DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Métodos Numéricos
Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Iván Arcuschin	678/13	iarcuschin@gmail.com
Martín Jedwabny	885/13	martiniedva@gmail.com
José Massigoge	954/12	jmmassigoge@gmail.com
Iván Pondal	078/14	ivan.pondal@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

Abstract

Fact: baby's videos are funny. And they are even funnier when watched in *slow motion*.

In this work, we look at 4 different Interpolation methods to automatically generate *slow motion* videos. This is achieved by interpolating some amount of new frames between each pair of original frames. Depending on the method, this procedure generates different levels of *smoothness*.

Once the methods are explained and implementation details are cleared out, we will present several experiments, namely in two categories. The first ones, will asure that the methods actually work, in the general case of function interpolation. The second ones, will provide different ways to compare the quality of the videos produced by each method.

Resumen

Hecho: los videos de bebes son graciosos. Y son mucho más gracios cuando se miran en *slow motion*.

En este trabajo, nos centraremos en 4 diferentes métodos de Interpolación para generar automaticamente videos en cámara lenta. Esto se logra interpolando una cierta cantidad de cuadros nuevos entre cada par de cuadros originales. Dependiendo del método, este procedimiento genera diferentes niveles de calidad.

Una vez que hayamos explicado los métodos junto con su implementación, presentaremos varios experimentos, dentro de dos categorías. La primera, asegurará que los métodos funcionen correctamente, en el caso general de interpolación de funciones. La segunda, proveerá diferentes formas de comparar la calidad de los videos producidos por cada método.

Keywords: Video, slow motion, interpolation methods, splines

Índice

1. Introducción

El objetivo principal de este Trabajo Práctico es estudiar, implementar y analizar métodos de Interpolación para generar Videos con *slow motion*.

Comenzaremos haciendo una breve introducción a los distintos métodos de Interpolación, para luego explicar cual es el modelo que subyace en cada uno:

- Interpolación por Vecinos.
- Interpolación Fragmentaria Lineal.
- Interpolación por Splines.
- Interpolación por Splines (bloques de tamaño fijo).

Una vez finalizada la parte del Modelo, pasaremos a describir la Implementación de los diferentes métodos presentados, realizadas en `C++`.

Ya llegando al final, pasaremos a presentar la Experimentación realizada, a la vez que iremos analizando y discutiendo los resultados obtenidos.

Los experimentos realizados pueden dividirse en dos categorías. La primera, relacionada con el costo temporal y la correctitud de los diversos algoritmos utilizados:

- Funcionamiento de los métodos implementados al tratar de interpolar diferentes familias de funciones.
- Comparación de tiempos de ejecución entre los distintos métodos.

La segunda, relacionada con el aspecto cualitativo de los métodos:

- Comparación del *Error Cuadrático Medio* y *Peak to Signal Noise Rate* entre los distintos métodos.
- Análisis del fenómeno de Artifacts.

Para finalizar, cerraremos el presente informe con una conclusión, en la cual discutiremos acerca de los métodos vistos, así como de la experimentación realizada. También, contaremos las dificultades encontradas al realizar el Trabajo Práctico, las posibles continuaciones que se podrían realizar, y si los objetivos planteados fueron alcanzados.

2. Modelo

2.1. Video

Definiremos un modelo para los videos con el cual sea fácil de trabajar a la hora de realizar el *slow motion*. Dado un video, definiremos:

- w el ancho en píxeles de cada frame.
- h el ancho en píxeles de cada frame.
- f_i el i-ésimo frame, con $0 < i < k$, donde k es la cantidad de frames totales.
- $p(x, y, f_i)$, con $0 < x < w$, $0 < y < h$, el píxel en la posición (x, y) del frame f_i.

Luego, si tomamos $p(x, y, f_i)$ y $p(x, y, f_{i+1})$ querremos agregar una cierta cantidad de píxeles entre ambos, de forma que haya una transición del primero al segundo y se produzca el *slow motion*.

Para elegir que valores agregar entre los píxeles, utilizaremos diferentes métodos de interpolación.

- *Vecinos*: Consiste en rellenar los nuevos frames replicando los valores de los píxeles del frame original más cercano.
- *Interpolación Lineal*: Consiste en rellenar los píxeles utilizando interpolaciones lineales entre píxeles de frames originales consecutivos.
- *Interpolación por Splines*: Consiste en rellenar los píxeles utilizando Splines entre píxeles de frames originales consecutivos. En este método, utilizaremos la información provista por todos los frames del video, y generaremos $k - 1$ funciones, cada una de a lo sumo grado cúbico.
- *Interpolación por Splines con tamaño de bloque variante*: Simliar al anterior, pero con la posibilidad de variar la cantidad de frames tomados en cuenta al generar las funciones.

En las siguientes secciones explicaremos con mayor detalle cada uno de los métodos.

2.2. Vecinos

En este método, eligiremos para cada nuevo píxel el valor del frame original que se encuentre más cercano. Si definimos c como la cantidad de frames a agregar entre cada par original, y $g_0, \ldots, g_{c-1}$ los nuevos frames. Tenemos que:

$$\begin{aligned}
p(x, y, f_i) &= p(x, y, f_i) \\
p(x, y, g_0) &= p(x, y, f_i) \\
&\vdots \\
p(x, y, g_{c/2-1}) &= p(x, y, f_i) \\
p(x, y, g_{c/2}) &= p(x, y, f_{i+1}) \\
&\vdots \\
p(x, y, g_{c-1}) &= p(x, y, f_{i+1}) \\
p(x, y, f_{i+1}) &= p(x, y, f_{i+1})
\end{aligned}$$

2.3. Interpolación Fragmentaria Lineal

En este método, buscaremos interpolar los píxeles de frames contiguos con una función lineal. Para ello, construiremos un Polinomio Interpolante de grado 1 utilizando *diferencias divididas*, ya que ofrece una construcción más sencilla que al seguir el método de Lagrange.

Luego, si llamamos f a la función (desconocida excepto en los puntos x_j), definimos:

- Diferencia dividida de orden cero en x_j:

$$f[x_j] = f(x_j)$$

- Diferencia dividida de orden uno en x_j, x_{j+1}:

$$f[x_j, x_{j+1}] = \frac{f[x_{j+1}] - f[x_j]}{x_{j+1} - x_j} = \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}$$

- Polinomio Interpolante de grado 1 para x_j, x_{j+1}:

$$P_1(x) = f[x_j] + f[x_j, x_{j+1}](x - x_j) = f(x_j) + \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j} * (x - x_j)$$

2.4. Interpolación por Splines

El método de interpolación por splines se basa en dados n puntos la construcción de $n-1$ funciones que interpolan los puntos y además cumplen una serie de condiciones que aseguran que la función por tramos resultante no posea las irregularidades de trabajar con polinomios de alto grado generando además uniones suaves entre cada segmento.

Dada una función f definida en el intervalo $[a, b]$ y un conjunto de nodos $a = x_0 < x_1 < \ldots < x_n = b$.

1. Definimos $S(x)$ como un polinomio cúbico denominándolo $S_j(x)$ en el subintervalo $[x_j, x_{j+1}]$ con $j \in [0, \ldots, n-1]$.
2. $S_j(x_j) = f(x_j)$ y $S_j(x_{j+1}) = f(x_{j+1})$ para todo $j \in [0, \ldots, n-1]$.
3. $S_j(x_{j+1}) = S_j(x_{j+1})$ para todo $j \in [0, \ldots, n-2]$.
4. $S'_j(x_{j+1}) = S'_j(x_{j+1})$ para todo $j \in [0, \ldots, n-2]$.
5. $S''_j(x_{j+1}) = S''_j(x_{j+1})$ para todo $j \in [0, \ldots, n-2]$.
6. Por último, se cumple una de la siguientes condiciones
   a) $S''(x_0) = S''(x_n) = 0$ (natural o de libre frontera).
   b) $S'(x_0) = f'(x_0)$ y $S'(x_n) = f'(x_n)$ (sujeta).

Un spline definido en un intervalo que está dividido en n subintervalos requiere determinar $4n$ constantes. Se aplican las condiciones descritas previamente a los siguiente polinomios cúbicos:

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$
$$\text{para cada } j \in [0, \ldots, n-1]$$

Para este trabajo, dado que no conocemos la función f que estamos interpolando se decidió utilizar la condición 6*a*, que define al spline como natural o libre.

Una vez establecidas las ecuaciones resultantes de aplicar las condiciones y despejando todas las variables en función de c_j nos quedan las siguientes igualdades:

$$\begin{aligned} h_j &= x_{j+1} - x_j \\ a_j &= f(x_j) \text{ para cada } j \in [0, \ldots, n] \\ b_j &= \frac{a_{j+1} - a_j}{h_j} - \frac{2c_j h_j - c_{j+1} h_j}{3} \\ d_j &= \frac{c_{j+1} - c_j}{3h_j} \\ &\text{para cada } j \in [0, \ldots, n-1] \end{aligned}$$

Donde los c_j nos quedan determinados por el siguiente sistema de ecuaciones:

$$c_0 = 0$$
$$c_n = 0$$
$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3(a_{j+1} - a_j)}{h_j} + \frac{3(a_{j-1} - a_j)}{h_{j-1}}$$

para cada $j \in [1, \ldots, n-1]$

El mismo se puede representar como la siguiente matriz:

$$\begin{array}{c} \\ c_0 \\ c_1 \\ \vdots \\ c_j \\ \vdots \\ c_{n-1} \\ c_n \end{array} \begin{array}{c} \begin{array}{ccccccccccc} c_0 & c_1 & c_2 & \ldots & c_{j-1} & c_j & c_{j+1} & \ldots & c_{n-2} & c_{n-1} & c_n \end{array} \\ \left[\begin{array}{ccccccccccc} 1 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & h_{j-1} & 2(h_{j-1}+h_j) & h_j & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} b \\ \left|\begin{array}{c} 0 \\ \frac{3(a_2-a_1)}{h_1} + \frac{3(a_0-a_1)}{h_0} \\ \vdots \\ \frac{3(a_{j+1}-a_j)}{h_j} + \frac{3(a_{j-1}-a_j)}{h_{j-1}} \\ \vdots \\ \frac{3(a_n-a_{n-1})}{h_{n-1}} + \frac{3(a_{n-2}-a_{n-1})}{h_{n-2}} \\ 0 \end{array}\right] \end{array}$$

Ahora para el problema planteado, que es la interpolación de los cuadros de un video, nuesto h_j será la distancia entre cada uno de ellos. Esta distancia la podemos pensar como el tiempo entre cada captura del video, y como la duración del mismo se define por la cantidad de cuadros por segundo, podemos afirmar que son equidistantes, por lo tanto tenemos $h_j = 1$ para todo $j \in [0, \ldots, n-1]$.

Reemplazando en la matriz anterior nos queda:

$$\begin{array}{c} \\ c_0 \\ c_1 \\ \vdots \\ c_j \\ \vdots \\ c_{n-1} \\ c_n \end{array} \begin{array}{c} \begin{array}{ccccccccccc} c_0 & c_1 & c_2 & \ldots & c_{j-1} & c_j & c_{j+1} & \ldots & c_{n-2} & c_{n-1} & c_n \end{array} \\ \left[\begin{array}{ccccccccccc} 1 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 1 & 4 & 1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 4 & 1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 1 & 4 & 1 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} b \\ \left|\begin{array}{c} 0 \\ 3(a_2 - a_1) + 3(a_0 - a_1) \\ \vdots \\ 3(a_{j+1} - a_j) + 3(a_{j-1} - a_j) \\ \vdots \\ 3(a_n - a_{n-1}) + 3(a_{n-2} - a_{n-1}) \\ 0 \end{array}\right] \end{array}$$

Como se puede observar resulta en una matriz cuadrada tridiagonal estrictamente dominante por filas. Como consecuencia tenemos que la misma y sus submatrices principales son inversibles, llevando a que el sistema tenga una única solución y que además A posea factorización LU.

Una vez que se tiene la factorización $A = LU$, sólo queda resolver el sistema planteado y así calcular los coeficientes de cada $S_j(x)$. Por último resta evaluar el spline en los puntos donde se agregaron cuadros nuevos para así poder lograr el efecto buscado.

El beneficio de utilizar la factorización LU es que el spline que se utiliza para interpolar cada pixel del video comparte el mismo sistema, lo que cambia son los valores de los a_j que definen nuestro vector b. De esta manera, recalculamos el vector b, utilizamos la factorización LU para resolver el sistema y nuevamente calculamos los coeficientes del spline.

2.5. Interpolación por Splines con tamaño de bloque fijo

Este método tiene el mismo fondo teórico que el anterior ya que también trabaja con splines. La diferencia es que en el modelo anterior, dados todos los valores que tomaba cada pixel del video se generaba un único spline para cada pixel, mientras que con splines de a bloques lo que hacemos es para cada pixel generar varios splines conectados entre si.

La motivación para realizar esto es la hipótesis de que si estamos trabajando con un video que contiene cambios bruscos, el utilizar toda la duración del video para alimentar un spline puede resultar impreciso,

mientras que si generamos splines asociados a tramos del video, creemos que podría comportarse mejor a variaciones abruptas.

Dada una función f definida en el intervalo $[a, b]$ y un conjunto de nodos $a = x_0 < x_1 < \ldots < x_n = b$, definimos B como el tamaño de cada bloque y $Sb(x)$ como un spline de a bloques con $Sb_j(x)$ siendo el spline definido en el subintervalo $[x_j, x_{j+B}]$ con $j \in [0, \ldots, n - B - 1]$ que interpola los puntos de f en ese mismo intervalo.

3. Implementación

3.1. Interpolación por vecinos

Este método consiste en reemplazar los cuadros intermedios a ser rellenados por el cuadro original mas cercano en el tiempo. Es decir, dados los cuadros del video sin camara lenta, generamos otro video en camara lenta copiando los cuadros originales de la siguiente manera:

Sean Frame1 y Frame2 dos cuadros consecutivos del video original:

Frame1	Frame2

Si queremos ahora 6 cuadros entre cada 2 del archivo original lo transformamos a:

Frame1	Frame1	Frame1	Frame1	Frame2	Frame2	Frame2	Frame2

El pseudocódigo sería el siguiente:

```
Sean W,H,I el ancho, alto y la cantidad de frames del video original
Sea video[W][H][I] el triple vector de numeros enteros que representa el
    video original
Sea K la cantidad de frames que queremos agregar entre cuadro y cuadro

Crear un triple vector de enteros new_video[W][H][I+(I-1)*K]
Para w = 0 hasta W-1 hacer
  Para h = 0 hasta H-1 hacer
    Para i = 0 hasta I-2 hacer
      Para j = 0 hasta K/2 hacer
        new_video[w][h].push_back(video[w][h][i])
      Fin para
      Para j = (K/2)+1 hasta K hacer
        new_video[w][h].push_back(video[w][h][i+1])
      Fin para
    Fin para
    new_video[w][h].push_back(video[w][h][I-1])
  Fin para
Fin para
Devolver new_video
```

3.2. Interpolación lineal

En este caso, usamos el polinomio interpolador de Lagrange entre cada par de puntos/pixeles consecutivos para aproximar los valores intermedios que irían en el video de camara lenta. Esto genera una función lineal para los pixeles consecutivos en la misma posición.

Por ejemplo, sean dos pixeles con valores 1 y 4:

1	4

Si queremos un video en camara lenta con 5 cuadros intermedios por cada 2 del original, estos se replicarán de la siguiente forma:

1	1.5	2	2.5	3	3.5	4

El procedimiento es el siguiente:

```
Sean W,H,I el ancho, alto y la cantidad de frames del video original
Sea video[W][H][I] el triple vector de numeros enteros que representa el
    video original
Sea K la cantidad de frames que queremos agregar entre cuadro y cuadro

Crear un triple vector de enteros new_video[W][H][I+(I-1)*K]
Para w = 0 hasta W-1 hacer
  Para h = 0 hasta H-1 hacer
    Para i = 0 hasta I-2 hacer
      coef_cero = video[w][h][i]
      coef_uno = (video[w][h][i+1] - video[w][h][i]) / (K+1);
      Para k = 0 hasta K hacer
        pixel = coef_cero + coef_uno*k;
        Si (pixel < 0) pixel = 0
        Si (pixel > 255) pixel = 255
        new_video.push_back(pixel)
    Fin para
    new_video.push_back(video[w][h][I-1])
  Fin para
Fin para
Devolver new_video
```

3.3. Interpolación por Splines

En este método aplicamos la técnica de Splines. Esta consiste en generar un sistema de ecuaciones para encontrar una función por partes que interpole cada par de puntos con polinomios de forma que la curva resultante sea continua y dos veces derivable. Como el sistema es tridiagonal, podemos aprovechar para guardar los valores de la matriz de forma más eficiente. El pseudocódigo es el siguiente:

```
Sean W,H,I el ancho, alto y la cantidad de frames del video original
Sea video[W][H][I] el triple vector de numeros enteros que representa el
    video original
Sea K la cantidad de frames que queremos agregar entre cuadro y cuadro

Crear un triple vector de enteros new_video[W][H][I+(I-1)*K]
Para w = 0 hasta W-1 hacer
  Para h = 0 hasta H-1 hacer
    Crear un vector de enteros valores[I+(I-1)*K]
    GenerarSpline(video[w][h], valores, I, K)
    new_video[w][h] = valores
  Fin para
Fin para
Devolver new_video

GenerarSpline(y, valores, I, K):
  n = y.size()
  Crear doble vector de enteros sistema[n][2]
  sistema[0] = {1,0}
  Para j = 1 hasta n-2 hacer
    sistema[j] = {1, 4}
  Fin para
  sistema[n-1] = {0,1}
  // Factorización LU
  Para j = 1 hasta n-2 hacer
```

```
    coef = sistema[i + 1][0]/sistema[i][1];
    sistema[i + 1][0] = coef
    sistema[i + 1][1] -= coef
  Fin para
  Crear vectores x[n], a[n], b[n], c[n], d[n]
  a = y
  Para i = 1 hasta n-2 hacer
    x[i] = 3*(a[i + 1] - 2*a[i] + a[i - 1]);
    x[i] -= x[i - 1]*sistema[i][0];
  Fin para
  // Resuelvo triangular superior (Uc = x)
  Para i = n - 2 hasta 1 hacer
    c[i] = x[i];
    c[i] -= c[i + 1];
    c[i] /= sistema[i][1];
  Fin para
  // Calculo mis coeficientes "b" y "d"
  Para i = 0 hasta n-2 hacer
    b[i] = a[i + 1] - a[i] - (2*c[i] + c[i + 1])/3;
    d[i] = (c[i + 1] - c[i])/3;
  Fin para
  //Calculo los pixeles resultantes con el Spline
  Para i = 0 hasta n-1 hacer
    Para j = 0 hasta K hacer
      dif = j/(K+1)
      val = a[i] + b[i]*(dif) + c[i]*(dif)*(dif) + d[i]*(dif)*(dif)*(dif
          )
      valores.push_back(val)
    Fin para
  Fin para
  valores.push_back(y[n-1])
```

3.4. Interpolación por Splines de a bloques

A diferencia del método anterior, en este caso vamos a querer aplicar la técnica de Splines a bloques de pixeles de tamaño fijo. Es decir, para una misma posición del video, en vez de utilizar todos los valores del pixel a través del tiempo, generamos splines entre particiones de la misma longitud. El procedimiento es:

```
Sean W,H,I el ancho, alto y la cantidad de frames del video original
Sea video[W][H][I] el triple vector de numeros enteros que representa el
    video original
Sea K la cantidad de frames que queremos agregar entre cuadro y cuadro
Sea T el tamano de bloque de los Splines
Observacion: reutilizamos el metodo GenerarSpline() de los Splines
   normales
Observacion: asumimos que I es divisible por T para simplificar este
   pseudocodigo

Crear un triple vector de enteros new_video[W][H][I+(I-1)*K]
Para w = 0 hasta W-1 hacer
  Para h = 0 hasta H-1 hacer
    Para t = 0 hasta T-1 hacer
      Crear subarreglo aux[T] = video[w][h][T*t..T*(t+1)-1]
```

```
      Crear arreglo bloque
      GenerarSpline(aux, bloque, T, K)
      Pushear a new_video[w][h] todos los valores de 'bloque' excepto el
          ultimo
    Fin para
    new_video[w][h].push_back(video[w][h][I-1])
  Fin para
Fin para
Devolver new_video
```

4. Experimentación

En esta sección, se detallan los diferentes experimentos que realizamos para medir el funcionamiento, la eficiencia y calidad de resultados, tanto de forma cuantitativa como cualitativa, de los métodos implementados.

Para lograr tal fin realizamos los siguientes tipos de experimentos:

- Funcionamiento de los métodos implementados: Mostraremos que los métodos de interpolación funcionan correctamente comparandolos contra diferentes familias de funciones. A su vez, basandonos en las precisiones obtenidas, determinaremos el tamaño de bloque óptimo para el método Interpolación por Splines.
- Medición del ECM y PSNR de los métodos: Compararemos los errores obtenidos en varias instancias de pruebas para los distintos métodos.
- Medición de los tiempos de ejecución de los métodos: Compararemos los tiempos de ejecución utilizando los distintos métodos.
- Análisis cualitativos de los métodos, fenómeno de artifacts: Buscaremos reconocer defectos de interpolación en los videos generados.

Los videos utilizados para los diversos experimentos, fueron los siguientes:

- **Video 1 - Skate**: 426x240, cantidad de cuadros originales: 151 , fps: 30, duracion: 5s.
- **Video 2 - Messi**: 426x240, cantidad de cuadros originales: 151, fps: 30, duracion: 5s.
- **Video 3 - Amanecer**: 426x240, cantidad de cuadros originales: 151, fps: 30, duracion: 5s.

Es importante mencionar que cada video representa una clase de video distinto, en donde el Video 1 contiene movimientos bruscos, el Video 2 cambios de camara, y el Video 3 movimientos suaves.

El motivo de estas elecciones se debe a la busqueda de diversos *artifacts* a partir de las caracteristicas de cada clase.

4.1. Detalles generales de la experimentación

- En los experimentos que se utilizaron números aleatorios, se generaron utilizando la función *rand*, provista por la librería `stdlib.h`.
- La semilla para los números aleatorios se seteo utilizando el método *srand(time(NULL))*, para evitar repeticiones de números en diferentes corridas.
- Las instancias de prueba fueron generadas con los archivos provistos por la cátedra. Adicionalmente, hicimos nuestras propias instancias emulando diferentes funciones (por ejemplo una función constante, lineal y cuadrática) para realizar el control de calidad de los métodos.
- Para medir los tiempos utilizamos la librería *chrono* y medimos los resultados en nanosegundos.
- A su vez, utilizamos el nivel de optimización *O2* de *C++* a la hora de compilar el código.
- Todos los tests fueron corridos en la misma máquina bajo las mismas condiciones.

4.2. Funcionamiento de los métodos implementados

En este experimento nuestro objetivo fue asegurarnos el correcto funcionamiento de nuestra implementación de la interpolación fragmentaria lineal, interpolación por splines, e interpolación por splines con tamaño de bloque fijo, tomando bloques de 2, 4, 8, 16, 32 y 64 cuadros.

Con este fin, realizamos una serie de tests que muestran el correcto funcionamiento de cada método para distintas familias de funciones:

- Función constante.
- Función lineal.
- Función cuadrática.
- Función cúbica.

Luego, cada método de interpolación fue testeado contra cada una de las familias de funciones mencionadas de la siguiente forma:

- Dada una familia de funciones, se generan aleatoriamente los coeficientes necesarios para definir una función de esa familia, i.e.: para una constante se genera solo el coeficiente independiente, mientras que para una cuadrática se generan 3 coeficientes.
- Una vez generada la función, se la evalua en un rango de valores para obtener un array de valores esperados.
- Luego, a partir del array de valores esperados se construye otro array quitandole elementos a intervalos fijos. Este nuevo array será el utilizado para realizar la interpolación, y lo que testearemos es la aproximación de la interpolación a los elementos que quitamos.
- Una vez que tenemos la interpolación con cualquiera de los métodos mencionados, basta recorrer los elementos del array de valores esperados a la vez que evaluamos la interpolación obtenida. Para cada par de valores: esperado e interpolado, querremos ver que la diferencia absoluta es menor que un epsilon/cota de precision que definiremos dependiendo del método utilizado y la función a interpolar.

Es importante mencionar algunas carácterísticas de las instancias utilizadas:

- Cantidad de puntos generados con la función(tamaño del array de valores esperados): 100
- Cantidad de puntos a interpolar: 50.
- Todos los coeficientes generados aleatoriamente están en el rango $[1, 10]$, para evitar que las funciones generadas crezcan de forma desmedida.

Luego, se obtuvieron las siguientes cotas de precisión para los distintos métodos y funciones:

	F. Constante	F. Lineal	F. Cuadrática	F. Cúbica
Interpolación por Vecinos	0.0001	10	1000	100000
Interpolación Fragmentaria Lineal	0.0001	0.0001	10	1000
Interpolación por Splines (bloques tamaño 2)	0.0001	0.0001	10	1000
Interpolación por Splines (bloques tamaño 4)	0.0001	0.0001	5	500
Interpolación por Splines (bloques tamaño 8)	0.0001	0.0001	1	200
Interpolación por Splines (bloques tamaño 16)	0.0001	0.0001	1	200
Interpolación por Splines (bloques tamaño 32)	0.0001	0.0001	1	200
Interpolación por Splines (bloques tamaño 64)	0.0001	0.0001	1	200
Interpolación por Splines (1 solo bloque)	0.0001	0.0001	1	200

Analizando dichas cotas vemos que:

- La Interpolación por Vecinos resulta razonable solo para funciones con muy poca variación (derivada a lo sumo constante). Esto se ve claramente en la cota de presición al interpolar una función Cuadrática.
- La Interpolación Fragmentaria Lineal es sustancialmente mejor que por Vecinos, y devuelve resultados razonables para funciones a lo sumo Cuadráticas.
- La Interpolación por Splines utilizando solo 2 puntos para cada bloque es equivalente a interpolar utilizando funciones lineales, y queda evidenciado al tener las mismas cotas de precisión.

- A partir de los tamaños de bloque 4 a 16, la Interpolación por Splines realiza una mejora "asintótica" de su cota de precisión.
- Entre los tamaños de bloque 16 a 64, no se notó una mejora significativa en la cota de precisión de la Interpolación por Splines.
- Al realizar Interpolación por Splines *standard* (1 solo bloque), vemos que su cota de precisión concuerda con las cotas a las cuales "convergen" los métodos de Interpolación por Splines con bloque de tamaño fijo. Es por esta razón que para los experimentos cualitativos, utilizaremos solamente la Interpolación por Splines *standard*, y no la de tamaño de bloques fijo.

4.3. Medición de los tiempos de ejecución de los métodos

A partir de la implementaciones descriptas en la Sección 3, podemos inferir una complejidad temporal para cada método.

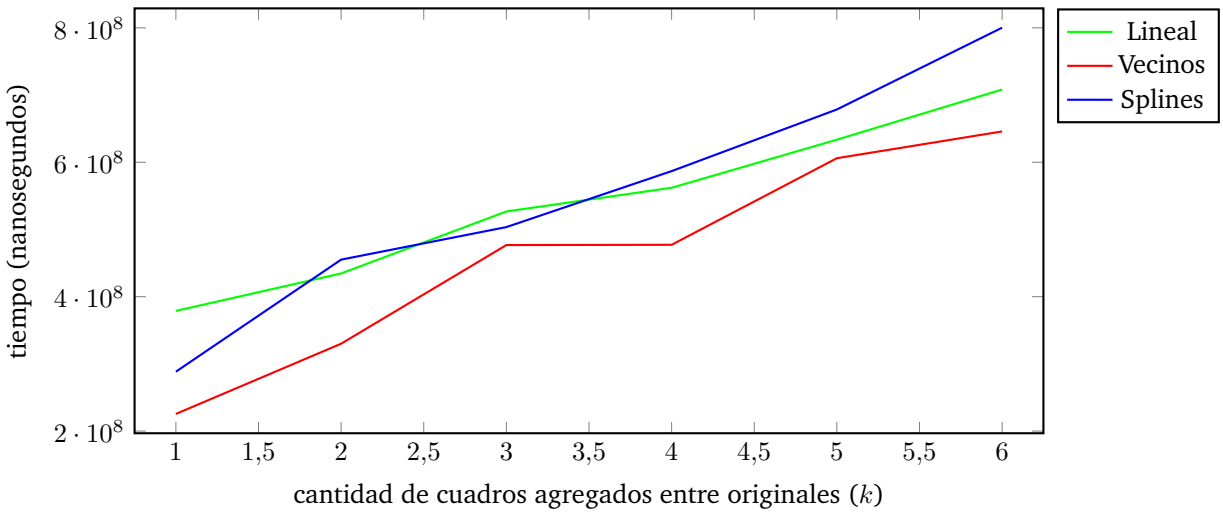
Sea w la cantidad de filas de píxeles en cada imagen, h la cantidad de columnas, i la cantidad de cuadros originales y sea k la cantidad de cuadros a agregar entre los originales:

Todos los metodos implementados tienen la misma complejidad temporal que es $\Theta(w*h*i*k*)$. Esta conclusion surge del hecho de que, en todos los casos, tenemos 4 ciclos anidados, en donde el primero se ejecuta w veces, el segundo h veces, el tercero i veces, y el cuarto k veces.

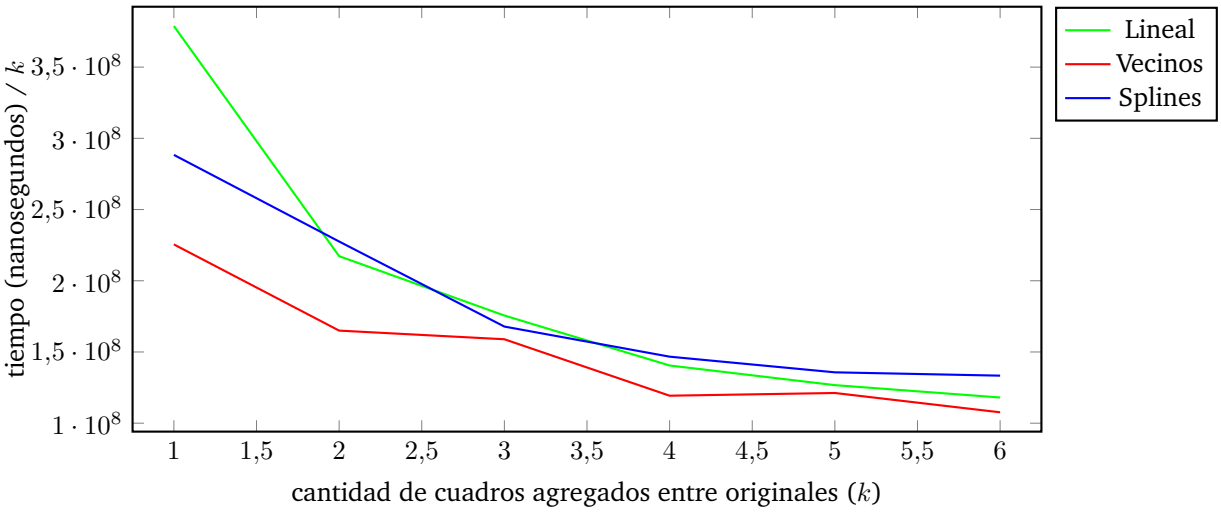
Para corroborar esta hipotesis planteamos un experimento con las siguientes caracteristicas:

- Utilizamos el video provisto por la catedra llamado *funnybaby*, el cual tiene 44 cuadros y es de 240x320.
- Solo medimos la resolucion del sistema, no su creacion o los respectivos pasajes de video a texto y viceversa.
- Dado que w, h y i son valores fijos que no podemos cambiar, variamos el k.
- Generamos instancias para valores de k entre 1 y 6.
- Para cada valor distinto de k generamos 4 instancias, cuyos valores vamos a promediar para mitigar los posibles valores distorsionados por algun procedimiento del procesador.

Los resultamos obtenidos fueron los siguientes:



Si tomamos los tiempos que arrojó la experimentación, y los dividimos por su respectivo k, obtenemos el siguiente resultado:



A partir de los graficos, queda de manifiesto que los metodos tienen la misma complejidad temporal, ya que solo difieren en una constante.

4.4. Medición del ECM y PSNR de los métodos.

Sea F un frame del vídeo real (ideal) , y $\bar{F}$ el mismo frame del vídeo efectivamente construidos por alguno de los métodos. Sea m la cantidad de filas de píxeles en cada imagen y n la cantidad de columnas.

Definimos el Error Cuadrático Medio, `ECM`, como el real dado por:

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

A su vez definimos *Peak to Signal Noise Ratio*, `PSNR`, como el real dado por:

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

Ambas medidas nos sirven para realizar un análisis cuantitativo de la calidad de los resultados obtenidos con los distintos métodos.

En este experimento utilizamos los videos propuestos al inicio de la experimentacion, variando la cantidad de cuadros que agregamos. Dichos videos fueron elegidos especificamente para variar la dificultad de Interpolación:

- El video **Amanecer** no contiene movimientos bruscos ni cambios de cámaras por lo que, para cualquier método, el error debería ser en lineas generales menor que pare el resto de los videos.
- El video **Skate** contiene movimientos bruscos pero no cambios de cámaras por lo que, se espera un mayor error que en el video anterior. Además, deberíamos ver un aumento del error en los frames donde hay movimientos bruscos.
- El video **Skate** contiene movimientos bruscos y cambios de cámaras por lo que, se espera un mayor error que en el resto de los videos. Además, deberíamos ver un aumento *importante* del error en los frames donde se produce el cambio de cámara.

A continuación, presentaremos e iremos analizando los resultados obtenidos:

4.4.1. ECM - Agregando 1 cuadro

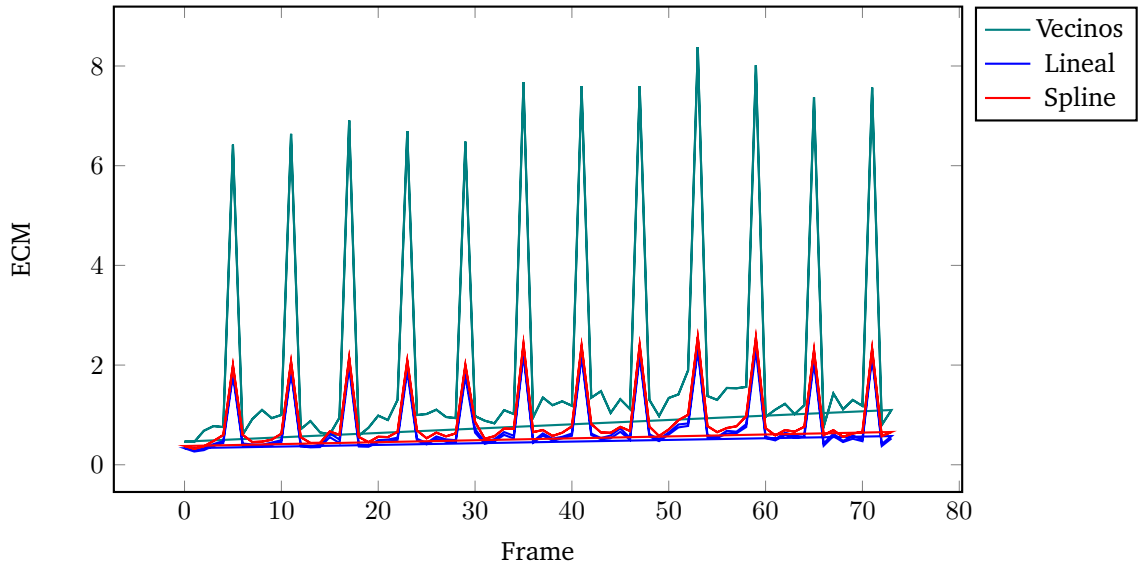


Figura 1: ECM para el video **Amanecer** al agregar 1 cuadro con distintos métodos de Interpolación.

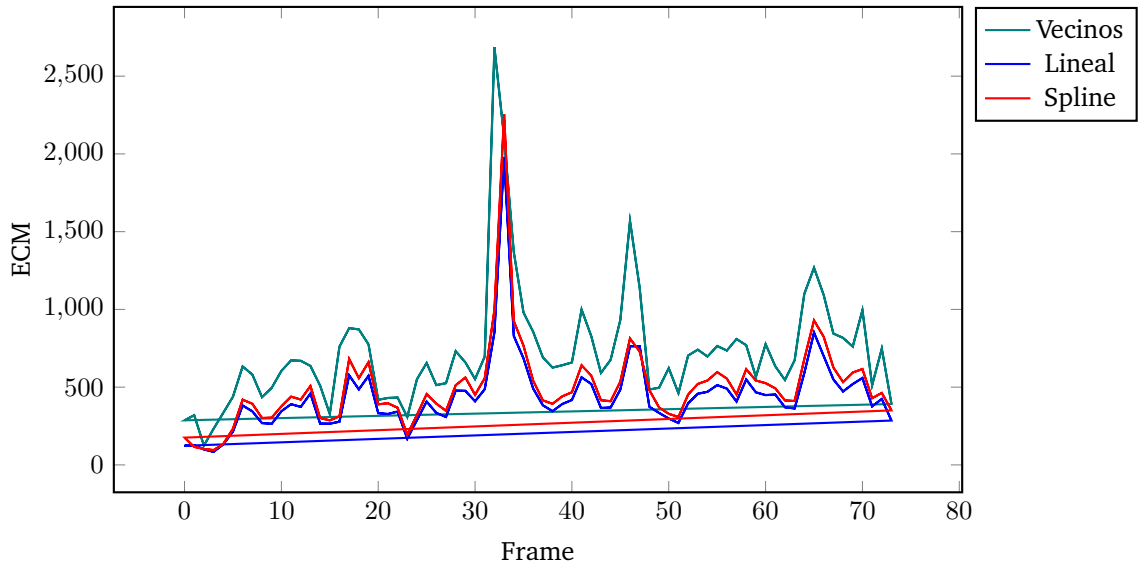


Figura 2: ECM para el video **Skate** al agregar 1 cuadro con distintos métodos de Interpolación.

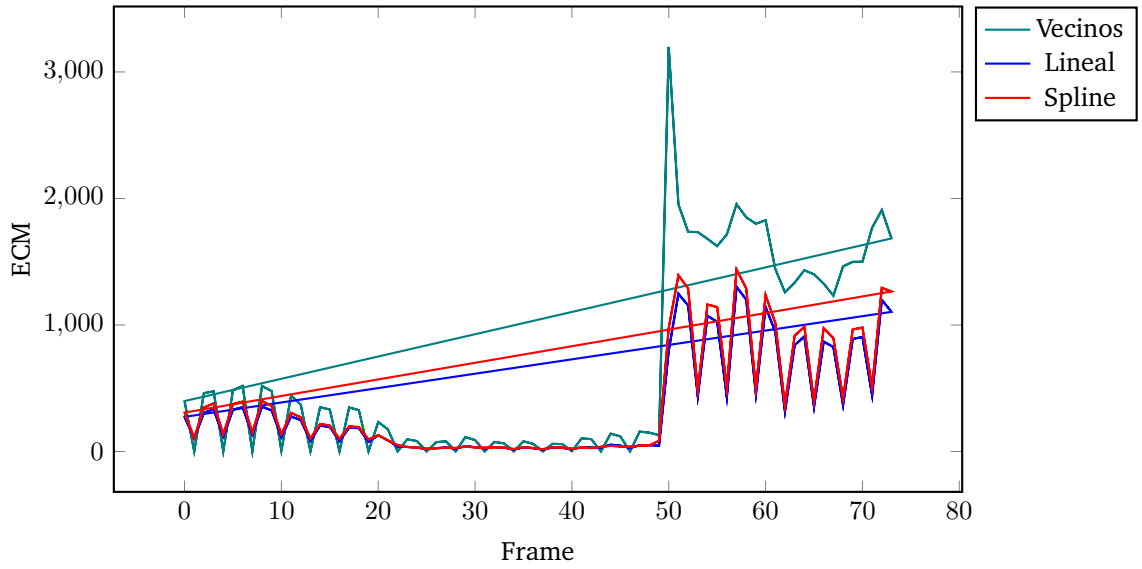


Figura 3: ECM para el video **Messi** al agregar 1 cuadro con distintos métodos de Interpolación.

Viendo las Figuras 1, 2 y 3 podemos decir que:

- La Interpolación por Vecinos obtiene consistentemente un mayor error que el resto de los métodos.
- La Interpolación Fragmentaria Lineal y la Interpolación por Splines obtienen en general un error similar, siendo la primera ligeramente mejor.
- El rango de error obtenido por todos los métodos en el video **Amanecer**, es significativamente menor que en el resto de los videos. Además, se puede ver que el error obtenido en ese video es en cierta forma “regular”, lo cual tiene sentido ya que tiene movimientos suaves y repetitivos.
- El rango de error obtenido por todos los métodos en los videos **Skate** y **Messi**, son similares, y ambos mayores al rango obtenido en el video **Amanecer**.
- El error obtenido en el video **Skate** es bastante irregular al compararlo con el video **Amanecer**, encontrandose un pico de error en el movimiento más brusco del video.
- A diferencia del caso anterior, el error obtenido en el video **Messi** es bastante regular, a excepción del pico de error que encontramos cuando ocurre el cambio de cámara, el cual es notorio para todos los métodos.

4.4.2. PSNR - Agregando 1 cuadro

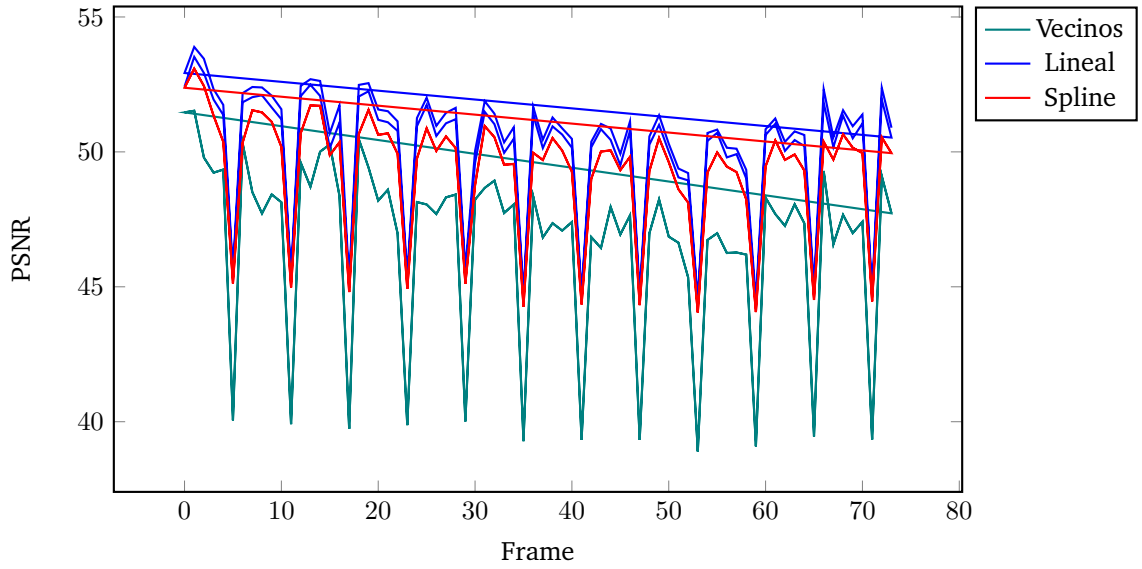


Figura 4: PSNR para el video **Amanecer** al agregar 1 cuadro con distintos métodos de Interpolación.

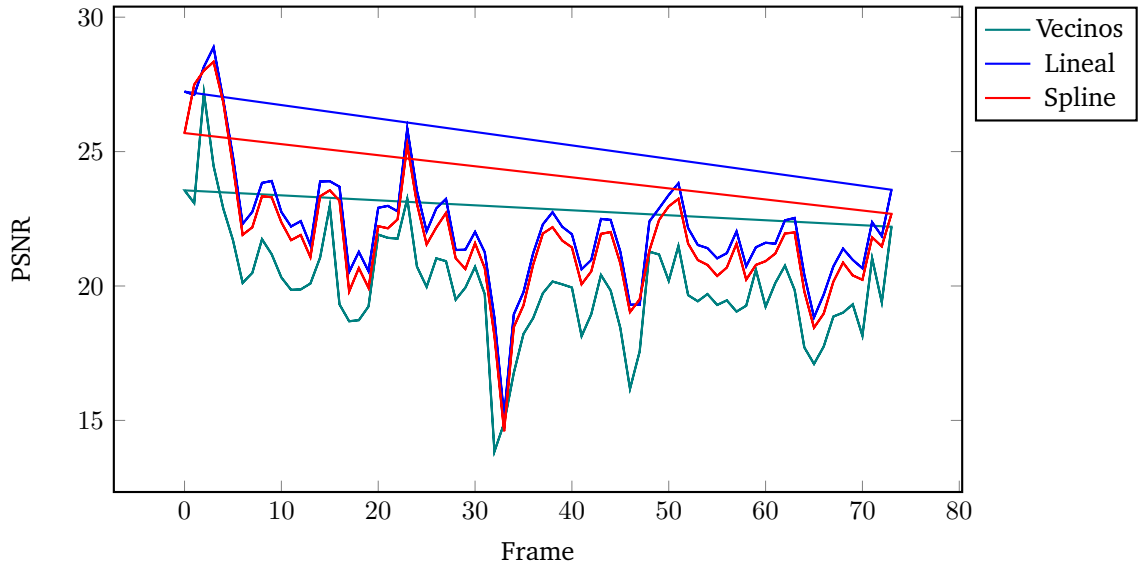


Figura 5: PSNR para el video **Skate** al agregar 1 cuadro con distintos métodos de Interpolación.

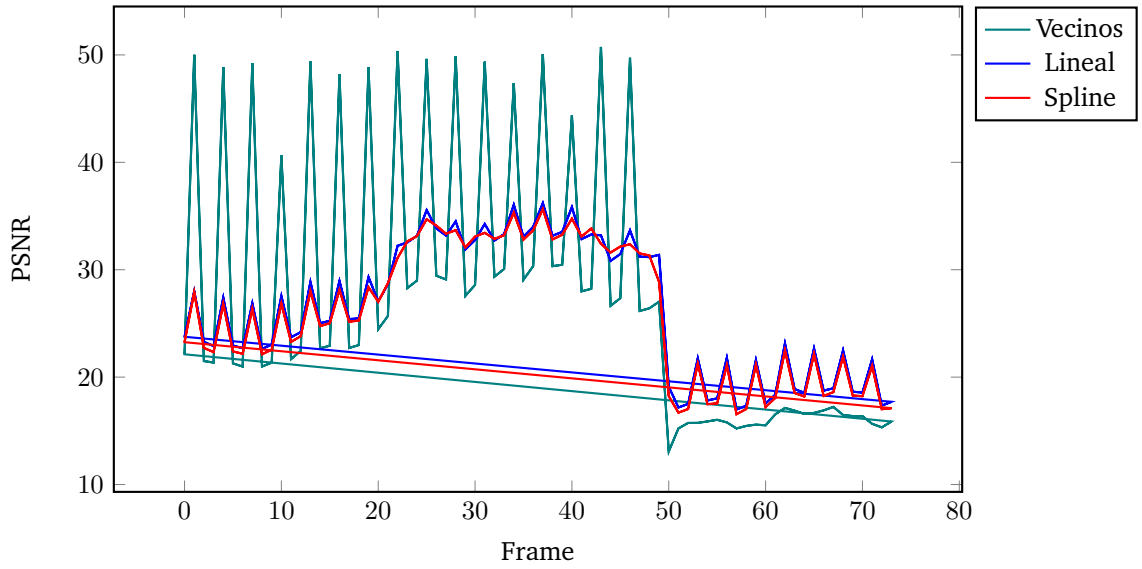


Figura 6: PSNR para el video **Messi** al agregar 1 cuadro con distintos métodos de Interpolación.

Sabiendo que, mientras más grande es el `ECM` más chico es el `PSNR`, encontramos que la información provista por este último no aporta nuevos elementos al análisis, ya que se condice con lo análizado previamente utilizando el `ECM`.

4.4.3. ECM - Agregando 5 cuadros

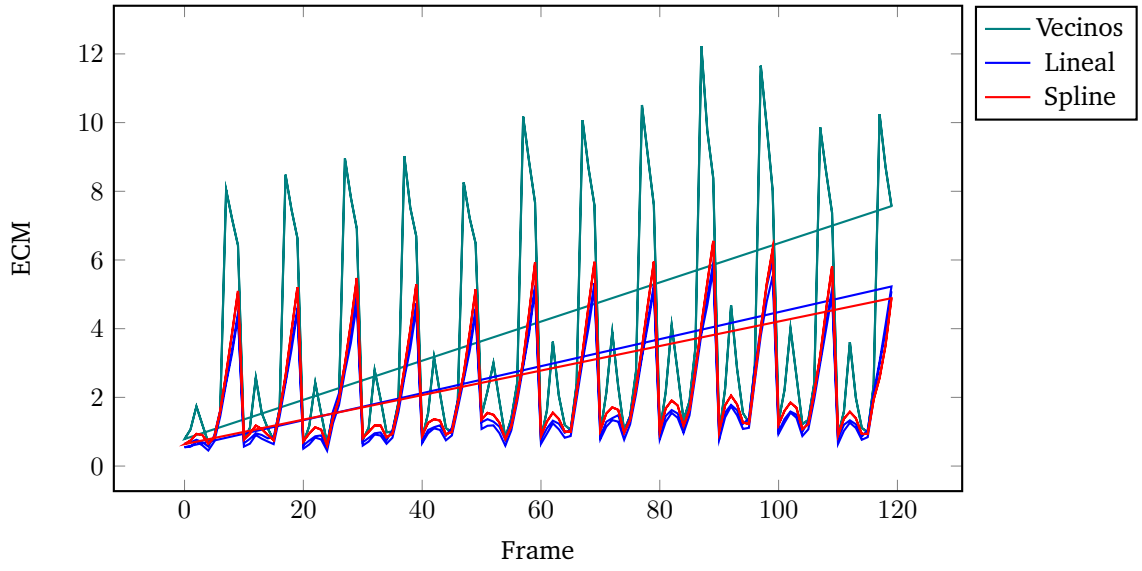


Figura 7: ECM para el video **Amanecer** al agregar 5 cuadros con distintos métodos de Interpolación.

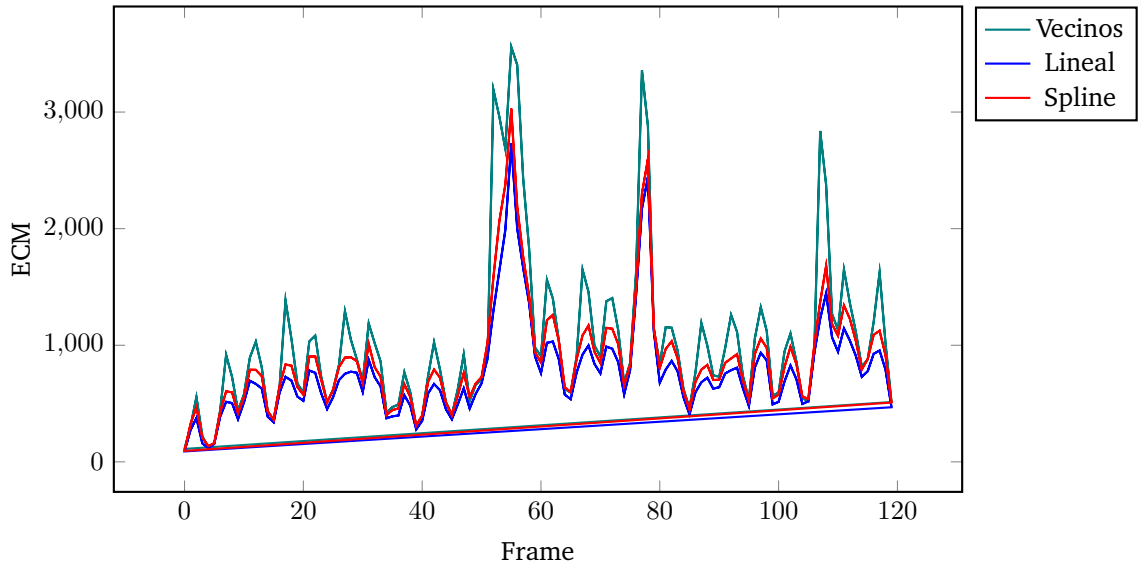


Figura 8: ECM para el video **Skate** al agregar 5 cuadros con distintos métodos de Interpolación.

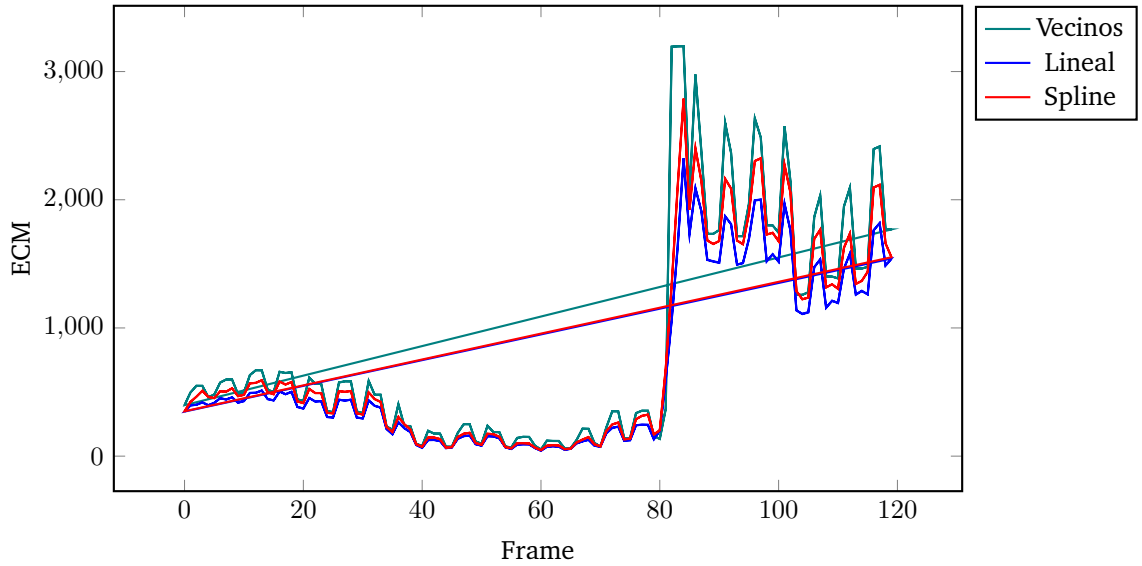


Figura 9: ECM para el video **Messi** al agregar 5 cuadros con distintos métodos de Interpolación.

Viendo las Figuras 7, 8 y 9 podemos decir que:

- El comportamiento general de los métodos para cada video no varió con respecto al escenario en el cual agregabamos 1 cuadro. Los análisis de movimiento brusco y cambios de cámaras se reafirman.
- En lineas generales, y en todos los videos, el error se incrementó con respecto al escenario en el cual agregabamos 1 cuadro, a excepción del método de Interpolación por Vecinos, que mantiene su rango de error.

4.4.4. PSNR - Agregando 5 cuadros

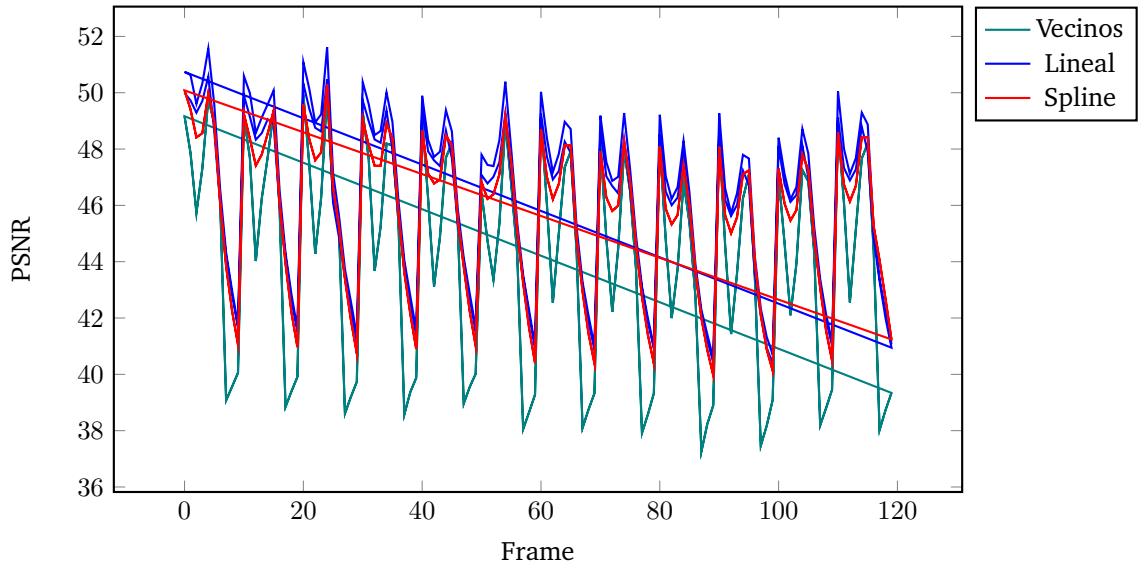


Figura 10: PSNR para el video **Amanecer** al agregar 5 cuadros con distintos métodos de Interpolación.

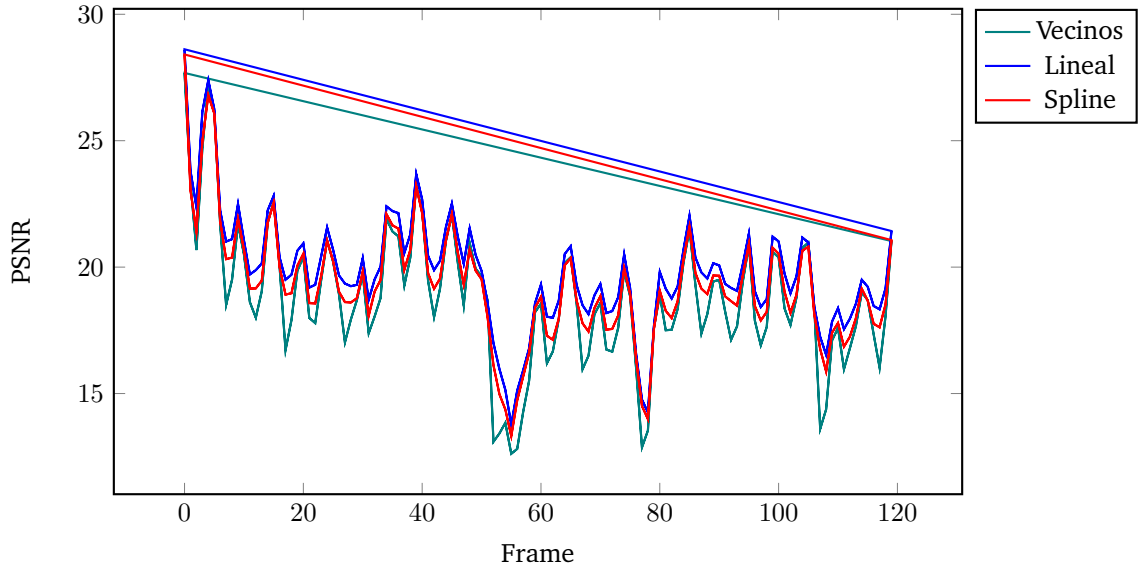


Figura 11: PSNR para el video **Skate** al agregar 5 cuadros con distintos métodos de Interpolación.

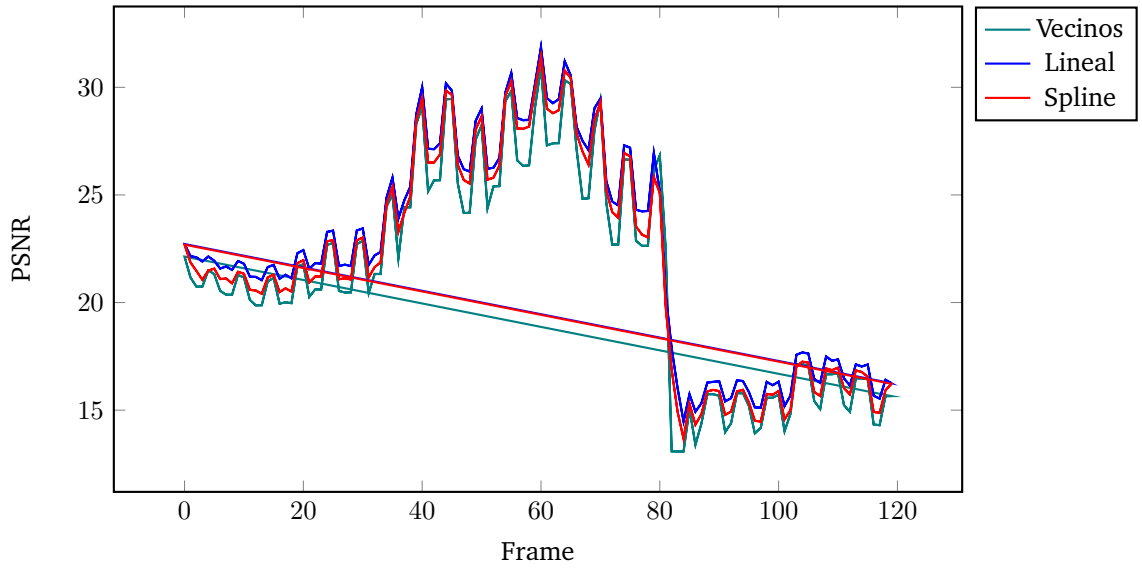


Figura 12: PSNR para el video **Messi** al agregar 5 cuadros con distintos métodos de Interpolación.

Sabiendo que, mientras más grande es el ECM más chico es el PSNR, encontramos que la información provista por este último no aporta nuevos elementos al análisis, ya que se condice con lo análizado previamente utilizando el ECM.

4.4.5. Conclusiones

En base a lo análizado, podemos concluir que:

- El error máximo cometido por el método de Interpolación por Vecinos, en comparación con los otros dos, lo hace inusable en la mayoría de los casos, y empeora a medida que agregamos más cuadros.
- Si bien en el Experimento de la Sección 4.2 pudimos lograr una mayor precisión con la Interpolación por Splines que con la Interpolación Fragmentaria Lineal, no sucedió lo mismo cuando aplicamos los métodos a la interpolación de videos. En este escenario, los métodos obtuvieron errores muy similares, siendo la Interpolación Fragmentaria Lineal mejor en algunos casos.

  Esto lo atribuimos a que los píxeles de de los videos no respetan ninguna familia de funciones, obviamente, y por lo tanto la función que infiere la Interpolación por Splines no es sustancialmente mejor que la obtenida mediante Interpolación Fragmentaria Lineal.

  Sin embargo, ya que las cotas de precisión definidas previamente no nos permitieron proyectar que la Interpolación por Splines iba a dar resultados similares a la Interpolación Fragmentaria Lineal, queda la duda de si la Interpolación por Splines de tamaño de bloque fijo puede dar mejores resultados. Para eso, haremos un experimento extra a continuación para comparar el error de dichos métodos.
- El video con movimientos suaves es mucho más sencillo de interpolar que los demás, y permite que incluso el método de Interpolación por Vecinos logre errores bajos. Al ir agregando movimientos bruscos y cambios de camaras, los videos se vuelven más difíciles de interpolar, aumentando el error obtenido con los distintos métodos.

4.4.6. Interpolación por Splines: bloques de tamaño variable vs fijo

A continuación presentamos los resultados de comparar los errores de los siguientes métodos, solo para el video **Messi**:

- Interpolación por Splines *standard*, o de bloque variable.

- Interpolación por Splines (bloques de tamaño 2)
- Interpolación por Splines (bloques de tamaño 8)
- Interpolación por Splines (bloques de tamaño 32)

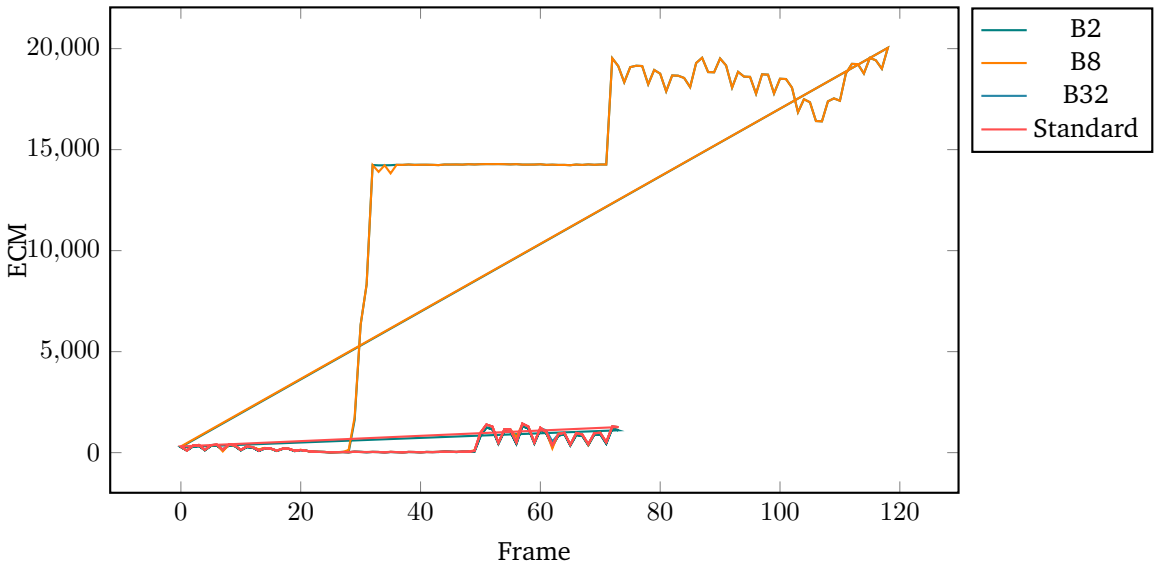


Figura 13: ECM para el video **Messi** al agregar 1 cuadro con variantes del método de Interpolación por Splines.

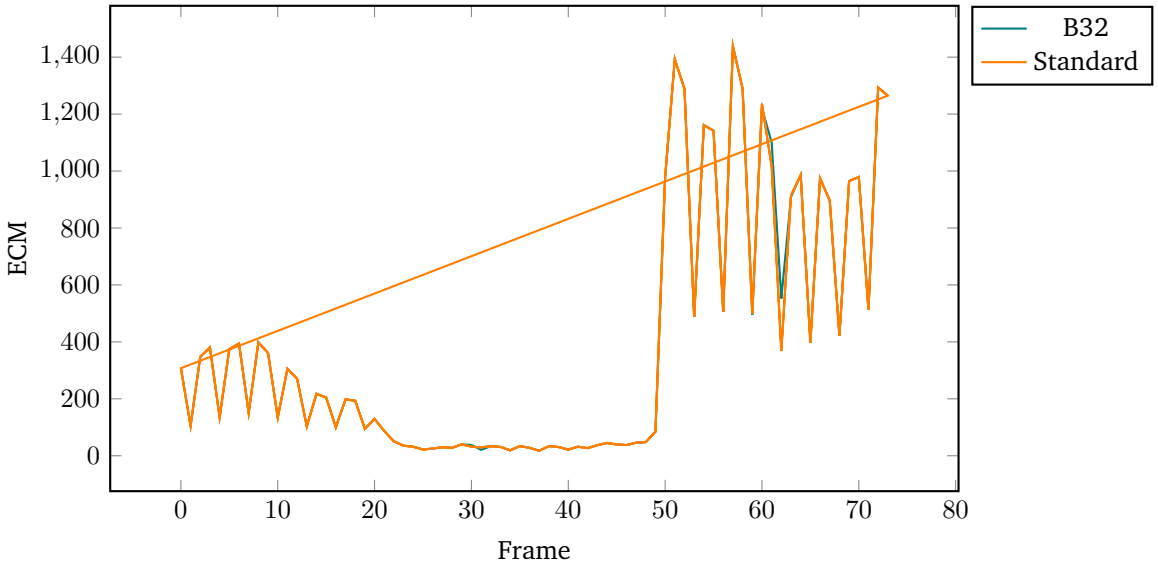


Figura 14: ECM para el video **Messi** al agregar 1 cuadro con variantes del método de Interpolación por Splines.

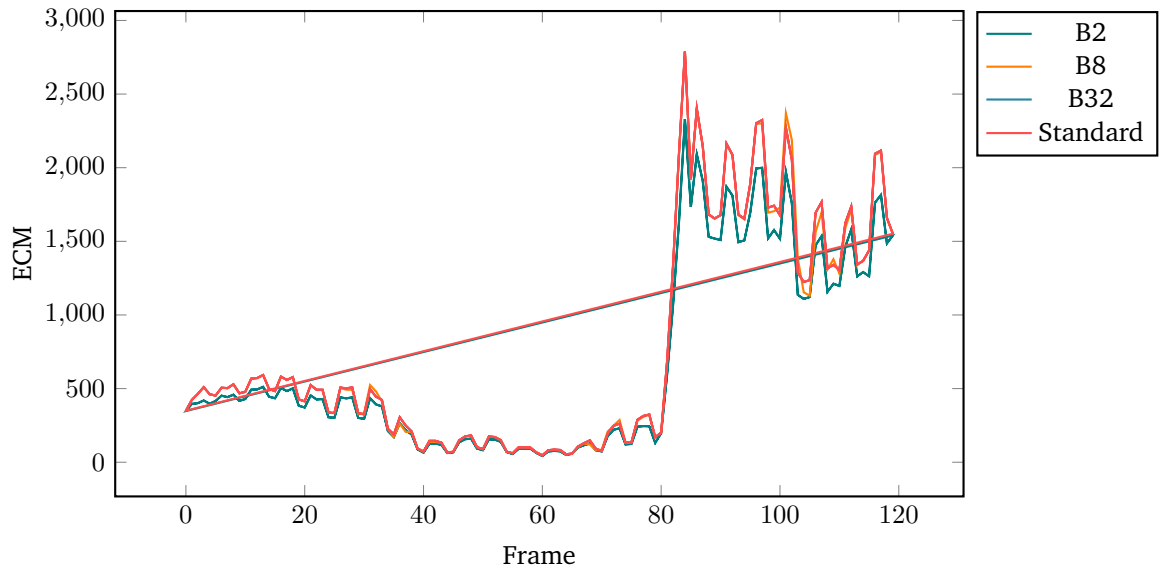


Figura 15: ECM para el video **Messi** al agregar 5 cuadros con variantes del método de Interpolación por Splines.

En base a las Figuras 13, 14 y15, podemos terminar de concluir que:

- El error del método de Interpolación por Splines usando bloques de tamaño fijo converge, al ir aumentando el tamaño del bloque, al error obtenido con la versión *standard*. Anteriormente ya habíamos confirmado un resultado parecido para diferentes familias de funciones, pero ahora podemos afirmar que esto vale también en el escenario en el cual estamos interpolando cuadros de video.
- Luego, la variante de Interpolación por Splines con tamaño fijo es similar o peor (dependiendo del tamaño del bloque) a la variante *standard*, y por lo tanto similar a la Interpolación Fragmentaria Lineal.

4.5. Análisis cualitativos de los métodos, fenómeno de artifacts.

Los *artifacts* son errores visuales resultantes de la aplicación de los métodos. Estos errores visuales se caracterizan por romper la coherencia entre imágenes al generar distorsiones evidentes.

4.5.1. Artifacts: Movimientos Bruscos

4.5.2. Artifacts: Cambios de Camara

4.5.3. Artifacts: Movimientos Armonicos

5. Conclusión

6. Referencias

7. Enunciado