# Laboratorio de Métodos Numéricos

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

## Trabajo Práctico Número 2

*Un juego de niños*



| Integrante | LU | Correo electrónico |
|---|---|---|
| Ciruelos Rodríguez, Gonzalo | 063/14 | gonzalo.ciruelos@gmail.com |
| Costa, Manuel José Joaquín | 035/14 | manuc94@hotmail.com |
| Gatti, Mathias Nicolás | 477/14 | mathigatti@gmail.com |



asdf

**key1** **key2** **key3** **key4**

# Índice

# 1. Introducción teórica

El objetivo del presente informe es resolver un problema práctico mediante el modelado matemático del mismo. Este problema consiste en generar videos en camará lenta dados los videos originales, utilizando métodos de interpolación numérica.

Por lo tanto, vamos a tener que colocar más cuadros entre cada par de cuadros consecutivos del video original. Para esto, vamos a pensar un pixel del video a lo largo de todos los frames.

$$p_{ij}(f)$$

Donde el pixel (ij-ésimo) de un video depende del frame. Entonces, la idea va a ser interpolar esta función para obtener los potenciales valores intermedios.

Entonces, por ejemplo, si tenemos un video de 1x1 píxeles, y los valores

$$p_{11}(0), p_{11}(1), ..., p_{11}(n)$$

Si nos piden colocar 2 nuevos frames entre cada frame viejo, vamos a estar interesado en los valores

$$p_{11}\left(0\right), p_{11}\left(\tfrac{1}{3}\right), p_{11}\left(\tfrac{2}{3}\right), p_{11}\left(1\right), p_{11}\left(\tfrac{4}{3}\right), p_{11}\left(\tfrac{5}{3}\right), p_{11}\left(2\right), ...,$$
$$p_{11}\left(n-1\right), p_{11}\left(n-\tfrac{2}{3}\right), p_{11}\left(n-\tfrac{1}{3}\right), p_{11}\left(n\right)$$

Entonces la idea es fijar un algoritmo de interpolación y obtener todos esos valores, para luego poder rearmar el video.

Estos métodos, además de permitirnos realizar cámaras lentas, también permiten realizar interpolación para otros fines. Por ejemplo, al streamear un video, podríamos bien no recibir todos los cuadros del video, y que el reproductor los interpole acordemente para generar un video fluido. Otro ejemplo de uso es una cámara – de mala calidad – que no puede filmar a 24 cuadros por segundo, pero que sin embargo con estos algoritmos, a partir de videos de menor framerate, podemos generear videos fluidos.

# 2. Desarrollo

## 2.1. Convenciones

## 2.2. Métodos numéricos usados

Como dijimos en la introducción, nuestro objetivo será, dada una lista de valores que toma un pixel a lo largo de diferentes frames, obtener una función que los interpole y luego usarla para obtener valores intermedios.

Dados,

$$p_0, p_1, p_2, ..., p_n$$

Queremos obtener

$$p_{ij}(f)$$

Tal que

$$p_{ij}(0) = p_0, p_{ij}(1) = p_1, p_{ij}(2) = p_2, ..., p_{ij}(n) = p_n$$

Y el resto de los valores intermedios se obtienen usando $p_{ij}$. Entonces, si nos piden colocar $k$ cuadros entre 2 cuadros existentes (supongamos cuadro 0 y cuadro 1 por simplicidad), debemos computar los siguientes valores:

$$p_{ij}\left(\frac{1}{k+1}\right), p_{ij}\left(\frac{2}{k+1}\right), p_{ij}\left(\frac{3}{k+1}\right), ..., p_{ij}\left(\frac{k}{k+1}\right)$$

Y análogamente para el resto de los cuadros.

Ahora pasemos a ver los métodos implementados y analizados en este trabajo.

### 2.2.1. Vecino más cercano

El primer método, llamado vecino más cercano o *nearest neighbour* en inglés, se basa en interpolar un punto por el valor del punto más cercano conocido. Formalmente, si $(x_1, y_1), ..., (x_n, y_n)$ son nuestros puntos de dato, la función que interpola sería:

$$f(x) = f(\arg\min_i |x - x_i|)$$

$$f(x_i) = y_i$$

En el caso particular de este problema, es más simple aún. Si tenemos que colocar $k$ cuadros entre 2 cuadros consecutivos $c_1$ y $c_2$, simplemente podemos poner $\frac{k}{2}$ cuadros con el valor de $c_1$ y luevo $\frac{k}{2}$ cuadros con el valor de $c_2$.

En el caso de que $k$ sea impar, simplemente tenemos un valor en el medio al que podemos darle cualquiera de los 2 valores, es una decisión de diseño. En el caso de este trabajo, ese valor es $c_2$

### 2.2.2. Interpolación lineal

La interpolación lineal (de a trozos) es la interpolación polinomial de a trozos más simple. Consiste en unir una serie de puntos

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2)), ..., (x_n, f(x_n))$$

por lineas rectas. Una desventaja de este método, como analizaremos en la experimentación, es que no necesariamente la función resultante va a ser diferenciable en los $x_i$, algo generalmente deseable, sobre todo en este caso en que la no diferenciabilidad se vería reflejada en cambios bruscos en el video.

En general, dados los puntos $(x_i, f(x_i)), (x_{i+1}, f(x_{i+1}))$, si queremos interpolarlos con una recta obtenemos

$$l_i(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i)$$

Entonces, el algoritmo se basa en esa fórmula, evaluándola en los puntos que se nombraron anteriormente.

### 2.2.3. Splines cúbicos

Los splines cúbicos son una interpolación polinómica (cúbica) de a trozos, que se basa en pedirle condiciones fuertes a los polinomios cúbicos que interpolan cada trozo de la función. En el caso del spline natural, si le llamamos $S_i$ al $i$ésimo spline cubico,

1. $S_i(x_i) = S_i(x_{i+1})$, es decir, que la función resultante sea continua en todo punto.
2. $S'_i(x_i) = S'_i(x_{i+1})$, es decir, que la función resultante sea derivable en todo punto.
3. $S''_i(x_i) = S''_i(x_{i+1})$, es decir, que la función resultante sea segundo derivable en todo punto.
4. $S''_0(x_0) = S''_{n-1}(x_n) = 0$, porque esta es la condición del spline natural.

Este método obviamente permite obtener como resultado una función mas apropiada, a priori, para nuestro problema, dado que la derivabilidad nos garantiza que las transiciones van a ser más suaves que con la interpolación lineal.

En el caso de este trabajo, se requirió que la cantidad de cuadros que se tienen en cuenta para hacer el spline sea fija y a elección del usuario. Analizaremos las implicaciones de esto en la experimentación.

Para una explicación más profunda del método, se puede consultar [BF11]. Además, la implementación de splines de este trabajo también esta basada en [BF11], posee algunas correcciones y está modificada para permitir realizar la interpolación de a bloques de tamaño fijo.

## 2.3. Estructuración del código

Para el modelado del problema utilizamos una interfaz muy simple. El archivo `tp-main.cpp` se ocupa de leer el todo el input, llamar a la rutina interpoladora correspondiente para cada pixel, y luego praparar el resultado para finalmente escribirlo en el archivo de output.

Lo que hacemos es tener una variable `frames` de tipo `std::vector <std::vector<unsigned int > >`, en la que el vector `frames[k]` representa los valores del pixel $k$ en cada frame(pensamos al video como una tira de pixeles en lugar de una matriz de pixeles).

Entonces, simplemente, para cada $k$ entre 0 y $ancho * alto$, tenemos un vector $frames[k]$, que podemos pensar como una funcion $k \mapsto$`frames[`$k$`]`; y esta función será la que interpolemos, como explicamos anteriormente.

Ahora pasaremos a describir brevemente los métodos utilizados.

#### 2.3.1. Nearest Neighbour

```
vector<unsigned int> nn(vector<unsigned int> valores, int cuadros)
```

- valores será, como dijimos antes, `frames[`$k$`]`, para algun $k$.
- cuadros serán la cantidad de valores nuevos a colocar entre dos valores consecutivos de valores.

El código es muy simple, y respeta lo explicado anteriormente cuando se describió el método

#### 2.3.2. Interpolación Lineal Fragmentaria

```
vector<unsigned int> lineal(vector<unsigned int> valores, int cuadros)
```

- valores será `frames[`$k$`]`, para algun $k$.
- cuadros serán la cantidad de valores nuevos a colocar entre dos valores consecutivos de valores.

Como dijimos anteriormente, la interpolación lineal fragmentaria se basa en, dados los puntos $(a, f(a)), (b, f(b))$, interpolar entre ellos con

$$\frac{f(b) - f(a)}{b - a}(x - a) + f(a)$$

En este caso, la formula para un $k$ dado sería,

$$\frac{valores[k+1] - valores[k]}{k+1-k}(x-k)+valores[k] = (valores[k+1]-valores[k])(x-k)+valores[k]$$

Entonces, hacemos como explicamos anteriormente, y en el caso de que $cuadros = c$, basta con tomar los siguientes valores para $x - k$:

$$\frac{1}{c+1}, \frac{2}{c+1}, ..., \frac{c}{c+1}$$

### 2.3.3. Splines cúbicos

`vector<unsigned int> splines(vector<unsigned int> valores, int cuadros, int radio)`

- valores será `frames[`$k$`]`, para algun $k$.
- cuadros serán la cantidad de valores nuevos a colocar entre dos valores consecutivos de valores.
- radio será la cantidad de cuadros que contendrá cada bloque. No es necesario que radio divida a la cantidad de cuadros, en ese caso el ultimo bloque simplemente tendrá menos cuadros (sí es necesario que el último bloque tenga más de 3 cuadros, para poder resolver el sistema).

La función splines simplemente lo que hace es partir al video en bloques, y para cada bloque generar el spline correspondiente e interpolar allí utilizandolo. Para llevar esto a cabo, se utilizará la funcion:

`void splines_bloque(vector<double>ys, int cuadros, vector<double>* resultado)`

Esta función se ocupará de obtener el spline que interpola a los valores de **ys** utilizando el algoritmo de [BF11] modificado para que funcione correctamente. Luego, hara **push_back** de los resultados en **resultado**.

## 2.4. Experimentación

La experimentación del presente trabajo se divide a grandes rasgos en tres partes:

- Primero nos ocuparemos de las mediciones de tiempos y análisis de complejidad de los algoritmos y métodos utilizados.
- Luego, nos centraremos en lo que concierne la medición del error de los resultados obtenidos a partir de los métodos, desde un punto de vista cuantitativo.
- Finalmente, nos centraremos en un análisis cualitativo de los resultados, intentando analizar subjetivamente los resultados obtenidos, especialmente buscando *artifacts* o errores que la simple medición del error numérico no nos permite percibir.

En cada parte explicaremos la metodología con la que realizamos los experimentos correspondientes, además de analizar en profundidad los resultados obtenidos.

## 3. Resultados y discusión

# 4. Conclusiones

# 5. Apéndices

# Referencias

[BF11] R. Burden y D. Faires. *Numerical Analysis*. Brooks/Cole, 2011.