único polinomio que interpole todos los puntos, se busca uno diferente para cada uno de los segmentos determinados entre dos puntos consecutivos. De esta manera, se tiene un control mucho más fino sobre la función resultante, que permite evitar el comportamiento errático que se obtiene con el polinomio de Lagrange.

El caso más simple de interpolación segmentaria se da cuando los polinomios buscados en cada segmento son de grado 1; a esta técnica se la conoce como *interpolación lineal*. Para un conjunto de puntos (x_k, y_k), $k = 0, \ldots, n$, la función interpoladora se define dentro de cada intervalo $[x_k, x_{k+1}]$ $(k = 0, \ldots, n-1)$ como

$$f(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k)$$

y resulta continua en todo el intervalo $[x_0, x_n]$, pero no necesariamente derivable cuando $x = x_k$, donde, en general, se presentan "picos".

Dado que esto representa un obstáculo para gran cantidad de aplicaciones, surge la necesidad de buscar variantes más refinadas de interpolación segmentaria. Considerando polinomios de mayor grado, pueden controlarse más sus características, logrando una función que sea diferenciable en todo el dominio de interpolación. En este sentido, es común utilizar polinomios de grado 3, dando lugar a lo que se conoce como *splines* o trazadores cúbicos.

Dado un conjunto de puntos (x_k, y_k), $k = 0, \ldots, n$, un *spline* cúbico interpolador para estos puntos es una función S que satisface las siguientes condiciones:

(a) $S(x_k) = y_k$, para $0 \leq i < n$.

(b) S_k, la restricción de S al intervalo $[x_k, x_{k+1}]$, es un polinomio cúbico, para $0 \leq i < n$.

(c) $S'_k(x_{k+1}) = S'_{k+1}(x_{k+1})$, para $0 \leq i < n-1$.

(d) $S''_k(x_{k+1}) = S''_{k+1}(x_{k+1})$, para $0 \leq i < n-1$.

(e) $S''(x_0) = S''(x_n) = 0$. (Existen variantes de *splines* donde difiere esta condición, pero no serán relevantes en el contexto de este trabajo; ver [1, p. 146]).

Para hallar una función que satisfaga lo anterior, se define en cada segmento $[x_k, x_{k+1}]$ el polinomio

$$S_k(x) = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3$$

donde $k = 0, \ldots, n-1$, y luego se plantea y resuelve el sistema de $4n$ ecuaciones y $4n$ incógnitas (los coeficientes a_k, b_k, c_k, d_k) que se desprende de sustituir esta definición de S_k en las condiciones recién enunciadas.

2.2. Aplicación y métodos utilizados

Como se expuso en la Introducción, los conceptos recién presentados resultan de gran utilidad a la hora de resolver el problema de calcular los cuadros a intercalar en un video para aplicarle un efecto de *slow motion*.

Es importante señalar que, a diferencia de otros contextos relacionados con la edición de imágenes y video, en este caso es importante la variación de cada uno de los píxeles de forma independiente a lo largo del tiempo, y no la relación entre píxeles contiguos de un mismo cuadro. Por lo tanto, es válido considerar al video de entrada como una *matriz de sucesiones* de píxeles (en lugar de hacerlo como una *sucesión de matrices* de píxeles) y realizar la interpolación sobre cada una de ellas por separado, construyendo luego el video de salida a partir de la matriz formada por todos los resultados individuales obtenidos, tal y como ilustra la Figura 1.

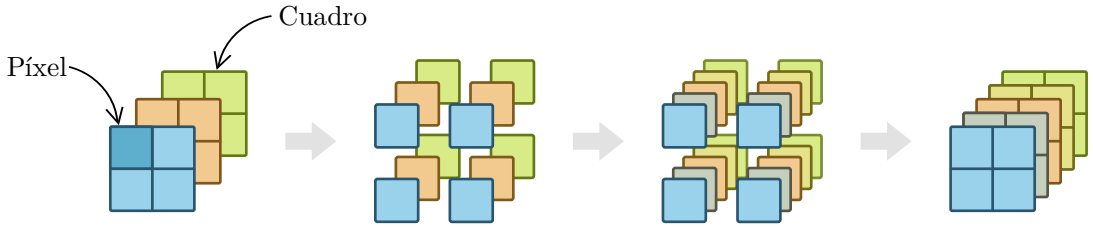


Figura 1: Interpolación de cada píxel de manera independiente.

Por otra parte, con el objetivo de simplificar la implementación y experimentación, se utilizaron únicamente videos en escala de grises, teniendo en cuenta que los resultados que se obtendrán serán fácilmente extensibles a videos en color, ya que en estos casos los canales rojo, verde y azul son procesados por separado.

En adelante, se utilizará la notación $\mathbf{V}_{ij}$ para referirnos a la sucesión de los valores que toma el píxel ubicado en la fila i, columna j del video $\mathbf{V}$ a lo largo del tiempo. $\mathbf{V}_{ij}$ será, por lo tanto, una sucesión de valores enteros en el rango $[0, 256)$. Además, con las notaciones $\mathbf{V}^{(k)}$ y $p^{(k)}$ se hará referencia al k-ésimo cuadro del video $\mathbf{V}$ y al valor del píxel p en el cuadro k-ésimo, respectivamente.

Para resolver el problema de calular los píxeles a intercalar en los videos, se emplearon tres métodos diferentes, que representan variantes de interpolación fragmentaria,[1] con el fin de estudiar su comportamiento, compararlos y evaluar sus ventajas y debilidades.

2.2.1. Método "*vecino más cercano*"

En este método, el más sencillo de los tres considerados, cada uno de los cuadros a generar se construye replicando los valores del cuadro original que se encuentra más próximo en el tiempo.

Más formalmente, sean $\mathbf{V}^{(k)}$ y $\mathbf{V}^{(k+1)}$ dos cuadros contiguos de un video, y supongamos que entre ellos desean intercalarse c nuevos cuadros. Si llamamos a estos *frames* $\mathbf{V'}^{(r)}$ para $r = 0, \ldots, c-1$, entonces cada uno de ellos cumplirá

$$\begin{aligned} \mathbf{V'}^{(r)} &= \mathbf{V}^{(k)} && \text{si } r < c/2 \\ \mathbf{V'}^{(r)} &= \mathbf{V}^{(k+1)} && \text{si } r \geq c/2 \end{aligned}$$

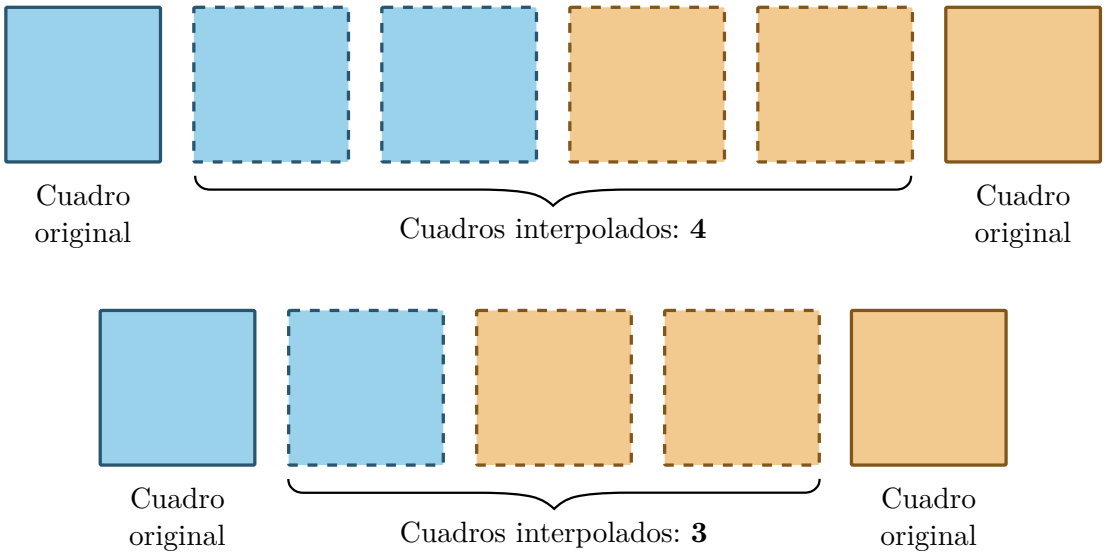


Figura 2: Método "*vecino más cercano*". Manejo de píxeles centrales.

Nótese el manejo que se produce de los píxeles centrales, ilustrado en la Figura 2. Si la cantidad de *frames* a agregar es par, la mitad de ellos replicarán el píxel inmediato anterior del video original, y la otra mitad el píxel inmediato siguiente. Si, en cambio, la cantidad es impar, el píxel central replicará el píxel inmediato siguiente. Este desbalance se verá compensado al interpolar el par de píxeles siguientes, evitando que se produzcan efectos no deseados.

2.2.2. Método de interpolación lineal

Una forma algo más sofisticada de obtener los cuadros buscados es interpolando, entre cada par de *frames* consecutivos, mediante una función lineal. Este método, a diferencia del anterior,

[1] En rigor, el primero de los métodos considerados no es interpolación segmentaria, dado que la función que se obtiene no es siquiera continua; sin embargo, sigue la idea general de definir de a segmentos la función interpoladora.

"inventa" nueva información para llenar el espacio entre los cuadros originales, combinando los datos que estos últimos proporcionan. De esta forma, se espera obtener una trancisión más suave entre los *frames* del video.

El método, expresado formalmente, intercala c cuadros nuevos entre dos cuadros $\mathbf{V}^{(k)}$ y $\mathbf{V}^{(k+1)}$ consecutivos de la siguiente manera: si llamamos $\mathbf{V'}^{(r)}$ a los cuadros interpolados, para $r = 0, \ldots, c-1$, entonces

$$\mathbf{V'}^{(r)} = \mathbf{V}^{(k)} + \Delta(r+1)$$

donde

$$\Delta = \frac{\mathbf{V}^{(k+1)} - \mathbf{V}^{(k)}}{c+1}$$

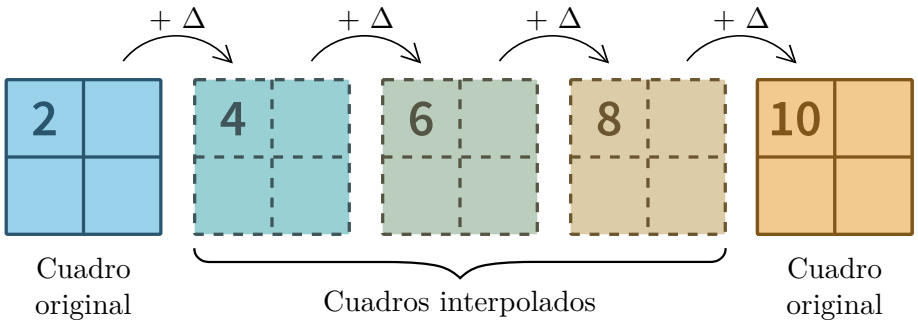


Figura 3: Método de interpolación lineal. Aquí $\Delta = \dfrac{10-2}{3+1} = 2$.

2.2.3. Método de interpolación por *splines*

El último de los tres métodos evaluados consiste en utilizar *splines* cúbicos para obtener los valores de los píxeles correspondientes a los cuadros a intercalar. Al igual que el caso anterior, los cuadros intermedios contienen información que no estaba presente en el video original, aunque esta vez se busca aprovechar la diferenciabilidad de los *splines*, propiedad no presente en la interpolación lineal, para obtener resultados más naturales.

Dado un video con *frames* $\mathbf{V}^{(0)}, \ldots, \mathbf{V}^{(n)}$, si entre cada par de cuadros quieren agregarse c nuevos cuadros, se procesa cada píxel $p = V_{i,j}$ de la siguiente manera: se consideran los puntos $(k, p^{(k)})$ y se construyen los n polinomios cúbicos S_n que interpolan en los segmentos $[k, k+1]$ $(k = 0, \ldots, n-1)$. Luego, para cada par de cuadros consecutivos $p^{(k)}$, $p^{(k+1)}$ se calcula el valor del píxel en los c cuadros intermedios $p'^{(r)}$, $r = 0, \ldots, c-1$, evaluando el polinomio S_k a intervalos regulares, es decir

$$p'^{(r)} = S_k\left(\frac{r}{c+1}\right)$$

De esta forma, como ya fue explicado (ver Figura 1), pueden componerse los cuadros a intercalar a partir de calcular, independientemente, el valor de cada uno de sus píxeles.

Variante por bloques

Para este último método se consideró también una variante en la que, en lugar de calcular el *spline* utilizando los valores de todos los cuadros, se divide al video en bloques de menor duración, interpolando dentro de cada uno de ellos, y luego se unen los resultados obtenidos. De esta manera, los sistemas de ecuaciones a resolver son más pequeños. Más allá de esto, la alternativa no presenta grandes diferencias desde el punto matemático. Cabe mencionar la forma en que se manejan los casos borde: se hace necesario que los bloques tengan un *frame* solapado en sus extremos (ver Figura 4) porque, en caso contrario, los cuadros a intercalar entre el final de cada bloque y el comienzo del siguiente no se calcularían como parte de ninguno de ellos.

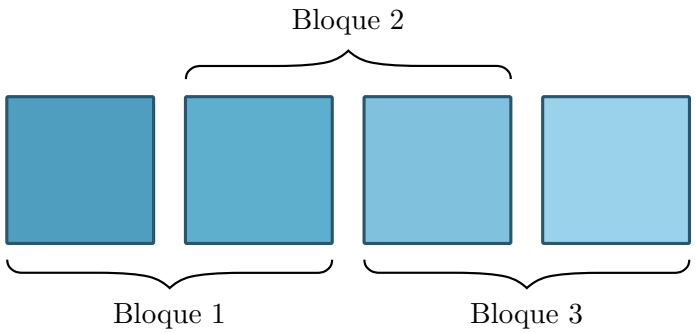


Figura 4: División del video en bloques para su procesamiento.

2.3. Implementación

Todos los métodos fueron implementados en lenguaje `C++`. Internamente, el video es considerado como una matriz tridimensional, donde la primeras dos dimensiones corresponden a la ubicación de los píxeles (fila y columna), mientras que la tercera a los *frames* que componen el video. De esta manera, en los tres casos se itera sobre los píxeles del video, procesándolos de manera independiente según el algoritmo correspondiente, para luego unir los resultados nuevamente y producir el video final.

La implementación de los métodos "*vecino más cercano*" y de interpolación lineal son inmediatas, ya que se limitan a aplicar repetidamente las fórmulas enunciadas en la subsección anterior, por lo que no se considera necesario brindar más detalles sobre las mismas.

En cuanto al método de interpolación por *splines*, se barajaron diferentes alternativas para su implementación. En una primera instancia, se planteó la posibilidad de utilizar algoritmos para la resolución de sistemas de ecuaciones en general, como los de Eliminación Gaussiana o Factorización LU. Finalmente, se decidió aprovechar el hecho de que el sistema a resolver presenta características muy particulares y puede ser ampliamente simplificado, y se implementó un algoritmo iterativo, extraído de [1, Algoritmo 3.4], cuyo pseudocódigo puede leerse en el Apéndice A.

2.3.1. Verificación de la correctitud de la implementación

Para corroborar la correctitud de los métodos, se elaboró un pequeño caso de test. Este consistió en un video compuesto por dos cuadros con cuatro filas y cuatro columnas cada uno. Se

aplicaron los tres métodos, repitiendo cada uno de ellos con diferentes valores para el parámetro de cantidad de cuadros a generar. Los valores en los dos cuadros del video de prueba fueron pensados para probar diferentes casos borde, y verificar que los resultados fueran los deseados.

Para "vecino más cercano", se buscó que la primera mitad de los cuadros generados fuera exactamente igual al primer cuadro del video original, y los restantes, idénticos al segundo. Para el método de interpolación lineal, se calculó manualmente la función lineal que debía obtenerse, y se la evaluó en los valores intermedios para compararla con los resultados arrojados por el algoritmo. Para el método de interpolación por *splines*, se tuvo en cuenta que, al tener solamente dos cuadros originales, la función interpoladora debería resultar lineal, por lo que los valores obtenidos deberían coincidir con los obtenidos con el método anterior.

Si bien la prueba recién mencionada no tiene el valor de una batería exhaustiva de casos de test o una demostración de correctitud, se consideró que dado su éxito, además del hecho de que los resultados producidos para videos reales fueron razonables, era posible concluir que la implementación de los algoritmos funciona correctamente.

3. Experimentación y discusión

A continuación, se presentan las pruebas experimentales que se realizaron con el fin de evaluar y comparar los diferentes métodos ya presentados, junto con los resultados obtenidos y una discusión de los mismos.

Todos los experimentos se ejecutaron en las computadoras del laboratorio 4 del Departamento de Computación (FCEN - UBA). Como datos de entrada para las pruebas, se seleccionaron videos extraidos de la serie de televisión estadounidense *Friends*, teniendo en cuenta características que los hacían apropiados para poner a prueba diferentes aspectos de cada algoritmo. Los videos utilizados se incluyen con los archivos fuentes del trabajo en el directorio `data`; por otra parte, el directorio `exp` contiene una serie de scripts en lenguaje Bash que permite reproducir los experimentos.

3.1. Experimento 1: Pruebas de rendimiento

Presentación

Este experimento se realizó con el objetivo de este experimento de comparar el rendimiento entre los métodos "*vecino más cercano*", interpolación lineal e interpolación por *splines*, teniendo en cuenta el tiempo insumido para la ejecución de cada uno de ellos. Se realizaron dos tipos de pruebas:

(a) Rendimiento en función de la duración del video de entrada.

(b) Rendimiento en función del tamaño en píxeles del video de entrada.

Para ello primero se variará la duración del video dejando como constante el tamaño y para cada una de las duraciones se medirá el tiempo de ejecución de cada uno de los métodos. Luego se dejará constante el tiempo de duración y se variará el tamaño del video. Con el fin de mejorar la precisión de los resultados y evitar posibles interferencias (por ejemplo, las generadas por otras tareas que puedan estar ejecutando la computadora) cada medición se repitió 10 veces, tomando luego el promedio de los datos obtenidos.

En cuanto al método de *splines*, el rendimiento de la variante del mismo que procesa el video dividiéndolo en bloques de menor tamaño se dejó para ser evaluado en experimentos posteriores.

Datos de entrada

Para realizar las pruebas, se utilizaron los siguientes 6 videos de entrada:

(a) Para analizar el tiempo de ejecución según la duración del video, se consideró un video de 378 cuadros (15 segundos de duración) (`exp1-a-15.avi`), que fue cortado para generar otros dos casos de prueba de 253 cuadros (10 segundos) (`exp1-a-10.avi`) y 127 cuadros (5 segundos) (`exp1-a-5.avi`).

(b) Para analizar el tiempo de ejecución según el tamaño en píxeles del video, se consideró un video de $352 \times 264 = 92928$ píxeles (`exp1-b-1.avi`), a partir del cual se generaron dos versiones reducidas de $176 \times 123 = 23232$ (`exp1-b-2.avi`) y $120 \times 90 = 10800$ (`exp1-b-3.avi`).

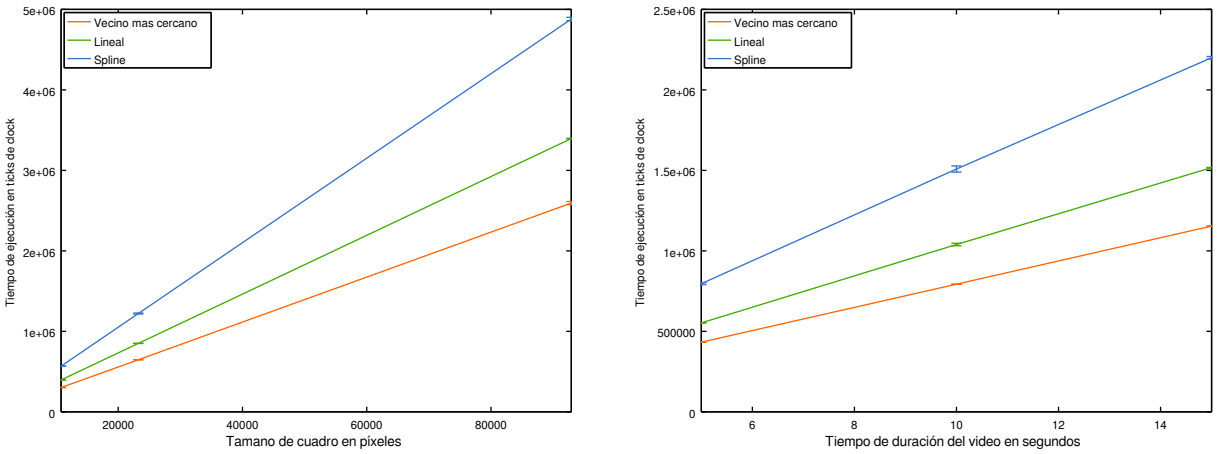
En ambos casos, las medidas se eligieron de forma arbitraria, procurando que los casos de prueba no fueran excesivamente grandes para no prolongar innecesariamente la duración de las pruebas. Como cantidad de cuadros a intercalar se eligió 3, número elegido arbitrariamente, ya que no se consideró que este parámetro fuera de relevancia para el experimento.

Hipótesis

Teniendo en cuenta la complejidad de las operaciones requeridas por cada uno de los métodos, la hipótesis fue que el método mas eficiente sería "*vecino más cercano*", seguido por el de interpolación lineal y por ultimo el de *splines*. En particular, el método "*vecino más cercano*" se limita a copiar valores, casi sin realizar opreaciones adicionales; el de interpolación lineal debe realizar operaciones matemáticas sencillas sobre los valores, mientras que el de interpolación por *splines* debe resolver un sistema de ecuaciones considerablemente más complejo.

Por otra parte, considerando que los píxeles del video se procesan de manera independiente, se consideró esperable que el tiempo de ejecución aumentara linealmente con la cantidad de píxeles del video de entrada. Además, dado que los tres algoritmos realizan iteraciones simples sobre los *frames* del video de entrada, y tras realizar un sencillo análisis de complejidad de los mismos, se elaboró la hipótesis de que el tiempo de ejecución de los tres variaría también linealmente con la longitud del video de entrada.

Resultados



Resultados obtenidos en el experimento 1

Discusión

Como se puede observar en los resultados, pudo verificarse la hipótesis, ya que tanto al comparar el tiempo de ejecución con respecto al tamaño de los cuadros como cuando se lo compara con diferentes duraciones de videos, el método de "*vecino más cercano*" es el más eficiente, seguido del de interpolación lineal y luego el de interpolación por *splines*.

A partir del análisis de los resultados y la observación de los gráficos, se llegó a la conclusión de que la hipótesis de la complejidad lineal de los algoritmos queda claramente confirmada, como puede observarse en los gráficos; no se considera necesario agregar más pruebas con videos de

diferentes tamaños ni de diferente duración.

3.2. Experimento 2: Comparación de resultados con videos reales

Presentación

El objetivo de este experimento fue contrastar los resultados arrojados por cada algoritmo con un video real. La metodología consistió en tomar el video original, eliminar una cierta cantidad de cuadros y luego recrearlos con cada uno de los algoritmos, para así observar cuál logró una mejor aproximación de la solución real. Como métrica de este último valor, se consideró el error cuadrático medio (ECM) *frame* a *frame*, definido para cada cuadro del video como

$$\mathrm{ECM}\left(\mathbf{V}^{(k)}, \mathbf{V'}^{(k)}\right) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(\mathbf{V}_{ij}^{(k)}, \mathbf{V'}_{ij}^{(k)}\right)^2$$

donde m es la cantidad de filas y n la cantidad de columnas del video, y se calculó también el *error total*, definido como

$$\mathrm{E}\left(\mathbf{V}, \mathbf{V'}\right) = \frac{1}{G}\sum_{k=0}^{N}\mathrm{ECM}\left(\mathbf{V}^{(k)}, \mathbf{V'}^{(k)}\right)$$

donde N es la cantidad total de frames del video original, y G es la cantidad de cuadros del total que fueron generados artificialmente (se consideró este último valor para el cociente, en lugar de N, para evitar considerar en la métrica los cuadros que no fueron producidos por el algoritmo).

Datos de entrada

Se tomaron dos casos diferentes de prueba. Para el primero (`exp2-1.avi`), se eligió un video suave, sin saltos bruscos de escena; en la primera parte del video hay movimiento en la primer mitad del video y en el final no. El segundo video (`exp2-2.avi`) presenta una primera parte con movimiento de cámara, luego un intervalo casi sin movimiento de imagen, y cerca del final, un corte de escena.

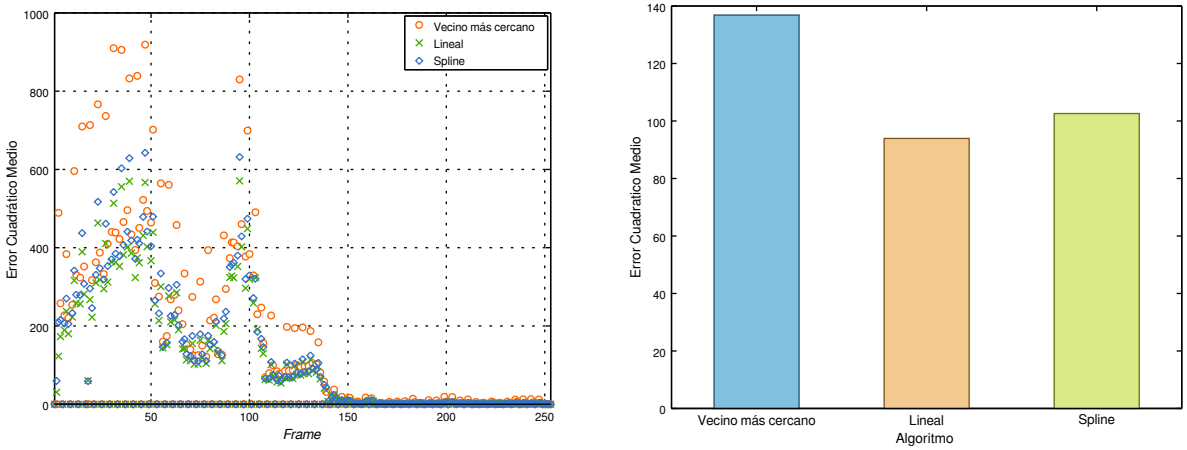
En ambos casos se eliminaron tres de cada cuatro cuadros del video original, procediendo luego a replicarlos con cada uno de los algoritmos a evaluar.

Hipótesis

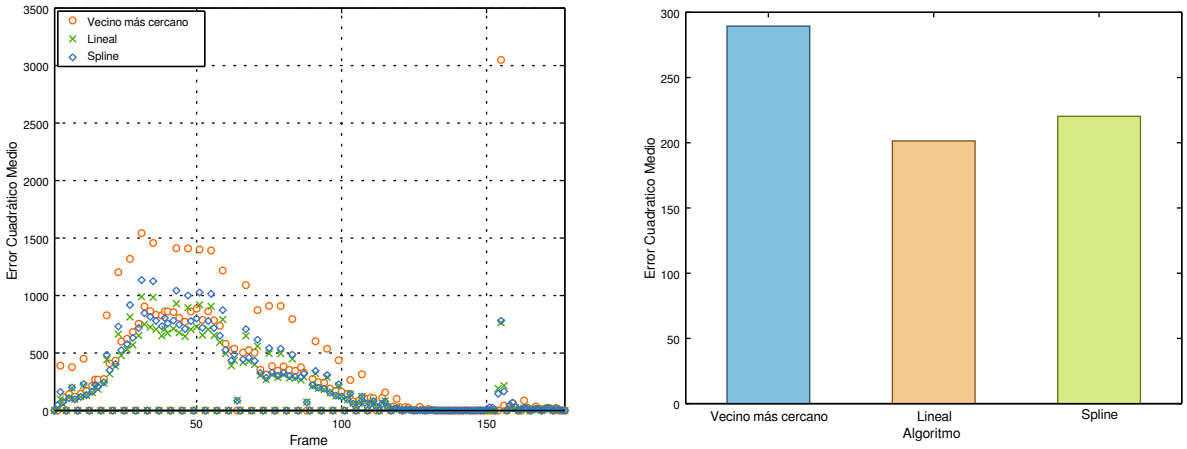
El experimento se realizó bajo la hipótesis de que la mejor aproximación al video original sería la del método de *splines*, dado que, para cada píxel, considera una mayor cantidad de valores a lo largo del tiempo y genera una función diferenciable, que cabe esperar que se parezca más al movimiento natural presente en un video. Los otros dos métodos, en cambio, no devuelven resultados "suaves"; el método de interpolación lineal, solo toma 2 valores consecutivos del píxel para generar el nuevo cuadro, y el de vecino mas cercano se limita a replicar cuadros ya existentes, con lo que cabe esperar que produzca videos con saltos bruscos entre imágenes fijas, teniendo un desempeño pobre a la hora de producir impresión de movimiento.

Resultados

Video 1



Video 2



Resultados obtenidos para el experimento 2. Los gráficos de la izquierda representan el error cuadrático medio en función del tiempo para cada algoritmo. Los gráficos de la derecha representan el error total para cada algoritmo.

Discusión

Del análisis de los valores obtenidos para el *error total*, pudo desprenderse que, para ambos videos, el método menos preciso es "vecino más cercano". No obstante, de forma contraria a lo esperado, se halló que la interpolación lineal aproxima mejor al video original que el método de *splines*. Dado que existe la posibilidad de que este resultado se deba a las características particulares de los videos elegidos, se considera pendiente para experimentos futuros corroborar este comportamiento y determinar la razón por la que ocurren estas diferencias.

Observando el comportamiento del error cuadrático medio a lo largo de ambos videos se pudo notar que, como cabía esperar, uno de cada cuatro cuadros arroja un error cuadrático medio nulo; esto se debe, evidentemente, a que este cuadro mo es generado artificialmente, sino copiado directamente del video original.

Es posible notar, en los 3 métodos, que en los resultados del primer video el error es considerablemente mayor durante los momentos iniciales del video, donde se observan personas en

movimiento pasando por delante de la protagonista; por el contrario, cerca del final, cuando en el video solo se encuentra la protagonista, con poco movimiento, el error cuadrático medio de los cuadros disminuye notoriamente.

Con los resultados del segundo video se pudo ver claramente que, de nuevo, el momento donde hay más diferencia entre los valores reales del cuadro y los valores del cuadro generado es cuando la cámara se encuentra en movimiento. Luego, cuando esta queda fija por un tiempo, el error disminuye notoriamente. Al final del video, se observó un pico brusco en los valores del error, que coincide con un cambio total de escena. Es relevante notar que para el método "vecino más cercano" hay un solo cuadro con una enorme diferencia; para interpolación lineal, hay más cuadros afectados pero presentan un error menor. En el método de *splines*, los cuadros afectados son aún más. Esto coincide con la naturaleza de los métodos, y resulta esperable teniendo en cuenta la cantidad de frames que considera cada uno de ellos para calcular los que son agregados.

Partiendo del supuesto de que este pico en el valor del error cuadrático medio se produjo debido a la gran variación repentina en los valores de los píxeles, se realizó un nuevo experimento para profundizar esta idea.

3.2.1. Experimento 2, bis: Análisis de cambios bruscos de escena

Presentación

En este experimento se buscó aislar la situación del experimento anterior donde se produjo un cambio brusco de escena, para confirmar la idea de que, ante esta situación, la medición de error con respecto al video real se dispara.

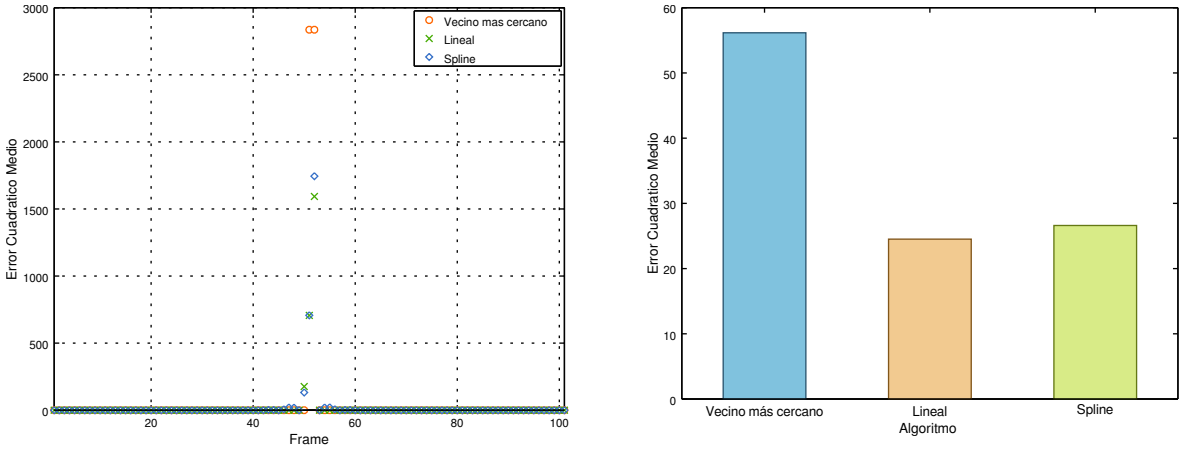
Datos de entrada

Se utilizó el video *exp2-3.avi*, que consiste simplemente en dos imágenes estáticas con fondo blanco, con un corte brusco al pasar de una a la otra. Se eliminaron 3 cuadros del video original, siendo luego recreados por medio de los tres métodos.

Hipótesis

Se partió de la hipótesis de que, en el momento donde cambia la foto, el ECM tomará valores muy elevados, resultando nulo en el resto del video.

Resultados



Resultados obtenidos para el experimento 2, bis. El gráfico de la izquierda representa el error cuadrático medio en función del tiempo para cada algoritmo. El gráfico de la derecha representa el error total para cada algoritmo.

Discusión

Como se puede observar en los gráficos, se obtuvo un ECM nulo cuando el video se corresponde con la imagen fija; en el momento en el que ésta cambia, el ECM toma valores mayores. En el método de vecino mas cercano, solo dos cuadros presentan un error no nulo, pero este presenta valores muy elevados. Esto se debe a que, dadas las características del método, el corte brusco de escena se reproduce, pero en un momento erróneo. De esta forma, los cuadros que no coinciden son completamente diferentes, arrojando valores de error más altos que los de los demás métodos. Cabe destacar que, más allá de esto, el video producido por este procedimiento es el que mejor aproxima visualmente al video original.

En el método de interpolación lineal, en lugar del corte brusco de escena, se obtuvo una fusión suave entre las dos imágenes a lo largo de los tres cuadros producidos por la interpolación. Esto se vio claramente reflejado en el análisis del error cuadrático medio.

Con *splines* el resultado obtenido fue similar, pero dado que la función interpoladora se ve afectada por una cantidad más grande de cuadros del video, el error se presenta distribuido a lo largo de un intervalo mayor.

En los momentos donde la imagen se mantiene constante, en los 3 métodos, el error cuadrático medio es nulo, reflejando el hecho de que al interpolar entre dos cuadros iguales, ambos métodos generan una serie de cuadros iguales a los dos entre los que se realiza la interpolación.

3.3. Experimento 3: Análisis de la variante por bloques del método de *splines*

Presentación

En este experimento se compararon los algoritmos de *splines* con y sin bloques y se estudió su aproximación a la solución real. La metodología fue idéntica a la del experimento 2, pero se generaron los nuevos videos ejecutando el método de *splines* con diferentes tamaños de bloques.

También se analizó el tiempo de ejecución para el método de *splines* con diferentes tamaños de bloques y sin utilizar bloques en absoluto. En este caso la metodología fue idéntica a la del

experimento 1, realizando 10 repeticiones de cada medición para evitar posibles interferencias.

Datos de entrada

Se utilizaron dos videos, ya empleados en el experimento anterior: el primero (*exp2-1.avi*) con mucho movimiento en la primer mitad del video y poco hacia el final, y el segundo (*exp2-2.avi*), con movimiento de cámara al comienzo, luego un intervalo con la cámara prácticamente fija, sin movimiento de imagen, y al final un cambio brusco de escena.

En ambos casos se eliminaron tres de cada cuatro cuadros del video original, procediendo luego a replicarlos con cada uno de los algoritmos a evaluar.

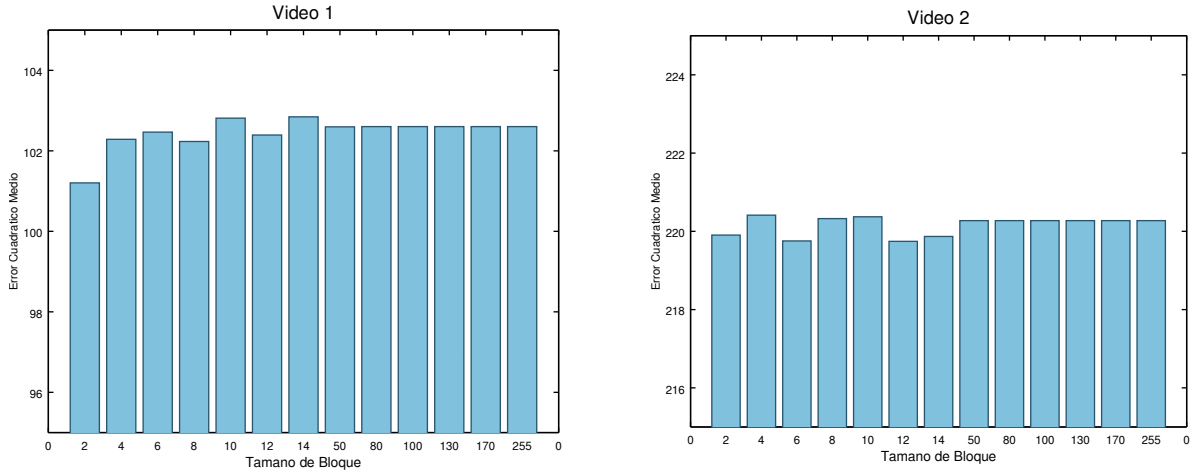
En una primera instancia, el experimento se realizó tomando tamaños de bloques grandes, mayores que 20. Dado que los resultados obtenidos se consideraron poco relevantes, ya que las diferencias entre los valores del error promediado para los diferentes tamaños de bloques eran mínimas, se decidió experimentar considerando tamaños de bloques más pequeños. Los valores finalmente empleados fueron los siguientes: 2, 4, 6, 8, 10, 12, 14, 50, 80, 100, 130, 170.

Hipótesis

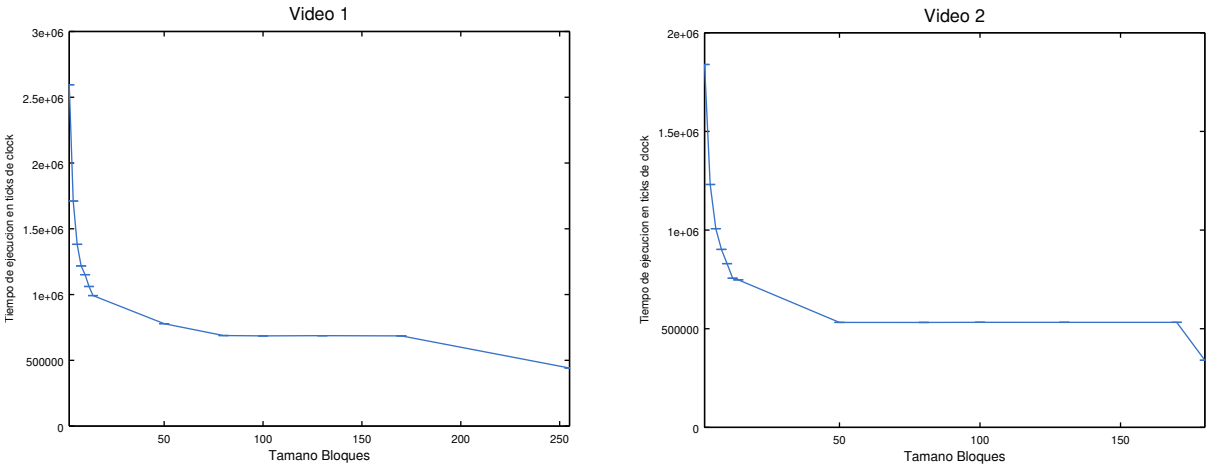
Se partió de la hipótesis de que la mejor aproximación al video original sería la del método de *splines* sin bloques, ya que para interpolar se tiene en cuenta más información, tomando valores a lo largo de más cuadros. Se consideró esperable que los resultados difirieran solamente alrededor de los puntos de corte de los bloques, habiendo pocas diferencias en las zonas centrales de los mismos, lo cual provocaría que los resultados con bloques grandes fueran bastante similares a los obtenidos al ejecutar el método sin utilizar bloques.

Se supuso que comparando los tiempos de ejecución entre los diferentes tamaños de bloques, por la forma en la que fue implementado el método, con más bloques sería mayor el tiempo de ejecución; dado a que la división en bloques genera que haya cuadros que se procesen dos veces, ya que los bloques tienen un solapamiento de un *frame* en los extremos. Así, la cantidad de cuadros a procesar se incrementaría en tantos bloques como entraran en el video. Si el método hubiera sido implementado resolviendo el sistema a través de un algoritmo diferente, como fue evaluado en un primer monento, el método de splines podría no haber resultado lineal en la cantidad de cuadros considerados; en tal caso, la división en bloques podría haber representado una importante mejora de rendimiento.

Resultados

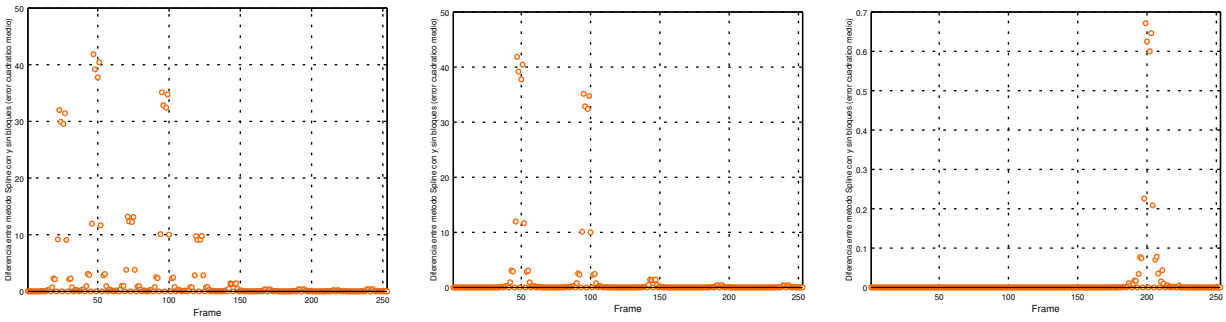


Resultados obtenidos para el experimento 3. Error cuadrático medio en función del tamaño del bloque.

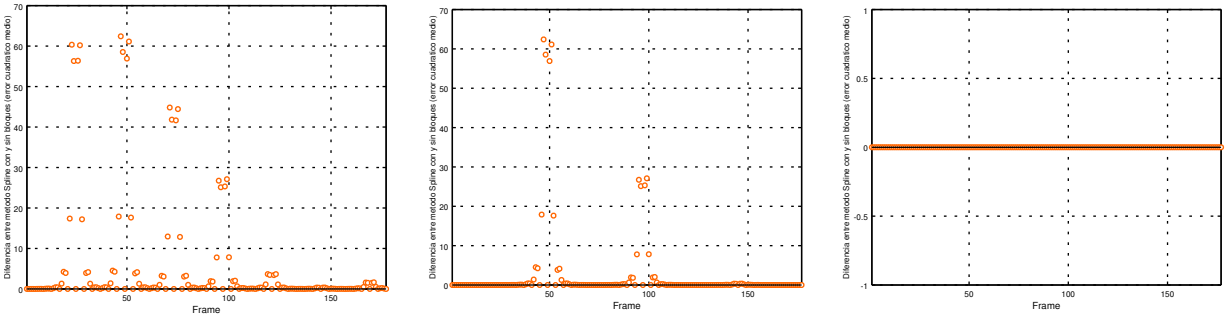


Resultados obtenidos para el experimento 3. Tiempo de ejecución en función del tamaño del bloque.

Video 1



Video 2



Resultados obtenidos para el experimento 3. Diferencia (según Error Cuadrático Medio) entre el video procesado de a bloques y el obtenido sin realizar la división. Los tamaños de bloques considerados son 6, 12 y 50, de izquierda a derecha.

Discusión

En el caso del spline en el cual se toma un tamaño de bloque muy cercano al de la cantidad de cuadros del video, el error que se genera en los píxeles que quedan es tan chico que en el error total queda igual que cuando no se divide por bloques. Para el resto de los tamaños las diferencias entre los errores es mínima y va variando dependiendo del tamaño sin poder asegurar que cuanto mas chicos sean los bloques mayor será la diferencia. Dejamos para experimentos futuros observar por que suceden estas diferencias.

Se puede observar que cuanto menor sea el tamaño de los bloques, el tiempo de ejecución del algoritmo será mayor. Esto se debe a que a medida que aumentamos la cantidad de bloques que haya en el video, mayor será la cantidad de cuadros a analizar ya que el cuadro que se encuentra entre dos bloques deberá ser utilizado para generar la función interpoladora en ambos dos.

3.4. Análisis cualitativo de los resultados obtenidos

Además de los experimentos ya expuestos, se realizó un estudio cualitativo (inevitablemente subjetivo) de los *artifacts*, es decir, los errores visuales, producidos por cada uno de los métodos analizados. Se tomaron dos videos, uno con y uno sin movimientos bruscos de cámara, eliminando en ambos casos una cierta cantidad de cuadros para luego recrearlos por medio de los tres métodos y realizar el análisis cualitativo sobre estos resultados.

Datos de entrada

Se utilizaron dos videos, ya empleados en los experimentos anteriores: el primero (*exp2-1.avi*) con mucho movimiento en la primer mitad del video y poco hacia el final, y el segundo (*exp2-2.avi*), con movimiento de cámara al comienzo, luego un intervalo con la cámara prácticamente fija, sin movimiento de imagen, y al final un cambio brusco de escena.

En ambos casos se eliminaron tres de cada cuatro cuadros del video original, procediendo luego a replicarlos con cada uno de los algoritmos a evaluar.

Hipótesis

La hipótesis fue que en el video generado con el método de vecino más cercano se vería la imagen congelada por momentos, y el movimiento se daría a través de pequeños saltos, ya que este algoritmo simplemente copia los cuadros ya existentes, haciendo que se muestren por un mayor intervalo de tiempo, y sin suavizar en forma alguna la transición entre ellos. Dado que no se genera información nueva, no se esperó la aparición de *artifacts* de ningún tipo.

En el método de interpolación lineal, se supuso que se generarían *artifacts* en los momentos en que hubiera movimientos bruscos, ya que para interpolar se toman de a dos cuadros del video original, generando cuadros con una fusión de imágenes diferentes, lo cual puede llevar a la aparición de defectos. Al interpolar con el método de *splines* se conjeturó que se obtendría un resultado similar, pero teniendo en cuenta la posibilidad de que los *artifacts* fueran más notorios, dado que la interpolación considera una cantidad de *frames* mayor, pudiendo exagerar el efecto recién mencionado.

Discusión

En el primer video, las versiones generadas por los métodos de interpolación lineal y por *splines* mostraron defectos en los *frames* donde pasan personas caminando, ya que por momentos se las ve dobles o con partes del cuerpo duplicadas; lo mismo ocurrió con el segundo video, donde al moverse la cámara las personas y el fondo parecen duplicarse. En los videos generados por el método de vecino mas cercano este efecto no se produce, pero si se observó que el video parece detenerse por momentos, dando la sensación de que la reproducción está "trabada".

Al tomar los videos generados por los métodos de interpolación lineal y *splines* y observarlos cuadro a cuadro, se pudo observar ver que existen cuadros donde no se detectan *artifacts*; estos cuadros son los copiados desde el video original, donde, al igual que el método "vecino más cercano", no se produjeron perturbaciones.

4. Conclusiones

Como conclusión del trabajo, y adoptando una visión general sobre los experimentos realizados, es posible afirmar que ninguno de los métodos evaluados resulta superior a los demás en todos los escenarios. La elección de uno u otro deberá ser evaluada cuidadosamente a partir de las características propias del video a procesar, y teniendo en cuenta si busca optimizarse el tiempo de ejecución o la calidad de los resultados obtenidos. En muchas ocasiones, puede resultar conveniente combinar dos de los métodos; por ejemplo, en los momentos en que hay cortes de escena, podría ser deseable aplicar el método "vecino más cercano", a pesar de que se utilice uno de los otros dos para el resto del video, ya que, como pudo observarse, el primero hace un mejor trabajo en ese tipo de situaciones. No obstante, para ello sería necesario disponer de un método para reconocer estos cambios de escena, lo cual, en determinados contextos, puede resultar inviable.

Es de interés notar que, a pesar de que las hipótesis iniciales no lo preveían, el método de interpolación lineal produjo resultados de una calidad considerable, aproximándose más en las métricas a los videos reales que el de interpolación por splines, produciendo en general *artifacts* menos pronunciados y teniendo al mismo tiempo un mejor rendimiento temporal y una mayor facilidad de implementación. Hay que tener en cuenta, sin embargo, que los videos generados por el método de *splines* resultan más suaves a la vista, siendo menos perceptibles que en los videos generados mediante interpolación lineal los puntos de quiebre marcados por los cuadros no interpolados. Aquí, otra vez, se hace necesario un análisis particular del caso de aplicación para decidir cuál de los métodos es conveniente aplicar.

Apéndices

A. Pseudocódigo del método de interpolación por *splines*

El pseudocódigo que se transcribe a continuación es una versión modificada[2] del que puede encontrarse en [1, Algoritmo 3.4], y permite hallar, dado un conjunto de pares de valores (k, y_k), $k = 0, \ldots, n$, los $4n$ coeficientes

$$a_k, b_k, c_k, d_k \qquad k = 0, \ldots, n-1$$

de los n polinomios

$$S_k = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3 \qquad k = 0, \ldots, n-1$$

que conforman el *spline* cúbico que interpola los puntos dados.

Algoritmo 1: Coeficientes de *spline* cúbico interpolador

Entrada: n, $a_0 = y_0$, …, $a_n = y_n$

Salida : a_j, b_j, c_j, d_j para $j = 0, \ldots, n-1$

para $i = 1, 2, \ldots, n-1$ **hacer** $\alpha_i \leftarrow 3a_{i+1} - 6a_i + 3a_{i-1}$
$l_0 \leftarrow 1$
$\mu_0 \leftarrow 0$
$z_0 \leftarrow 0$
para $i = 1, 2, \ldots, n-1$ **hacer**
    $l_1 \leftarrow 4 - \mu_{i-1}$
    $\mu_1 \leftarrow 1/l_1$
    $z_1 \leftarrow (\alpha_i - z_{i-1})/l_i$
fin
$z_n \leftarrow 0$
$c_n \leftarrow 0$
para $j = n-1, n-2, \ldots, 0$ **hacer**
    $c_j \leftarrow z_j - \mu_j c_{j+1}$
    $b_j \leftarrow (a_{j+1} - a_j) - (c_{j+1} + 2c_j)/3$
    $d_j \leftarrow (c_{j+1} - c_j)/3$
fin
devolver a_j, b_j, c_j, d_j para $j = 0, \ldots, n-1$

[2]El algoritmo original consideraba también como datos de entrada los valores de la variable independiente en cada uno de los puntos a interpolar. Este detalle fue omitido, ya que en el contexto del presente trabajo estos valores son simplemente los naturales consecutivos $0, 1, \ldots, n$. De esta forma, el algoritmo aquí presentado es ligeramente más sencillo que el que puede encontrarse en la fuente citada.

B. Enunciado del trabajo práctico

B.1. Introducción

¿Quién nunca ha visto un video gracioso de bebés? El éxito de esas producciones audiovisuales ha sido tal que el sitio youborn.com es uno de los más visitados diariamente. Los dueños de este gran sitio, encargado de la importantísima tarea de llevar videos graciosos con bebés a todo el mundo, nos ha pedido que mejoremos su sistema de reproducción de videos.

Su objetivo es tener videos en cámara lenta (ya que todos deseamos tener lujo de detalle en las expresiones de los chiquilines en esos videos) pero teniendo en cuenta que las conexiones a internet no necesariamente son capaces de transportar la gran cantidad de datos que implica un video en *slow motion*. La gran idea es minimizar la dependencia de la velocidad de conexión y sólo enviar el video original. Una vez que el usuario recibe esos datos, todo el trabajo de la cámara lenta puede hacerse de modo offline del lado del cliente, optimizando los tiempos de transferencia. Para tal fin utilizaremos técnicas de interpolación, buscando generar, entre cada par de cuadros del video original, otros ficticios que nos ayuden a generar un efecto de slow motion.

B.2. Definición del problema y metodología

Para resolver el problema planteado en la sección anterior, se considera el siguiente contexto. Un video está compuesto por cuadros (denominados también *frames* en inglés) donde cada uno de ellos es una imagen. Al reproducirse rápidamente una después de la otra percibimos el efecto de movimiento a partir de tener un "buen frame rate", es decir una alta cantidad de cuadros por segundo o fps (frames per second). Por lo general las tomas de cámara lenta se generan con cámaras que permiten tomar altísimos números de cuadros por segundo, unos 100 o más en comparación con entre 24 y 30 que se utilizan normalmente.

En el caso del trabajo práctico crearemos una cámara lenta sobre un video grabado normalmente. Para ello colocaremos más cuadros entre cada par de cuadros consecutivos del video original de forma que representen la información que debería haber en la transición y reproduciremos el resultado a la misma velocidad que el original. Las imágenes correspondientes a cada cuadro están conformadas por píxeles. En particular, en este trabajo utilizaremos imágenes en escala de grises para disminuir los costos en tiempo necesarios para procesar los datos y simplificar la implementación; sin embargo, la misma idea puede ser utilizada para videos en color.

El objetivo del trabajo es generar, para cada posición (i, j), los valores de los cuadros agregados en función de los cuadros conocidos. Lo que haremos será interpolar en el tiempo y para ello, se propone considerar al menos los siguientes tres métodos de interpolación:

1. *Vecino más cercano:* Consiste en rellenar el nuevo cuadro replicando los valores de los píxeles del cuadro original que se encuentra más cerca.

2. *Interpolación lineal:* Consiste en rellenar los píxeles utilizando interpolaciones lineales entre píxeles de cuadros originales consecutivos.

3. *Interpolación por Splines:* Simliar al anterior, pero considerando interpolar utilizando splines y tomando una cantidad de cuadros mayor. Una alternativa a considerar es tomar la

información de bloques de un tamaño fijo (por ejemplo, 4 cuadros, 8 cuadros, etc.), con el tamaño de bloque a ser determinado experimentalmente.

Cada método tiene sus propias características, ventajas y desventajas particulares. Para realizar un análisis cuantitativo, llamamos F al frame del video real (ideal) que deberíamos obtener con nuestro algoritmo, y sea $\bar{F}$ al frame del video efectivamente construido. Consideramos entonces dos medidas, directamente relacionadas entre ellas, como el *Error Cuadrático Medio* (ECM) y *Peak to Signal Noise Ratio* (PSNR), denotados por $\texttt{ECM}(F, \bar{F})$ y $\texttt{PSNR}(F, \bar{F})$, respectivamente, y definidos como:

$$\texttt{ECM}(F, \bar{F}) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} |F_{k_{ij}} - \bar{F}_{k_{ij}}|^2 \tag{1}$$

y

$$\texttt{PSNR}(F, \bar{F}) = 10 \log_{10} \left(\frac{255^2}{\texttt{ECM}(F, \bar{F})} \right). \tag{2}$$

Donde m es la cantidad de filas de píxeles en cada imagen y n es la cantidad de columnas. Esta métrica puede extenderse para todo el video.

En conjunto con los valores obtenidos para estas métricas, es importante además realizar un análisis del tiempo de ejecución de cada método y los denominados *artifacts* que produce cada uno de ellos. Se denominan *artifacts* a aquellos errores visuales resultantes de la aplicación de un método o técnica. La búsqueda de este tipo de errores complementa el estudio cuantitativo mencionado anteriormente incorporando un análisis cualitativo (y eventualmente subjetivo) sobre las imágenes generadas.

B.3. Enunciado

Se pide implementar un programa en `C` o `C++` que implemente como mínimo los tres métodos mencionados anteriormente y que dado un video y una cantidad de cuadros a agregar aplique estas técnicas para generar un video de cámara lenta. A su vez, es necesario explicar en detalle cómo se utilizan y aplican los métodos descriptos en 1, 2 y 3 (y todos aquellos otros métodos que decidan considerar opcionalmente) en el contexto propuesto. Los grupos deben a su vez plantear, describir y realizar de forma adecuada los experimentos que consideren pertinentes para la evaluación de los métodos, justificando debidamente las decisiones tomadas y analizando en detalle los resultados obtenidos así como también plantear qué pruebas realizaron para convencerse de que los métodos funcionan correctamente.

B.4. Programa y formato de entrada

Se deberán entregar los archivos fuentes que contengan la resolución del trabajo práctico. El ejecutable tomará cuatro parámetros por línea de comando que serán el archivo de entrada, el archivo de salida, el método a ejecutar (0 para vecinos más cercanos, 1 para lineal, 2 para splines y otros números si consideran más métodos) y la cantidad de cuadros a agregar entre cada par del video original.

Tanto el archivo de entrada como el de salida tendrán la siguiente estructura:

- En la primera línea está la cantidad de cuadros que tiene el video (`c`).

- En la segunda línea está el tamaño del cuadro donde el primer número es la cantidad de filas y el segundo es la cantidad de columnas (`height width`).
- En la tercera línea está el framerate del video (`f`).
- A partir de allí siguen las imágenes del video una después de la otra en forma de matriz. Las primeras `height` líneas son las filas de la primera imagen donde cada una tiene `width` números correspondientes a los valores de cada píxel en esa fila. Luego siguen las filas de la siguiente imagen y así sucesivamente.

Además se presentan herramientas en Matlab para transformar videos (la herramienta fue probada con la extensión .avi pero es posible que funcione para otras) en archivos de entrada para el enunciado y archivos de salida en videos para poder observar el resultado visualmente. También se recomienda leer el archivo de README sobre la utilización.

Sobre la entrega

- FORMATO ELECTRÓNICO: Martes 10 de Noviembre de 2015, **hasta las 23:59**, enviando el trabajo (`informe` + `código`) a `metnum.lab@gmail.com`. El `asunto` del email debe comenzar con el texto `[TP3]` seguido de la lista de apellidos de los integrantes del grupo. Ejemplo: `[TP3] Artuso, Belloli, Landini`
- FORMATO FÍSICO: Miércoles 11 de Noviembre de 2015, en la clase práctica.

Referencias

[1] Richard L. Burden y J. Douglas Faires. *Numerical Analysis*. Ninth Edition. California: Brooks/Cole Publishing Company, 2010.