DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Usando Interpolación para generar videos slow-motion

Métodos Numéricos

Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Alvarez, Lautaro Leonel	268/14	lautarolalvarez@gmail.com
Maddonni, Axel Ezequiel	200/14	axel.maddonni@gmail.com
Thibeault, Gabriel Eric	114/13	gabriel.eric.thibeault@gmail.com

Presentamos tres métodos de interpolación para generar en videos el efecto de cámara lenta agregando frames intermedios. El primero, copia los frames consecutivos al que queremos agregar, generando efectos parecidos a un `gif`, mientras que los otros dos usan interpolación lineal y cúbica respectivamente, produciendo efectos más semejantes a cámara lenta pero a su vez, produciendo exploits en las salidas.

interpolación, splines, slow motion

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://dc.uba.ar

Índice

1. Introducción Teórica

Hoy en día los videos en cámara lenta o *slow motion* son un éxito. Muchas páginas web se dedican exclusivamente a compartir estos videos. Pero teniendo en cuenta que las conexiones a internet no necesariamente son capaces de transportar la cantidad de datos que implican este tipo de videos, se trata de minimizar la dependencia de la velocidad de conexión solamente enviando el video original. Una vez que el usuario recibe esos datos, el trabajo de la cámara lenta puede hacerse en modo offline del lado del cliente. Para tal fin utilizaremos técnicas de interpolación, buscando generar, entre cada par de cuadros del video original, otros ficticios que ayuden a generar el efecto de slow motion.

Consideraremos a un video un conjunto de cuadros o *frames* que al reproducirse rápidamente transmite el efecto de movimiento a partir de tener un buen *frame rate*, es decir, una alta cantidad de cuadros por segundo. Para producir el efecto de cámara lenta sobre un video, generaremos más cuadros entre cada par de cuadros consecutivos del video original y reproduciremos el resultado a la misma velocidad que el original. Las imágenes correspondientes a cada cuadro están conformadas por píxeles. En particular, 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.

Lo que haremos será interpolar en el tiempo los valores de los pixeles para cada posición de la imagen, generando así una función interpolante, que permita la obtención de nuevos frames partiendo del conocimiento de un conjunto discreto de puntos. Se proponen tres métodos de interpolación explicados a continuación: Vecino más cercano, Interpolación Lineal e Interpolación por Splines.

1.1. Vecino más cercano

El método de interpolación del Vecino mas Cercano se basa en tomar los valores del frame orginal mas cercano a la posición del nuevo frame y asignárselos.

Por ejemplo, si queremos agregar un frame B en la posición x_k con $x_i < x_k < x_{k+1}$, asumiendo que contamos con un frame A que se encuentra en la posición x_i y un frame C que se encuentra en la posición x_{i+1} y que C y B son los vecinos mas cercanos de A, entonces para determinar qué frame vamos a copiar, tomamos dist(x_i, x_k) y dist(x_k, x_{i+1}). Luego tomamos A o C dependiendo de cuáñ se encuentra mas cerca. Finalmente copiamos los píxeles del vecino mas cercano a B, obteniendo así el nuevo frame.

1.2. Interpolación Lineal

La interpolación lineal se basa en encontrar la recta que une dos puntos, y evaluarla en los puntos que se desea interpolar.

Considerando puntos de la forma $(x, f(x))$, debemos asignarle a los píxeles valores de x para poder interpolarlos ($f(x)$ son los valores de los puntos en la recta). Los valores específicos de x son irrelevantes, mientras sean equidistantes; en nuestro caso, para el i-ésimo frame intermedio, $x = i$. A su vez, para los frames que usamos para interpolar (que pertenecen al video original), $x = 0$ para el primero y $x = cantidad\ de\ frames\ intermedios + 1$ para el segundo.
La ecuación que utilizamos para la interpolación lineal es la siguiente:

$$f(x) = (y_2 - y_1)/(x_2 - x_1) * x + y_1 \tag{1}$$

Con x_1, x_2 los valores en x de los dos frames a partir de los cuales estamos interpolando, e $y_1 = f(x_1)$, $y_2 = f(x_2)$.

1.3. Interpolación por Splines

Para interpolar los frames de nuestro video, consideraremos una tabla de puntos $[x_i,\ y_i]$ con $i = 0, 1, ..., n$ por cada pixel de un frame del tamaño ingresado por parámetro, con $n = \#frames\ originales$. En nuestro modelo cada x_i representará el número de frame correspondiente, mientras que el y_i representará el valor del píxel en dicho frame. Entonces computaremos un spline por cada píxel que luego usaremos para formar los frames intermedios que buscamos agregar al video.

Cada x_i tomará en cuenta los frames intermedios, por ejemplo, si tomamos una entrada con framesIntermedios = 2 y cantidad de frames originales = 2, $x_0 = 0$(frame inicial), $x_1 = 3$ (frame final), y luego con el spline obtenido con el algoritmo calcularemos los valores intermedios tomando $x = 1$ y $x = 2$ respectivamente. Otra opción sería tomar $x_0 = 0$ y $x_1 = 1$, y calcular los frames Intermedios tomando $x = 1/3$ y $x = 2/3$, pero para trabajar con valores enteros y evitar perder precisión, usaremos la primera alternativa.

$$x_i = framesIntermedios * i + i$$

A continuación se explica el método de interpolación por Splines o Trazados Cúbicos para una tabla de puntos $[x_i,\ y_i]$, es decir, en nuestro modelo, un pixel por cada frame ubicados en la misma posición. El algoritmo presentado computa los splines para los demás pixels de manera análoga. En la sección Desarrollo se especifican cuestiones sobre el funcionamiento del algoritmo y la paralelización deĺ cálculo de los splines.

Como se menciona anteriormente, comenzamos con una tabla con $n + 1$ puntos y n intervalos entre ellos. Un spline de interpolación cúbica es una curva continua definida por partes, que pasa por todos los valores de la tabla. Hay un polinomio cúbico por cada intervalo, cada uno con sus coeficientes de la siguiente manera:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \quad \forall x \ \in \ [x_i, x_{i+1}]$$

Estos polinomios juntos se denominan spline $S(x)$.

Como hay n intervalos y 4 coeficientes para cada polinomio, requerimos $4n$ parametros para poder definir el spline, es decir, es necesario encontrar $4n$ condiciones independientes que cumpla el spline. Obtenemos 2 condiciones por cada intervalo para que el polinomio fitee los valores de la tabla en ambos extremos del intervalo y resulte interpolante:

$$S_i(x_i) = y_i \quad \forall i = 0, ..., n - 1 \tag{cond1}$$

$$S_i(x_{i+1}) = y_{i+1} \quad \forall i = 0, ...n - 1 \tag{cond2}$$

Notar que estas condiciones resultan en que la función definida por el spline sea continua. Todavía necesitamos $2n$ condiciones. Pedimos que las derivadas primeras y segundas de la función también sean continuas:

$$S'_{i+1}(x_{i+1}) = S'_i(x_{i+1}) \quad \forall i = 0, ..., n - 2 \tag{cond3}$$

$$S''_{i+1}(x_{i+1}) = S''_i(x_{i+1}) \quad \forall i = 0, ..., n - 2 \tag{cond4}$$

Estas últimas dos condiciones aplican para $i = 0, ..., n-2$, resultando en 2(n-1) ecuaciones. Por lo que necesitamos dos condiciones más para completar el spline. Existen dos opciones:

- Spline natural : $S''_0(x_0) = S''_{n-1}(x_n) = 0$
- Spline sujeto : $S'_0(x_0) = f'(x_0)\ y\ S'_{n-1}(x_n) = f'(x_n)$

Con estas $4n$ ecuaciones, es posible reducir las condiciones a un sistema triangular con los coeficientes c_i como las incógnitas. Procedemos a reducir las ecuaciones:

De la condición (cond1):

$$\begin{aligned} S(x_i) &= f(x_i) && \forall i = 0, ..., n - 1 \\ a_i + b_i(x_i - x_i) + c_i(x_i - x_i)^2 + d_i(x_i - x_i)^3 &= f(x_i) && \forall i = 0, ..., n - 1 \\ \boxed{a_i = f(x_i)} & && \forall i = 0, ..., n - 1 \end{aligned}$$

Expresemos (cond2) restando en 1 los subíndices:

$$y_i = S_{i-1}(x_i) \qquad \forall i = 1, ..., n$$
$$y_i = a_{i-1} + b_{i-1}h_{i-1} + c_{i-1}h_{i-1}^2 + d_{i-1}h_{i-1}^3 \qquad \forall i = 1, ..., n$$

Como $a_{i-1} = y_{i-1}$ $\forall i = 1, ..., n$ por (1):

$$\boxed{y_i = y_{i-1} + b_{i-1}h_{i-1} + c_{i-1}h_{i-1}^2 + d_{i-1}h_{i-1}^3} \qquad \forall i = 1, ..., n$$

donde $h_{i-1} = (x_i - x_{i-1})$

Calculamos la derivada de un spline:

$$S_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2 \qquad \forall i = 0, ..., n-1$$

Tomando $x = x_i$:

$$S_i'(x_i) = b_i \qquad \forall i = 0, ..., n-1$$

Expresamos (cond3) restando en 1 los subíndices :

$$S_{i-1}'(x_i) = S_i'(x_i) \qquad \forall i = 1, ..., n-1$$

Combinándolas tenemos:

$$b_i = S_i'(x_i) = S_{i-1}'(x_i) = b_{i-1} + 2c_{i-1}h_{i-1} + 3d_{i-1}h_{i-1}^2 \qquad \forall i = 1, ..., n-1$$

Tomando $b_n = S_{n-1}'(x_n) = b_{n-1} + 2c_{n-1}h_{n-1} + 3d_{n-1}h_{n-1}^2$

$$\boxed{b_i = b_{i-1} + 2c_{i-1}h_{i-1} + 3d_{i-1}h_{i-1}^2} \qquad \forall i = 1, ..., n$$

donde $h_{i-1} = (x_i - x_{i-1})$

Ahora calculamos la segunda derivada:

$$S_i''(x) = 2c_i + 6d_i(x - x_i) \qquad \forall i = 0, ..., n-1$$
$$S_i''(x_i) = 2c_i \qquad \forall i = 0, ..., n-1$$

Expresamos (cond4) restando en 1 los subíndices:

$$S_{i-1}''(x_i) = S_i''(x_i) \qquad \forall i = 1, ..., n-1$$

Combinándolas tenemos:

$$2c_i = S_i''(x_i) = S_{i-1}''(x_i) = 2c_{i-1} + 6d_{i-1}h_{i-1} \qquad \forall i = 1, ..., n-1$$

Tomando $c_n = S_{n-1}''(x_n) = 2c_{n-1} + 6d_{n-1}h_{n-1}$

$$\boxed{2c_i = 2c_{i-1} + 6d_{i-1}h_{i-1}} \qquad \forall i = 1, ..., n$$

donde $h_{i-1} = (x_i - x_{i-1})$

Ahora podemos armar el sistema para calcular los c_i. Los siguientes cálculos permiten expresar los c en términos de h e y, eliminándo las b y d de las ecuaciones:

Tenemos:

$$y_i = y_{i-1} + b_{i-1}h_{i-1} + c_{i-1}h_{i-1}^2 + d_{i-1}h_{i-1}^3 \qquad \forall i = 1, ..., n \qquad \text{(eq1)}$$
$$b_i = b_{i-1} + 2c_{i-1}h_{i-1} + 3d_{i-1}h_{i-1}^2 \qquad \forall i = 1, ..., n \qquad \text{(eq2)}$$
$$2c_i = 2c_{i-1} + 6d_{i-1}h_{i-1} \qquad \forall i = 1, ..., n \qquad \text{(eq3)}$$

Despejamos b_{i-1} de (eq1):

$$b_{i-1} = \frac{y_i - y_{i-1}}{h_{i-1}} - c_{i-1}h_{i-1} - d_{i-1}h_{i-1}^2 \qquad \forall i = 1, ..., n \qquad \text{(eq4)}$$
$$b_i = \frac{y_{i+1} - y_i}{h_i} - c_ih_i - d_ih_i^2 \qquad \forall i = 0, ..., n-1 \qquad \text{(eq5)}$$

Escribimos (eq3) como:

$$2c_i - 2c_{i-1} = 6d_{i-1}h_{i-1} \qquad \forall i = 1, ..., n$$

multiplicamos por h_{i-1} y dividimos por 2 ambos lados:

$$h_{i-1}(c_i - c_{i-1}) = 3d_{i-1}h_{i-1}^2 \qquad \forall i = 1, ..., n \tag{eq6}$$

De la (eq2) podemos despejar

$$3d_{i-1}h_{i-1}^2 = b_i - b_{i-1} - 2c_{i-1}h_{i-1} \qquad \forall i = 1, ..., n \tag{eq7}$$

Reemplazamos (eq7) en el lado derecho de (eq6):

$$h_{i-1}(c_i - c_{i-1}) = b_i - b_{i-1} - 2c_{i-1}h_{i-1} \qquad \forall i = 1, ..., n$$

$$2c_{i-1}h_{i-1} + h_{i-1}(c_i - c_{i-1}) = (b_i - b_{i-1}) \qquad \forall i = 1, ..., n$$

$$h_{i-1}(c_i + c_{i-1}) = (b_i - b_{i-1}) \qquad \forall i = 1, ..., n \tag{eq8}$$

Reemplazamos (eq4) y (eq5) en (eq8):

$$h_{i-1}(c_i + c_{i-1}) = (\frac{y_{i+1} - y_i}{h_i} - c_i h_i - d_i h_i^2) - (\frac{y_i - y_{i-1}}{h_{i-1}} - c_{i-1}h_{i-1} - d_{i-1}h_{i-1}^2)$$

$$\forall i = 1, ..., n-1$$

$$c_i h_i - c_{i-1}h_{i-1} + h_{i-1}(c_i + c_{i-1}) = (\frac{y_{i+1} - y_i}{h_i} - d_i h_i^2) - (\frac{y_i - y_{i-1}}{h_{i-1}} - d_{i-1}h_{i-1}^2)$$

$$\forall i = 1, ..., n-1$$

$$c_i h_i + c_i h_{i-1} = (\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}) + (d_{i-1}h_{i-1}^2 - d_i h_i^2)$$

$$\forall i = 1, ..., n-1 \tag{eq9}$$

De (eq6) podemos despejar:

$$d_{i-1}h_{i-1}^2 = \frac{1}{3}h_{i-1}(c_i - c_{i-1}) \qquad \forall i = 1, ..., n \tag{eq10}$$

$$d_i h_i^2 = \frac{1}{3}h_i(c_{i+1} - c_i) \qquad \forall i = 0, ..., n-1 \tag{eq11}$$

Reemplazo (eq10) y (eq11) en (eq9):

$$c_i h_i + c_i h_{i-1} = (\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}) + \frac{1}{3}\Big((h_{i-1}(c_i - c_{i-1}) - h_i(c_{i+1} - c_i)\Big) \forall i = 1, ..., n-1$$

multiplicamos por 3 en ambos lados y movemos los términos con c a la izquierda

$$3c_i h_i + 3c_i h_{i-1} - \Big(h_{i-1}(c_i - c_{i-1}) - h_i(c_{i+1} - c_i)\Big) = 3\Big(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\Big) \forall i = 1, ..., n-1$$

$$c_{i-1}h_{i-1} + 2c_i h_{i-1} + 2c_i h_i + c_{i+1}h_i = 3\Big(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\Big) \forall i = 1, ..., n-1$$

$$\boxed{c_{i-1}h_{i-1} + 2c_i(h_{i-1} + h_i) + c_{i+1}h_i = 3\Big(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\Big)} \tag{eq12}$$

$$\forall i = 1, ..., n-1$$

Podemos escribir el sistema de ecuaciones resultante para calcular los c_i usando la siguiente matriz tridiagonal:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ h0 & 2(h_0+h_1) & h_1 & & & \\ 0 & h_1 & 2(h_1+h_2) & h_2 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} & 0 \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{bmatrix} = 3 \begin{bmatrix} 0 \\ \frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0} \\ \vdots \\ \vdots \\ \vdots \\ \frac{y_n-y_{n-1}}{h_{n-1}} - \frac{y_{n-1}-y_{n-2}}{h_{n-2}} \\ 0 \end{bmatrix}$$

donde la primera y última de la matriz corresponden a las condiciones generadas por el hecho de que el spline sea Libre. Un spline libre cumplía:

$$\begin{aligned} S''(x_0) = S_0''(x_0) &= 0 \\ 2c_0 + 6d_0(x_0 - x_0) &= 0 \\ c_0 &= 0 \qquad \text{Primera fila} \end{aligned}$$

$$\begin{aligned} S''(x_n) = S_{n-1}''(x_n) &= 0 \\ \text{pero } S_{n-1}''(x_n) &= c_n \\ \Rightarrow c_n &= 0 \qquad \text{Última fila} \end{aligned}$$

Una vez obtenidos los c_i, podemos calcular los b_i usando la (eq5) y los d_i usando (eq3). La forma en que resolvemos el sistema está detallada en la sección Desarrollo. Notar que la matriz resultante es estrictamente diagonal dominante, por lo que es inversible y tiene solución.

2. Desarrollo

2.1. Vecino más cercano

Para comenzar con el desarrollo del algoritmo de Vecinos mas Cercanos, nos planteamos la idea principal del algoritmo y qué decisión tomaremos en caso de que un frame con igual distancia a ambos vecinos. En esa circunstancia le daremos el valor de su siguiente vecino.

El algoritmo se basa en un ciclo que itera sobre la cantidad de frames del video original. En cada iteración lo que hacemos es tomar el frame previo e imprimirlo en el archivo de salida $F/2$ veces, siendo $F =$ "cantidad de frames intermedios que se quieren agregar". Con esto logramos que los vecinos que se encuentran de la mitad hacia el lado del vecino inferior tomen su valor. Mientras que luego imprimimos otros $F/2$ frames pero con los valores del vecino superior. En caso de que $F/2$ no sea un valor entero, tomaremos ese valor central como mas cercano al superior.

2.2. Interpolación Lineal

Para interpolación lineal, tomamos dos frames de referencia, y encontramos, pixel por pixel, la recta que une los dos valores. Para generar los frames intermedios, tomamos los valores de cada pixel según las rectas encontradas.

2.3. Interpolación por Splines

El algoritmo de splines genera un spline por cada posición de un pixel de un frame, de manera que para generar un nuevo frame intermedio se utilizan todos los splines generados. A su vez, se divide la cantidad de frames del video en bloques. Esto se hace ya que no tiene sentido calcular los splines tomando todos los frames del video ya que pueden ser muy diferentes o pertenecer a distintas tomas, por ejemplo. Además cabe destacar que al aumentar el tamaño de los bloques aumenta la complejidad espacial del algoritmo, ya que necesita el valor de los píxeles de todos los frames del bloque para poder calcular las incógnitas. La cantidad mínima de frames por bloques es 4 ya que se necesita de 4 puntos como mínimo para poder calcular el polinomio cúbico.

El algoritmo entonces, itera sobre cada uno de los bloques en los que se divide el video (según el tamaño de bloque ingresado por parámetro, que debe ser mayor o igual a 4) para calcular los splines correspondientes. En cada iteración, se calcula la matriz para calcular los c_i y el vector res para cada posición $[i, j]$ de los frames, como fue explicado en la sección de introducción teórica. Como fue mencionado anteriormente, esta matriz es estrictamente diagonal dominante y por lo tanto inversible, por lo que se puede calcular la solución al sistema sin problemas. Para calcularla, el algoritmo primero diagonaliza la matriz aprovechando el hecho de que sea una matriz tridiagonal y luego simplemente calcula los c_i despejando los valores.

Una vez obtenidos los c_i para cada spline, se calculan los d_i y los b_i y se calculan los valores de los píxeles de los nuevos frames intermedios usando los splines generados. Para eso, se evalúan los polinomios generados en los x correspondientes a la distancia entre el nuevo frame y el x_i anterior.

2.4. Planteo de Experimentos

Los experimentos realizados se pueden dividir en tres categorías:

- *Tiempo de Ejecución*: experimentos que buscan comparar la complejidad temporal de los algoritmos y ver cómo influyen los parámetros de entrada.
- *Análisis Cuantitativo*: experimentos que comparan los videos obtenidos por cada algoritmo, utilizando métricas comunes como lo son ECM (*Error Cuadrático Medio)* y PSNR *(Peak to Signal Noise Ratio)* para medir la calidad de los resultados.
- *Análisis Cualitativo*: análisis más subjetivo de los videos generados, comparando los errores visuales resultantes producidos por cada método.

3. Resultados y Discusión

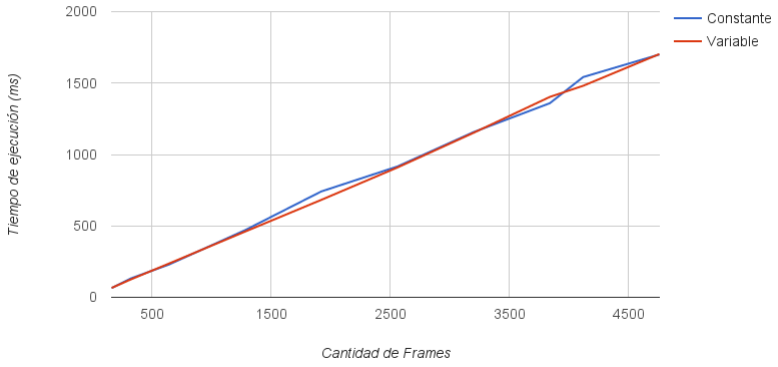
3.1. Tiempo de Ejecución

3.1.1. Experimento 1: Efecto de la variación de colores

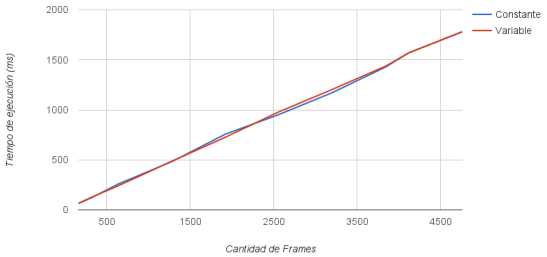
Para empezar con la experimentación relacionada a los tiempos de ejecución de los distintos métodos correremos los distintos métodos sobre un video con colores constantes y otro video con colores variables y compararemos los resultados obtenidos. Esto nos va a servir a la hora de llevar a cabo otros experimentos, porque si los tiempos son similares entre un video y otro significa que los colores que elijamos para los distintos videos no van a influirnos en los tiempos de ejecución.

Vamos a crear un video que tiene resolución de 10px x 10px con un color fijo constante y otro video, de la misma resolución, pero con colores que oscilan de forma ordenada entre el blanco, el negro y sus tonos intermedios. Tomaremos bloques de 5 frames (podríamos haber elegido cualquier número) y agregaremos 2 frames intermedios entre cada frame del video de entrada.

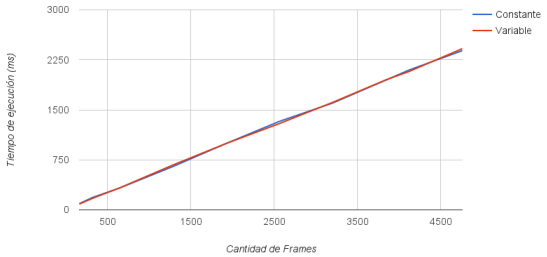
3.1.1.1 Resultados del experimento 1



(a) Método del Vecino mas cercano



(b) Método Lineal



(c) Método de Splines naturales por bloques

Como podemos observar en los gráficos, los métodos se comportan de la misma forma con ambas imágenes. Se notan diferencias en algunos momentos del método del vecino mas cercano, pero manteniendo el mismo orden. No nos van a preocupar mucho estas diferencias, pero si las tendremos en cuenta de aquí en adelante.

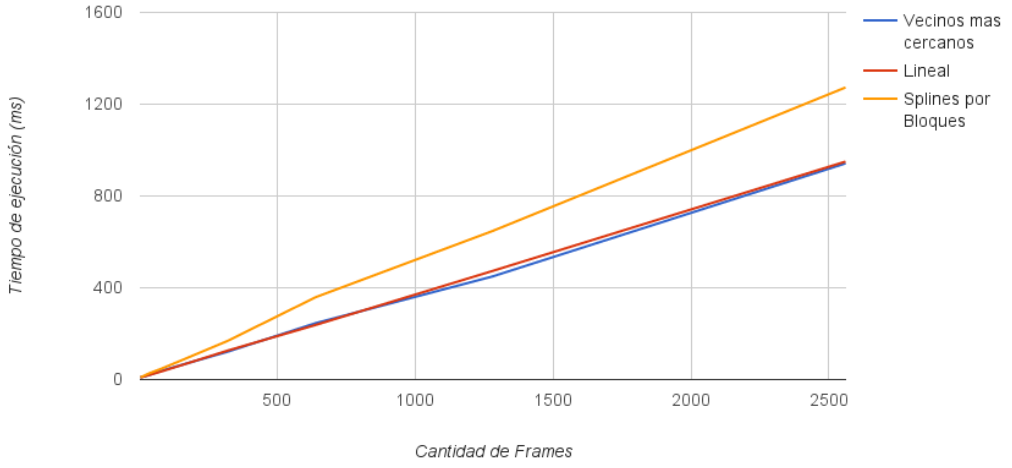
Luego de este experimento podemos asumir que el cambio de colores prácticamente no va a influir en los tiempos de ejecución. Ahora sí' podemos continuar con la experimentación teniendo en claro que ocurre con los colores.

3.1.2. Experimento 2: Tiempos de ejecucíón vs Cantidad de Frames

En este experimento vamos a comparar el tiempo de ejecución de los diferentes métodos al aumentar la cantidad de frames de un video y tratar de determinar el orden de los mismos. Esperábamos que el orden del método de Splines por bloques sea mayor, pero eniendo en cuenta lo visto en el Experimento 1 estamos mas cerca de creer que se comportan de forma lineal. Igualmente queremos llevar a cabo este experimento para abarcar cantidades mayores y sacarnos las dudas.

Para llevar a cabo el experimento vamos a crear videos con patrones de imágenes repetidos (tomando en consideración esos pequeños cambios que vimos en el experimento anterior) manteniendo el mismo ancho y alto de imágenes y cantidad de frames en aumento. Correremos los métodos con un valor de frames intermedios fijo para enfocarnos solo en la cantidad de frames.

3.1.2.1 Resultados del experimento 2



(d) Tiempos de ejecución variando la cantidad de frames

Luego de realizar el experimento podemos creer que el tiempo de ejecución de los tres métodos es de orden lineal. Podemos notar que los Splines por Bloques se mantiene siempre por encima de los otros, pero mantienen su forma lineal. Mas adelante veremos las diferencias de calidad entre los métodos y tendremos que tener en cuenta esta igualdad de orden, ya que no es tan grande como esperábamos en un principio.

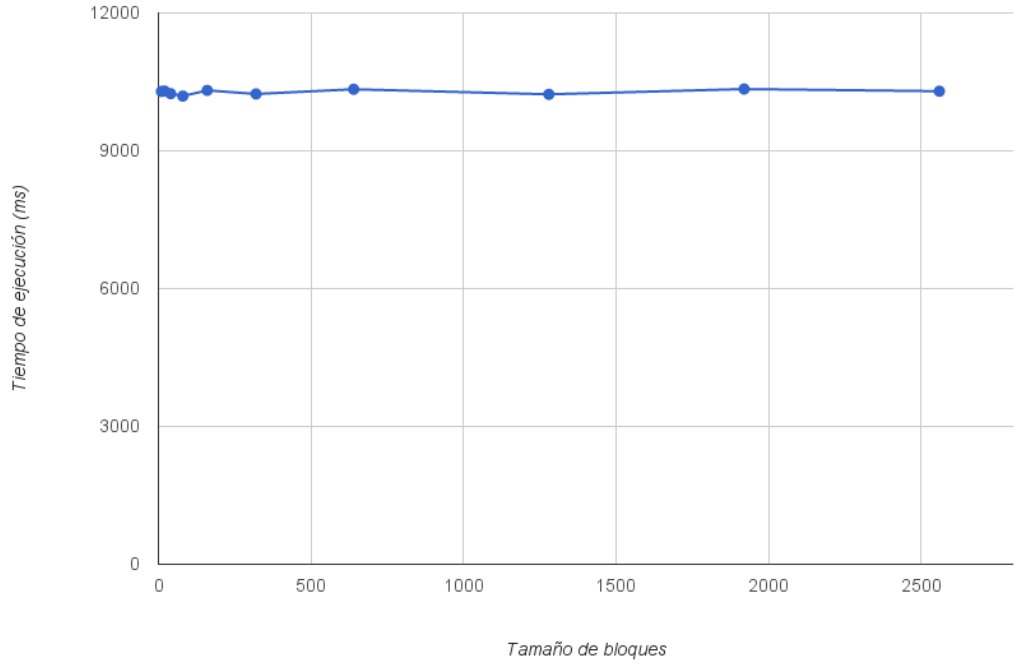
3.1.3. Experimento 3: Tiempos de ejecucíón vs Tamaño de Bloques

Como tomamos la determinación de aplicar el método de Splines de a bloques de frames, queremos medir cuanto nos afecta el tamaño que decidamos darle a estos bloques en el tiempo de ejecución del método.

Para esto vamos a tomar un video con 20480 frames, y vamos a comenzar a aumentar el tamaño de los bloques hasta llegar a un solo bloque que abarque todos el video.

3.1.3.1 Resultados del experimento 3

Podemos observar que a pesar de algunas diferencias el orden del tiempo se mantiene constante y no se vió afectado por el aumento del tamaño de bloques. Nos parece normal que esto ocurra, ya que el algoritmo realiza la misma cantidad de pasos. La diferencia se ve en como se agrupan los datos, lo que si puede influir en las imágenes decididas a agregar como nuevos frames.

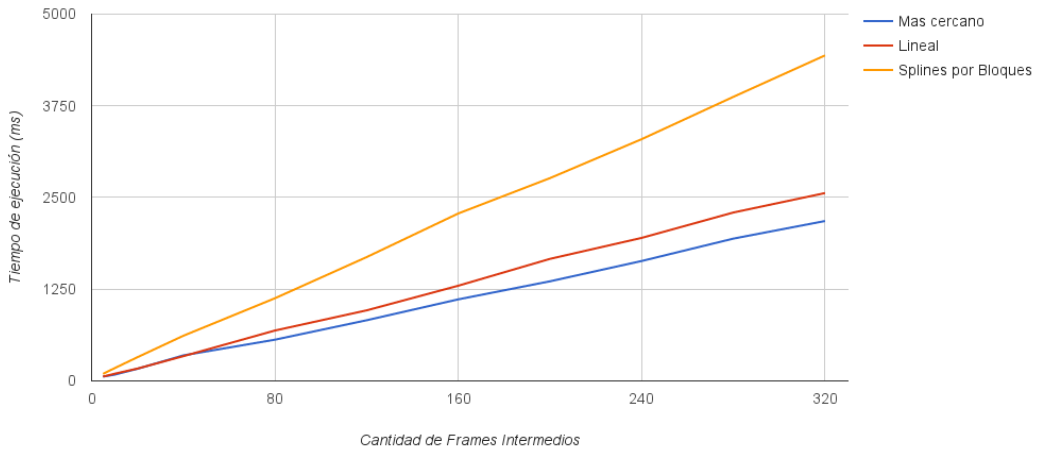


(e) Tiempos de ejecución variando el tamaño de los bloques

3.1.4. Experimento 4: Tiempos de ejecucíón vs Cantidad de frames intermedios

Vamos a observar como se comportan los métodos al variar la cantidad de frames que se desean agregar. A pesar de que no se hacen muchos pasos extra para agregar un frame intermedio y solo se realiza una ecuación, esta ecuación se repite para cada pixel del nuevo frame. Creemos que esto va a afectar mucho en el tiempo de ejecución de los tres métodos.

3.1.4.1 Resultados del experimento 4



(f) Tiempos de ejecución variando la cantidad de frames intermedios

Tras realizar el experimento notamos que el aumento de la cantidad de frames intermedios se relaciona de forma lineal con el tiempo de ejecución. Analicemos un poco mas la situación:

Supangamos que tomamos N como la cantidad de frames del video original y F como cantidad de frames

intermedios a agregar. De esta forma al aplicar alguno de los métodos se agregan F frames N-1 veces $\longrightarrow$ F*(N-1). Si aplicamos uno de los método nuevamente, pero esta vez con F+1 frames intermedios, se agregarán F+1 frames N-1 veces $\longrightarrow$ (F+1)*(N-1) = F*(N-1) + (N-1). Con este razonamiento simple y sin entrar en detalles podemos convencernos de que al aumentar la cantidad de frames aumentamos de forma lineal la cantidad de operaciones a realizar (agregamos N-1 operaciones 'agregar frame intermedio').

3.2. Análisis Cuantitativo

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:

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

y

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

Ambas métricas se utilizan para medir la diferencia absoluta entre dos señales, en este caso entre los valores de los pixels del video de entrada y del de salida. Los algoritmos implementados para calcular ECM y PSNR calculan dichos valores frame a frame y luego hacen un promedio entre todos los frames del video. Utilizaremos estos promedios obtenidos para comparar la efectividad (en términos cuantitativos) de cada algoritmo, dependiendo de los parámetros de entrada. (El algoritmo para ECM puede encontrarse en el Apéndice 5.1.2).

Notar que en los casos en que el ECM entre dos frames sea 0, (debido a que dicho frame no se modificó), el PSNR da infinito. Para calcular el promedio del PSNR de un video, en estos casos, no se tomarán en cuenta dichos frames.

3.2.1. Experimentación con ECM y PSNR

Para calcular el ECM obtenido a partir de los videos generados con los algoritmos propuestos, implementamos tests que comparan frame a frame el video original ya en slow motion, con el video obtenido luego de eliminar frames intermedios y aplicar cada método para regenerar dichos frames. Es decir, si al video original le eliminamos 2 frames intermedios (utilizando la función `eliminarFrames`, ver Apéndice 5.1.1), compararemos el video original con el video obtenido luego de aplicar cada uno de los algoritmos, tomando como parámetro la misma cantidad de frames intermedios que le habíamos eliminado al video en un principio, en este caso 2.

Realizaremos esto para 6 videos diferentes, con distintas calidades y duraciones. Los videos elegidos como inputs se encuentran en slow motion originalmente, y muestran situaciones que consideramos interesantes para analizar resultados cuantitativos y cualitativos, como por ejemplo, una pelota de tenis volando o fideos cayendo lentamente, ya que se pueden apreciar las diferencias con los videos originales más claramente. Además, analizaremos los resultados según el tamaño y calidad del video, ya que algunos algoritmos podrían mejorar su rendimiento en dichos casos. Creemos que videos de mayor calidad generan resultados de mayor precisión.

Los tests que generamos calculan el ECM para dichos inputs variando la cantidad de framesIntermedios eliminados al original (2, 4 o 6 frames) para cada algoritmo. Para el algoritmo de splines, a su vez, realiza el mismo cálculo tomando distintos tamaños de bloque (4, 8 o 16 frames por bloque). Con esto pretendemos observar cómo afectan estos parámetros cuantitativamente al resultado final. Bloques de splines de menor tamaño supondrían una mayor precisión debido a que los videos cambian de tomas rápidamente, y al prolongar el tamaño del bloque permite interpolar frames tomando como parámetros frames pertenecientes a distintas tomas, y por lo tanto generar resultados no deseados como imágenes transparentes entre una toma y otra. Estos *artifacts* serán analizados con más detalle en la próxima sección de análisis cualitativo. Aquí nos limitaremos a analizar los resultados cuantitativos.

A continuación una breve descripción de los videos de entrada que utilizamos para los tests: (de menor a mayor calidad)

- *cupcake* : #frames = 252, tamaño 240x360
- *perro* : #frames = 252, tamaño 360x360
- *morocho* : #frames = 452, tamaño 360x360
- *tenis* : #frames = 450, tamaño 240x426

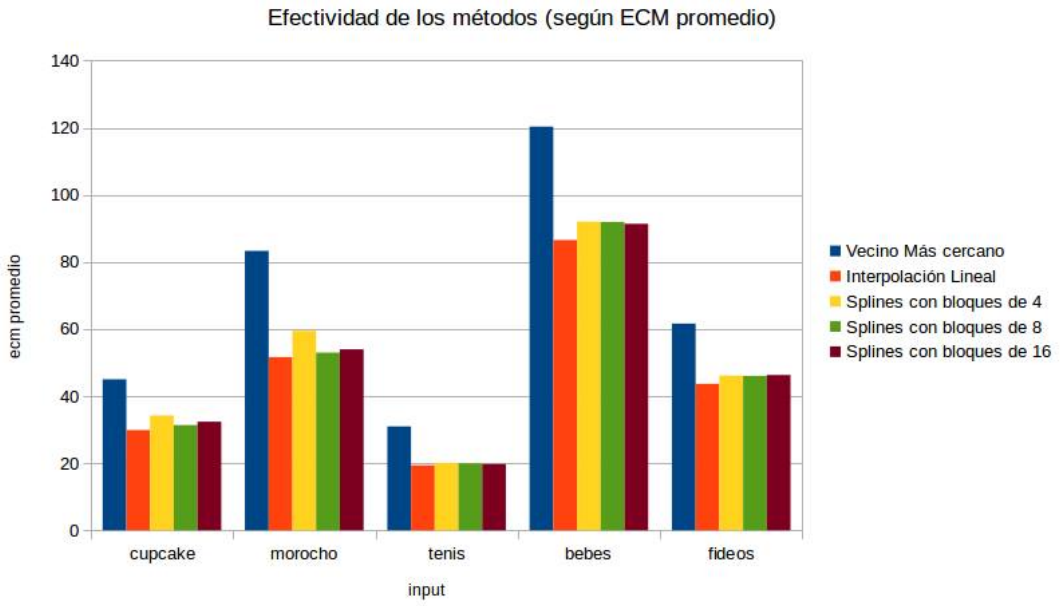
- *bebes* : #frames = 225, tamaño 544x1280
- *fideos* : #frames = 125, tamaño 720x1280

En el Apéndice 5.2 se puede encontrar una tabla con todos los valores obtenidos sobre el ECM promedio y los ECM máximos obtenidos por cada algoritmo para cada input. A continuación estudiaremos el comportamiento de cada método desde distintos enfoques.

3.2.2. Resultados sobre ECM y PSNR

3.2.2.1 Experimento 1: Efectividad de los métodos según ECM

Comparamos la efectividad de los métodos según el ECM promedio obtenido en cada uno de los inputs con 2 frames intermedios eliminados. Cuanto menor sea el error, consideraremos que el método es más efectivo cuantitativamente. Dejaremos afuera el input *perro* para que el gráfico de barras sea más claro, ya que son valores muy altos comparados a los demás.



(g) Efectividad de los métodos según ECM

Se puede observar en el gráfico que el método que arroja mejores resultados con respecto al error cuadrático medio es el de Interpolación Lineal, a diferencia de lo que creíamos. Supusimos que splines por el mayor grado de complejidad de los cálculos y del polinomio interpolante daría como resultado videos más aproximados al original (ideal). Sucede que el algoritmo de splines divide los frames en bloques de como mínimo 4 frames para poder calcular el polinomio de grado 3 (4 variables), lo que hace que en varios de esos bloques los frames pertenezcan a tomas diferentes del video. Si los frames son de tomas diferentes, o los frames varían demasiado entre uno y otro, la función interpolante resultante tiene más error. Como el método lineal "divide"los frames en bloques de a 2, ya que realiza una función cada 2 frames, el impacto de este fenómeno es menor.

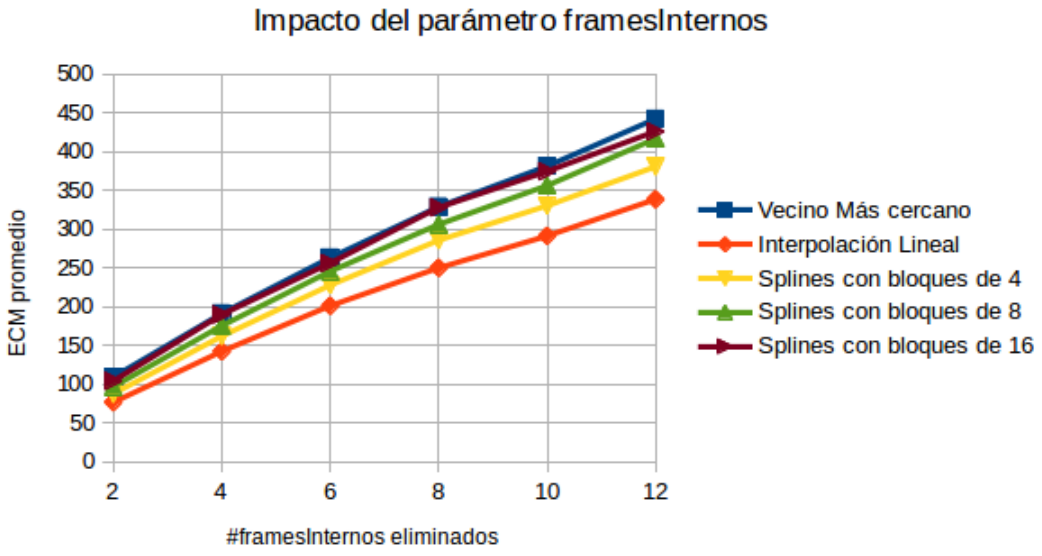
Además, se puede observar en el gráfico el impacto del **tamaño de bloque** tomado para realizar los splines. Variar el tamaño del bloque no modifica significativamente el error cometido, sin embargo, estimamos que al aumentar considerablemente el tamaño de los bloques se podrán observar errores visuales (más sobre esto en la sección Análisis Cualitativo).

Se puede observar que el tamaño de los videos y la calidad no impacta significativamente en el error cuadrático medio. Sin embargo, el video *perro* tiene mayores valores de error en todos los algoritmos (ver Apéndice), debido a que es el único de los videos que fue grabado con un celular en movimiento.

Por último, variar la calidad del video no trae grandes diferencias en términos cuantitativos, sin embargo, al observar los outputs, se nota una leve mejora del método de splines en términos cualitativos (se producen menos exploits).

3.2.2.2 Experimento 2: Impacto del parámetro Frames Internos eliminados

Usando los mismos inputs, correremos los métodos tomando distintas cantidades de frames internos eliminados y compararemos los resultados. Aumentar la cantidad de frames internos a regenerar, debería aumentar el error cuadrático medio.



(h) Efectividad de los métodos según frames intermedios eliminados

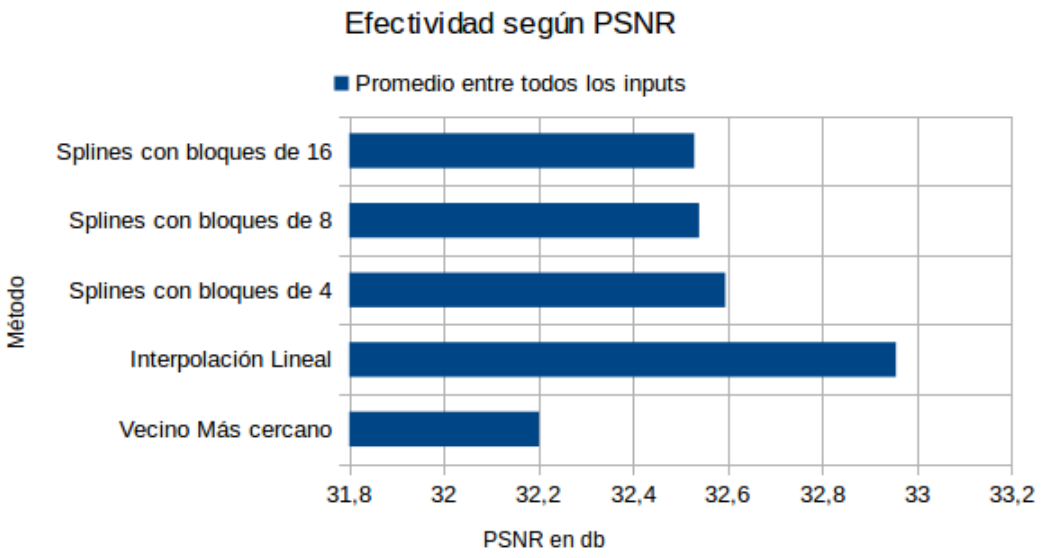
En el gráfico se muestran los valores obtenidos al calcular el promedio de los ECM de cada input generados por cada método, según cuál fue la cantidad de frames internos pasada por parámetro.

Efectivamente, aumentar framesIntermedios aumenta el error cuadrático medio. Se puede ver en el gráfico que lo hace linealmente, en todos los métodos. Esto se debe a que es mayor la cantidad de frames que se obtienen usando las funciones interpolantes, aumentando el error producido.

3.2.2.3 Experimento 3: Efectividad de los métodos según PSNR

El PSNR es la métrica utilizada con mayor frecuencia como medida de la calidad objetiva de video. Sin embargo, a causa del comportamiento nolineal del sistema visual humano los valores de PSNR no poseen una perfecta correlación con la calidad visual percibida.

Análogamente, calculamos los PSNR promedio obtenidos con cada algoritmo para los mismos inputs mencionados anteriormente:



(i) Efectividad de los métodos según PSNR

Como era de esperar, los resultados se corresponden con los obtenidos a partir de comparar los ECM producidos por cada algoritmo. Por definición de PSNR, cuanto menor sea el error, mayor será el PSNR y por lo tanto mejor la calidad.

Interpolación lineal es el método con mayor efectividad en términos cuantitativos.

3.3. Análisis Cualitativo

En esta sección realizaremos un análisis cualitativo de los métodos desarrollados. Con este fin, observaremos el resultado de aplicar los distintos métodos a ciertas entradas; buscaremos *artifacts* (errores visibles en el video causados por la interpolación), compararemos los diferentes procedimientos entre sí, y los efectos de distintos parámetros en cada uno.

3.3.1. Artifacts

Si bien encontrar una interpolación ideal puede no ser posible, se pueden generar frames intermedios que no sean completamentamente fidedignos a los idóneos, pero que son de todas formas verosímiles. Un objetivo quizás más plausible que una interpolación perfecta o ideal, es una de la cual el observador no puede discernir de los frames originales (al menos en una observación "normal"). Los *artifacts* son errores en la interpolación que tienen un efecto visible en el video, y que detraen de este objetivo. Los estudiaremos con especial atención ya que tienen un significativo efecto negativo en la percibida calidad del video resultante.

3.3.1.1 Vecino más cercano

La mayoría de los métodos de interpolación intentan "adivinar" los frames intermedios en base a los de referencia, lo que conlleva posibles errores. Vecino más cercano, por otro lado, genera a los intermedios copiando a los de referencia, por lo que no se pueden producir artifacts.

3.3.1.2 Lineal y Splines

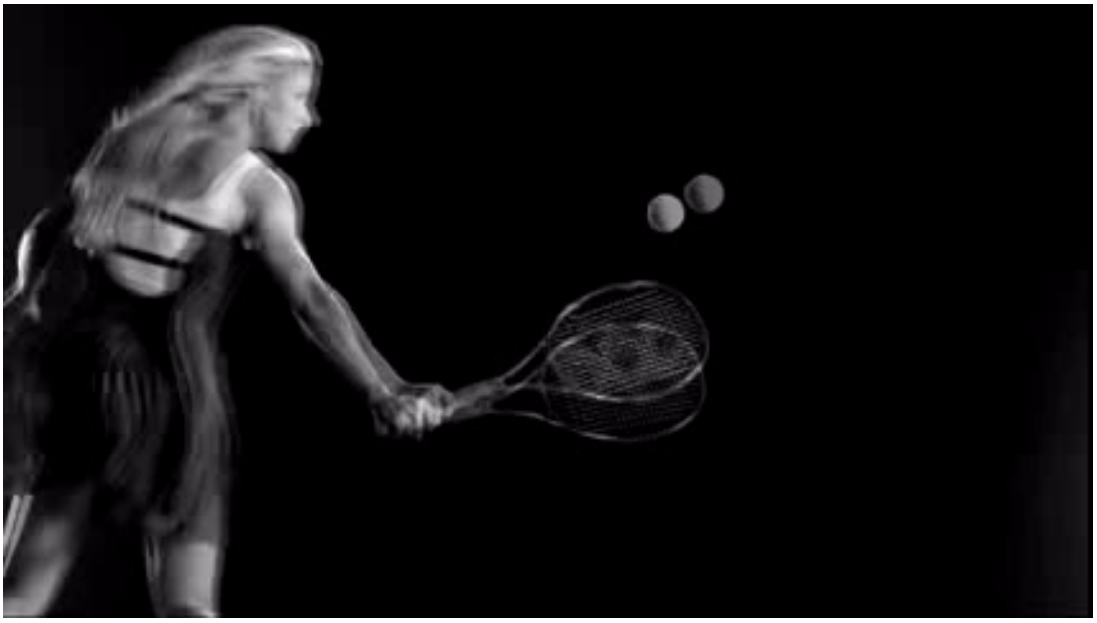
Ambos métodos funcionan de forma similar: viendo cómo varían los valores de los mismo píxeles (es decir, de los píxeles que están en la misma posición de cada frame) entre los frames de referencia, e interpolando los valores de los intermedios. Lo que varía es la familia de funciones mediante la cual aproximan, y cuántos frames de referencia utilizan. Debido a esto, ambos métodos generan "trazas" al interpolar movimiento. A continuación ahondaremos en esto, y daremos ejemplos.

3.3.1.3 Lineal

Si el video representa el movimiento (algo casi ubicuo en este tipo de aplicaciones) de algún cuerpo a lo largo de los frames, lo que este método hace es esencialmente un "fade-in" y "fade-out" del objeto del primer frame de referencia al segundo. Para clarificar lo anterior: se observa que el objeto en el primer frame se va transformando en el fondo (en la posición de dicho objeto) del segundo, mientras que el fondo (en la nueva posición del objeto) del primero se va transformando en el objeto del segundo.

En la realidad (a escala no-cuántica), el movimiento es continuo, por lo que idealmente la interpolación debería generar frames intermedios donde el cuerpo en movimiento se encuentra en los puntos intermedios de dicho movimiento. El método lineal genera movimiento discontinuo, donde el cuerpo se funde de una posición en un frame de referencia a la del próximo frame. Un claro ejemplo de estas trazas o "fade-in"-"fade-out" se puede observar en la imagen 1, que es un frame interpolado linealmente.

Figura 1: Traza en interpolacion lineal - video: Tenis con 6 frames intermedios



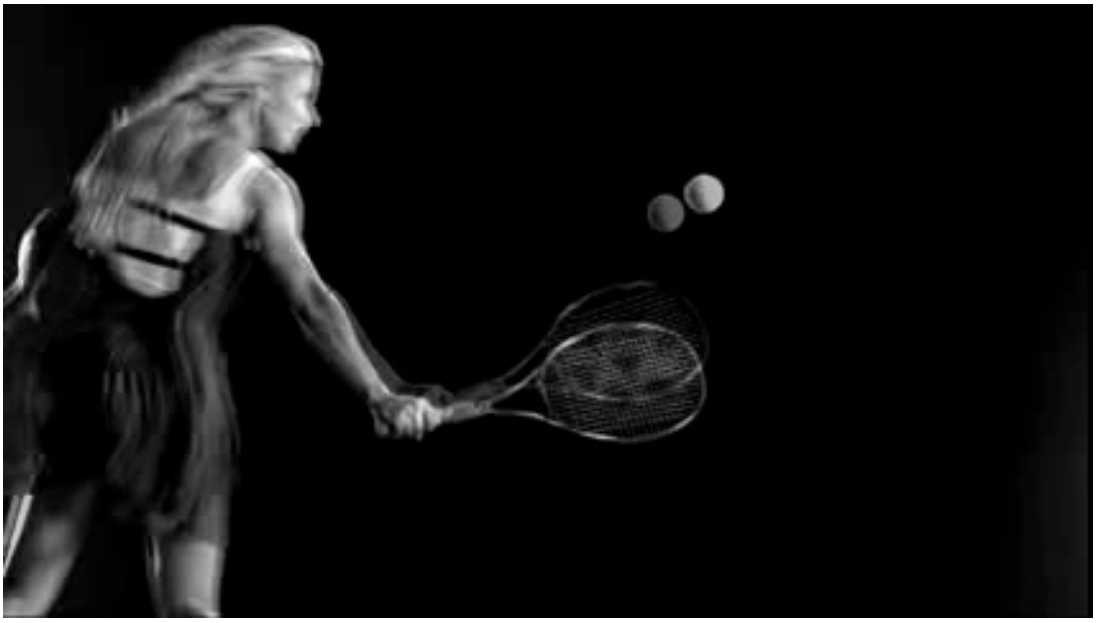
Si bien estas trazas técnicamente son artifacts, ya que no representan la realidad del video, cabe destacar que a los seres humanos éstas no nos parecen muy fuera de lugar o inverosímiles; en dibujo, efectos especiales y demás, una forma de representar movimiento es justamente mediante tales trazas; incluso nuestro propio sentido de visión en ciertas circunstancias captura el movimiento de tal forma (como cualquiera habrá hecho en algún momento, si uno mueve la mano rápidamente de un lado al otro con el brazo fijo se observan trazas en vez de un movimiento continuo - otro ejemplo es el clásico truco del lápiz que parece hecho de goma).

Debido a esto, las trazas pueden tener un impacto incluso positivo en el video en lo que concierne a interpolación del movimiento, en especial con framerate alto y pocos frames intermedios.

3.3.1.4 Splines

En el método de Splines se observan trazas similares a las de la interpolación lineal; la diferencia yace en cómo se ve el “fade-in”-“fade-out”. Un ejemplo de tales trazas, para el mismo video que el de la imagen 1, se encuentra en la figura 2.

Figura 2: Traza en Splines - video: Tenis con 6 frames intermedios, tamaño de bloque 4



Cabe destacar que las trazas en Splines se ven con mayor claridad al incrementar el tamaño de bloque. Se observa que esto detrae del efecto positivo mencionado previamente.

3.3.2. Cambio de Escena

Los componentes (en el sentido, de objetos, cuerpos, paisajes y demás) de un video pueden pasar de un frame al otro esencialmente de tres maneras: se pueden quedar quietos (en cuyo caso los tres métodos empleados las mantendrán iguales como deben, y por ende sin artifacts), se pueden mover de algún lugar del video al otro (los efectos de los métodos de interpolación en el movimiento fueron detallados en la sección previa), o pueden aparecer/desaparecer (o saltar de un lugar a otro de una forma discontinua).

Cuando desaparece toda la escena y aparece una enteramente nueva, se dice que hubo un cambio de escena. Éste es el caso que analizaremos, ya que en las imágenes resultantes se ve con mayor claridad los efectos de los distintos métodos de interpolación, pero todo lo que detallaremos en esta sección se aplica perfectamente en el caso de que sólo algunos componentes aparezcan o desaparezcan.

El caso de cambio de escena es distinto que el de movimiento (real, el ficticio que no necesariamente es continuo lo contemplamos como aparición/desaparición por lo que se detallará a continuación), ya que no necesariamente sabemos que es una transición continua. Es perfectamente posible que la transición de escenas sea un “fade-out” de una y un “fade-in” de otra, o que sea un corte limpio en el cual todo frame pertenece exclusivamente a una u otra (de hecho, ambas transiciones son usadas ampliamente en la cinematografía), o incluso mediante alguna otra técnica (como ejemplo, considerar las distintas formas de pasar de una diapositiva a otra en PowerPoint). Adicionalmente, en ciertos casos es directamente imposible determinar cuál método de transición se utilizó realmente (por ejemplo si el input es un subconjunto de frames de un cierto video original, y sólo se tienen frames anteriores y posteriores a la transición).

Debido a esto, no consideraremos la forma en que los métodos terminan efectuando el cambio de escena como errores o artifacts. De todas modos, observaremos tales formas como parte de nuestro análisis cualitativo.

3.3.2.1 Vecino más cercano

Los frames intermedios a los de dos escenas distintas (es decir, los que representan tal cambio de escena) en este método serán copias del frame más cercano, por lo que resultarán en una transición “limpia”, repentina, que pasa de una escena a la otra sin frames que pertenecen a ambas.

3.3.2.2 Lineal y Splines

El método lineal resultará en un "fade-in"-"fade-out" entre las escenas, con todos los frames intermedios pertenecientes a ambas. Un ejemplo de esta transición se puede ver en la imagen 3, que corresponde a un frame intermedio a dos escenas distintas (en la escena saliente se ve al bebé con mayor zoom, en la entrante con menor).

Figura 3: Transicion Lineal - video: Bebes con 6 frames intermedios

El uso de Splines resulta en el mismo tipo de transición que la interpolación lineal (ejemplo: figura 4). Nuevamente, la diferencia entre ambos yace en la intensidad de cada escena en el "fade-in"-"fade-out".

Figura 4: Transicion en Splines - video: Bebes con 6 frames intermedios, tamaño de bloque 4

Sin embargo, Splines utiliza varios frames de referencia, por lo que en algunos casos otra escena puede alterar frames que no son de transición de una a otra. Esto resulta en artifacts, como el que se puede observar en la figura 5. Cabe destacar que la frecuencia y el efecto de este artifact incrementar al aumentar el tamaño de bloque.

Figura 5: Artifact de transicion en Splines - video: Bebes con 6 frames intermedios, tamaño de bloque 50

3.3.3. Comparación de la calidad de los métodos

En lo que concierne a Splines y Lineal contra Vecino más cercano, observamos que el efecto de trazas presente en los primeros genera un efecto más dinámico y fluido en los videos, en contraposición al efecto más estático de este último. Si bien la decisión de cuál es mejor es subjetiva, personalmente nos pareció que en videos de movimiento, Splines y Lineal eran mejores. En videos que consisten mayoritariamente de cambios de escena (un slideshow de fotos, por ejemplo), depende de la opinion personal de cada uno con respecto al método de transición preferido.

En lo que concierne a Splines y Lineal, observamos que la traza es más visible (se nota más desde el punto de vista conciente), y nos pareció que era preferible la traza más sutil de interpolación lineal. Esto, junto a los artifacts de transición en Splines, nos llevan a recomendar la interpolación lineal por sobre este método.

4. Conclusiones

Los resultados cualitativos nos llevan a concluir que la Interpolación Lineal es preferible al método de Splines Cúbicos, debido a la traza más sutil del primero.

Los resultados cuantitativos muestran una ventaja del algoritmo de Interpolación Lineal por sobre los demás métodos, dejando como conclusión que no siempre el algoritmo con mayor complejidad es el que arroja mejores resultados.

Sin embargo, el algoritmo de Vecino Más Cercano, a pesar de ser de los que arrojan los peores resultados cuantitativos, podría llegar a generar videos más atractivos comparados con los otros métodos, produciendo el mismo efecto que hacen los archivos `.gif`. A veces, resulta más atractivo conseguir este tipo de videos libres de artifacts y con transiciones limpias, aunque no posean el mismo efecto de una cámara slow motion.

Desde un punto de vista más subjetivo, no recomendamos utilizar Splines cúbicos, debido a los errores visuales causados por éste método al cambiar las escenas. Por otra parte, para mayor cantidad de frames intermedios, es más útil utilizar interpolación lineal por sobre vecino más cercano, ya que este último a medida que aumenta la cantidad de frames a regenerar irá perdiendo la fluidez, resultando en videos más cortados que lineal.

5. Apéndices

5.1. Apéndice A : Algoritmos

5.1.1. Algoritmo para eliminar Frames Intermedios

```
void eliminarFrames (Parametros &params){

        assert(params.framesIntermedios < (params.frames -2));

        params.output << (params.frames/(params.framesIntermedios+1)) + ( (params.frames
            %(params.framesIntermedios+1) != 0) ? 1 : 0 ) << std::endl;
        params.output << params.height << " " << params.width << std::endl;
        params.output << params.framerate << std::endl ;

        int k = 0;
        int i, j;

        vector<vector<int> >* frame = new vector<vector<int>>(params.height, vector<int>(
            params.width));

        while (k < params.frames){
                if (k %(params.framesIntermedios+1) == 0 ){
                        //Cargo la imagen actual
                        for (i=0; i < params.height; i++){
                                for (j=0; j < params.width; j++){
                                params.input >> (*frame)[i][j];
                                }
                        }
                        imprimirFrame(params.output, *frame, params.height, params.width)
                            ;
                }else{
                        // tiro el frame
                        for (i=0; i < params.height; i++){
                                //ignoro la linea
                                //params.input.ignore(std::numeric_limits<std::streamsize
                                    >::max(), '\n');
                                for (j=0; j < params.width; j++){
                                        params.input >> (*frame)[i][j];
                                        }
                                }
                        }
                k++;
        }
        delete frame;
}
```

5.1.2. Algoritmo para calcular ECM

```
double ecm (Parametros &params, Parametros &params2){
        assert(params.height == params2.height);
        assert(params.width == params2.width );
        int frames = (params.frames <= params2.frames) ? params.frames : params2.frames;
        double promedio = 0;
        double ecm;
        double max = 0;
        int i,j,k;

        vector<vector<double> >* frameA = new vector<vector<double>>(params.height,
            vector<double>(params.width));
        vector<vector<double> >* frameB = new vector<vector<double>>(params.height,
            vector<double>(params.width));

        for (k=0; k< 5; k++){
                //cargo Frame A
```

```
                for (i=0; i < params.height; i++){
                        for (j=0; j < params.width; j++){
                        params.input >> (*frameA)[i][j];
                        }
                }
                //cargo Frame B
                for (i=0; i < params.height; i++){
                        for (j=0; j < params.width; j++){
                        params2.input >> (*frameB)[i][j];
                        }
                }
                ecm = 0;
                for (i=0; i < params.height; i++){
                        for (j=0; j < params.width; j++){
                                ecm += pow ( abs( (*frameA)[i][j] - (*frameB)[i][j] ),
                                    2);
                        }
                }
                promedio += ecm; //sumo el ecm sin haberlo dividido por el mn , lo divido
                     al final una vez sumados todos
                ecm = ecm / (params.width * params.height);
                //Imprimo ECM del frame k
                cout << "ECM frame " << k << ": " << ecm << std::endl ;
                max = (ecm > max) ? ecm : max ;

        }
        promedio = promedio / (params.width * params.height);
        promedio = promedio / frames;
        params.output << "ECM promedio: " << promedio << std::endl;
        params.output << "Maximo error obtenido: " << max << std::endl;
        return promedio;

}
```

5.2. Apéndice B: Resultados cuantitativos sobre ECM y PSNR

La siguiente tabla muestra el ECM promedio obtenido con cada algoritmo para cada input, según la cantidad de frames internos (FI) eliminados:

Cuadro 1: ECM: Promedios

Método	**FI**	**cupcake**	**perro**	**morocho**	**tenis**	**bebes**	**fideos**
Vecino Más Cercano	2	45.0112	314.75	83.308	30.9356	120.327	61.5792
Vecino Más Cercano	4	75.0243	571.526	122.311	60.5956	201.26	116.639
Vecino Más Cercano	6	91.8027	762.867	203.109	90.501	264.514	165.972
Interpolación Lineal	2	29.8591	229.528	51.6138	19.3621	86.5087	43.6452
Interpolación Lineal	4	53.6601	427.874	92.725	40.4578	153.048	86.1893
Interpolación Lineal	6	75.7466	585.446	146.864	63.0961	209.167	125.779
Splines (tamBloque=4)	2	35.3717	256.704	65.2409	20.8447	98.4989	48.9282
Splines (tamBloque=4)	4	62.177	488.344	98.4069	43.6463	179.913	98.9888
Splines (tamBloque=4)	6	85.1963	664.815	162.418	68.5048	242.925	142.752

La siguiente tabla muestra el PSNR promedio obtenido con cada algoritmo para cada input, según la cantidad de frames internos (FI) eliminados:

Cuadro 2: PSNR: Promedios

Método	**FI**	**cupcake**	**perro**	**morocho**	**tenis**	**bebes**	**fideos**
Vecino Más Cercano	2	32.5224	22.2815	39.1805	35.4634	34.6831	29.0771
Vecino Más Cercano	4	30.5887	20.5858	33.6034	33.3133	32.5697	27.2282
Vecino Más Cercano	6	29.4282	19.5938	31.3594	31.9285	31.0408	26.0286
Interpolación Lineal	2	34.1298	23.6189	34.4392	38.4812	36.8022	30.2675
Interpolación Lineal	4	31.7302	21.6657	32.8016	35.2684	34.0206	28.3235
Interpolación Lineal	6	30.3006	20.5687	31.1006	33.7438	32.1325	26.998
Splines (tamBloque=4)	2	33.8389	23.1742	33.4144	38.7222	36.2412	30.1813
Splines (tamBloque=4)	4	31.4883	21.0954	32.7601	35.3889	33.3246	27.7709
Splines (tamBloque=4)	6	29.7734	20.0804	30.7256	33.4113	31.4074	26.5067

Referencias

[1] Richard L. Burden y J. Douglas Faires. Numerical Analysis. *Ninth Edition, Youngstown State University.*