PSYCO PSEUDO PIXELA FILTERS

IMPLEMENTACIÓN DE LOS FILTROS POLIFÁSICOS DE ANÁLISIS Y SÍNTESIS SEGÚN LA NORMATIVA

ISO_IEC_11172-3_1993

(MPEG I, LAYER III)

Julián Esteban Nieto Díaz

Christian Rafael Mora Parga

Julián Guillermo Rodríguez Mora

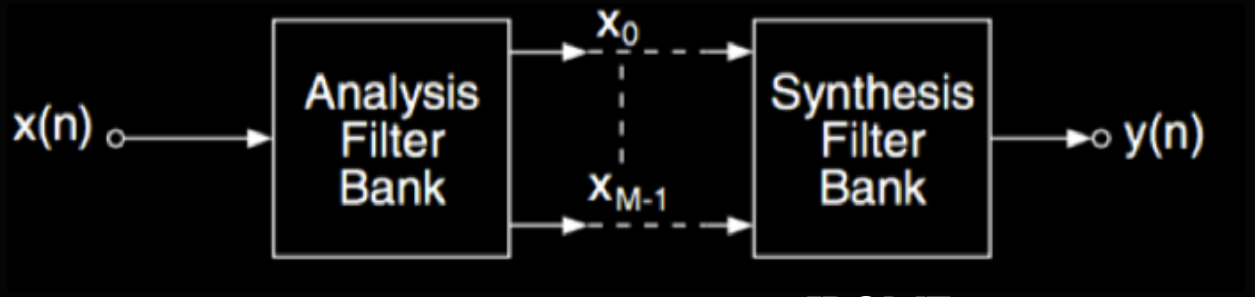
Universidad de San Buenaventura Bogotá

Ingeniería de Sonido

Procesamiento Digital de Señales, 2019-1

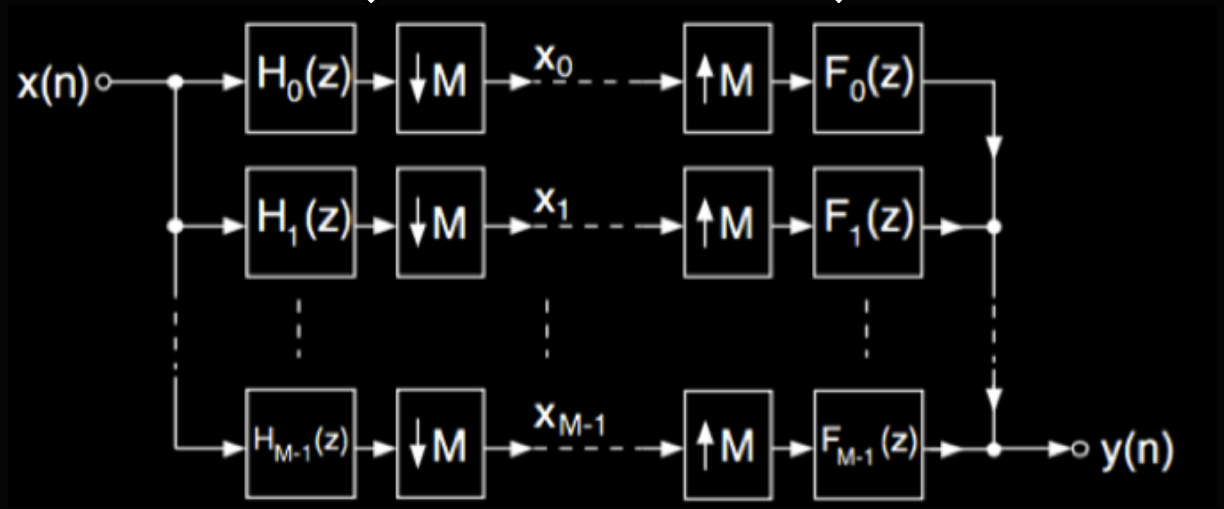
INTRODUCCIÓN

Los filtros polifásicos, mejor conocidos como PQMF (Pseudo Quadrature Mirror Filters) y su inverso IPQMF, son un banco de filtros que pretenden descomponer en 32 bandas para *n* frames de tiempo, señales de audio. Todo esto con el propósito de hacer una posterior discriminación de alocación en bits a partir de un modelo psicoacústico.

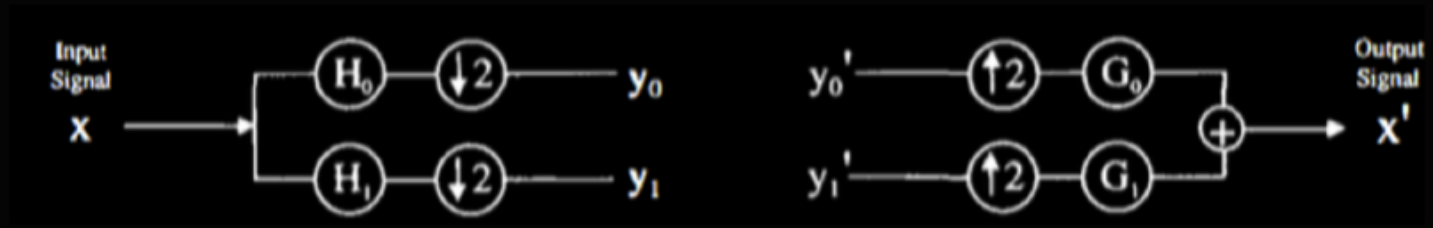


PQMF IPQMF

"Digital Audio Signal Processing" U. Zölzer, 2008, p. 287.



Par de Filtros Espejo de Cuadratura



Marina & Richard, 2003

-La resultante del overlap-add para la reconstrucción de $x[n]$ es $x'[n]$

$$TZ\{x'[n]\} = X'(z) = Y_0(z^2)G_0(z) + Y_1(z^2)G_1(z)$$

Condiciones de simetría (espejo) para ambos filtros:

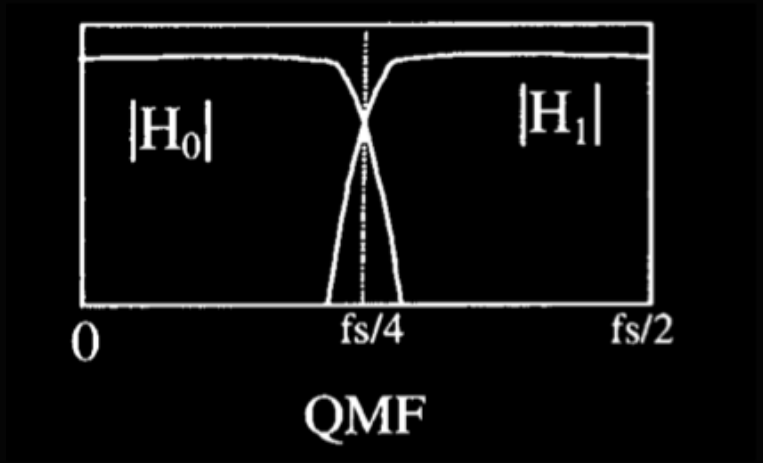
$$\pm H_0(\pm z) = \mp H_1(\mp z)$$

$$G_0(z) = -H_1(-z), \qquad G_1(z) = H_0(-z)$$

Eventualmente se llega a:

$$X'(z) = \frac{1}{2}\left(H_0^2(z) - H_0^2(-z)\right)X(z)$$

$h_{0,1}[n]$: filtros de análisis
$g_{0,1}[n]$: filtros de síntesis (reconstrucción)



Marina & Richard, 2003

Se requiere que:

$$H_0^2(z) - H_0^2(-z) = 2z^{-l}$$

Se logra una "reconstrucción perfecta" visualisable en la forma de un FIR de orden l de un único retraso.

$$X'(z) = X(z)z^{-l} \rightarrow x'[n] = x[n-l]$$

FILTROS ESPEJO DE PSEUDO CUADRATURA (PQMF)

SECCIÓN DE ANÁLISIS:

Down sampling (decimación) a 32:

$$\big((h[k] * x) \downarrow 32\big)[n]$$

Convolución con cada filtro de análisis h_k:

$$y[k,n] = h[k,n] * x[n], \qquad k = 0, 1, \dots, 31$$

Suponiendo 512 muestras de audio, se filtra en 32 bandas, y la resultante total de $y[k,n]$ organizada será:

$$y[k,n] = \left[y_0[0] \dots y_0[15], y_1[0] \dots y_1[15], \dots, y_{31}[0] \dots y_{31}[15]\right]$$

$y[k,n]$ posee 512 muestras en total, dado que está críticamente muestreada (efecto de la decimación).

Sin decimación ésta sería :
$y[k,n] = \left[y_0[0] \dots y_0[511], y_1[0] \dots y_1[511], \dots, y_{31}[0] \dots y_{31}[511]\right]$

SECCIÓN DE SÍNTESIS

Up sampling (interpolación) a 32:
Se insertan ceros entre las muestras de $y_k[n]$

$$\big((y[k]) \uparrow 32\big)[n]$$

De-convolución con cada filtro de síntesis (reconstrucción) g_k:

$$y[k,n] * g[k,n]$$

Overlap-add:

$$x[n] = \sum_{k=0}^{31} y[k,n] * g[k,n]$$

Modulación del Filtro Base H_0 para obtener el Banco de Filtros H_k

-Convolución de $x[n]$, con filtros de análisis $h_k[n]$

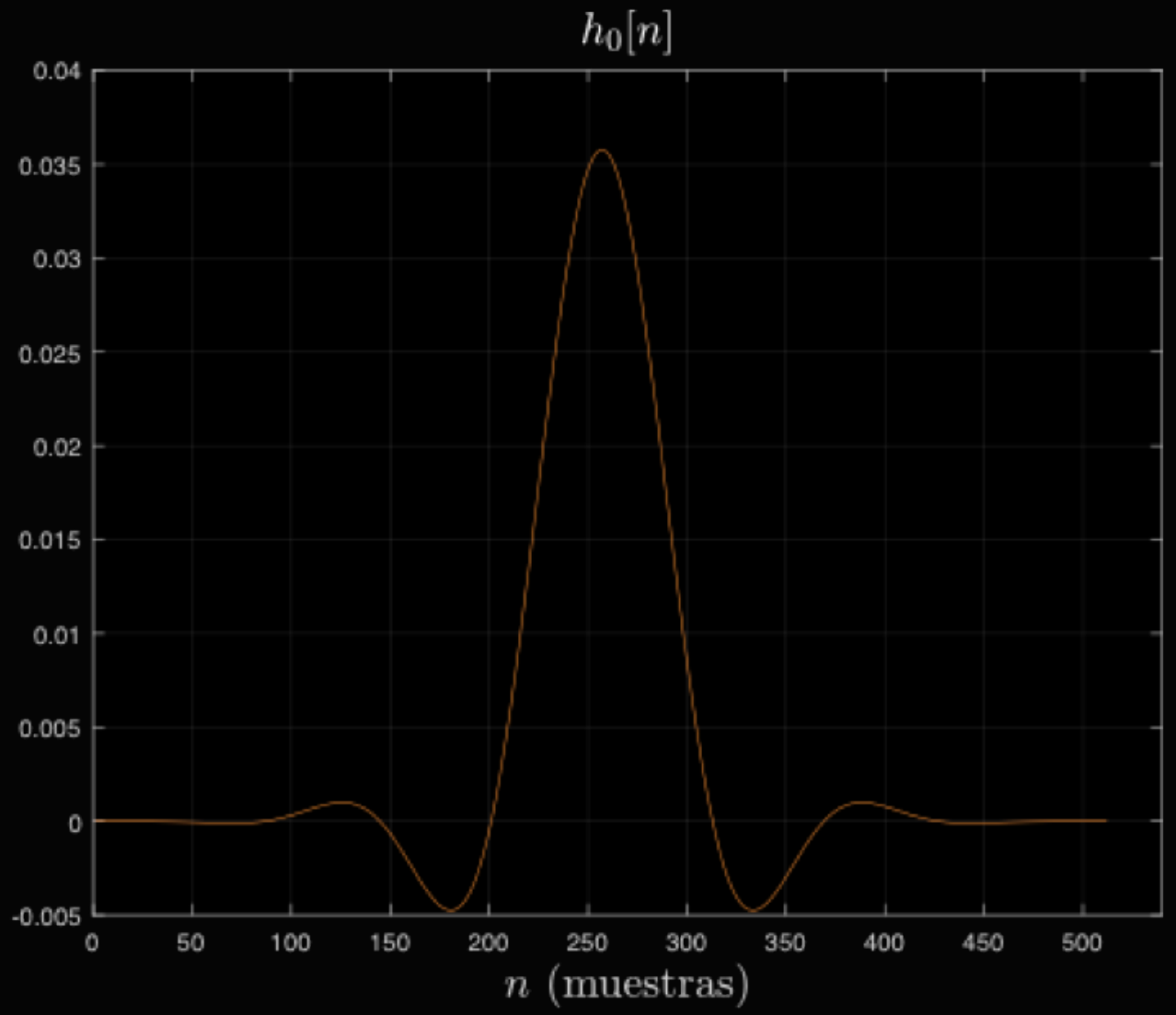
$h_k[n] * x[n],\ k = 0, 1, \dots, 31$

-Modulación de un filtro base de análisis $h_0[n]$ para obtener los distintos $h_k[n]$

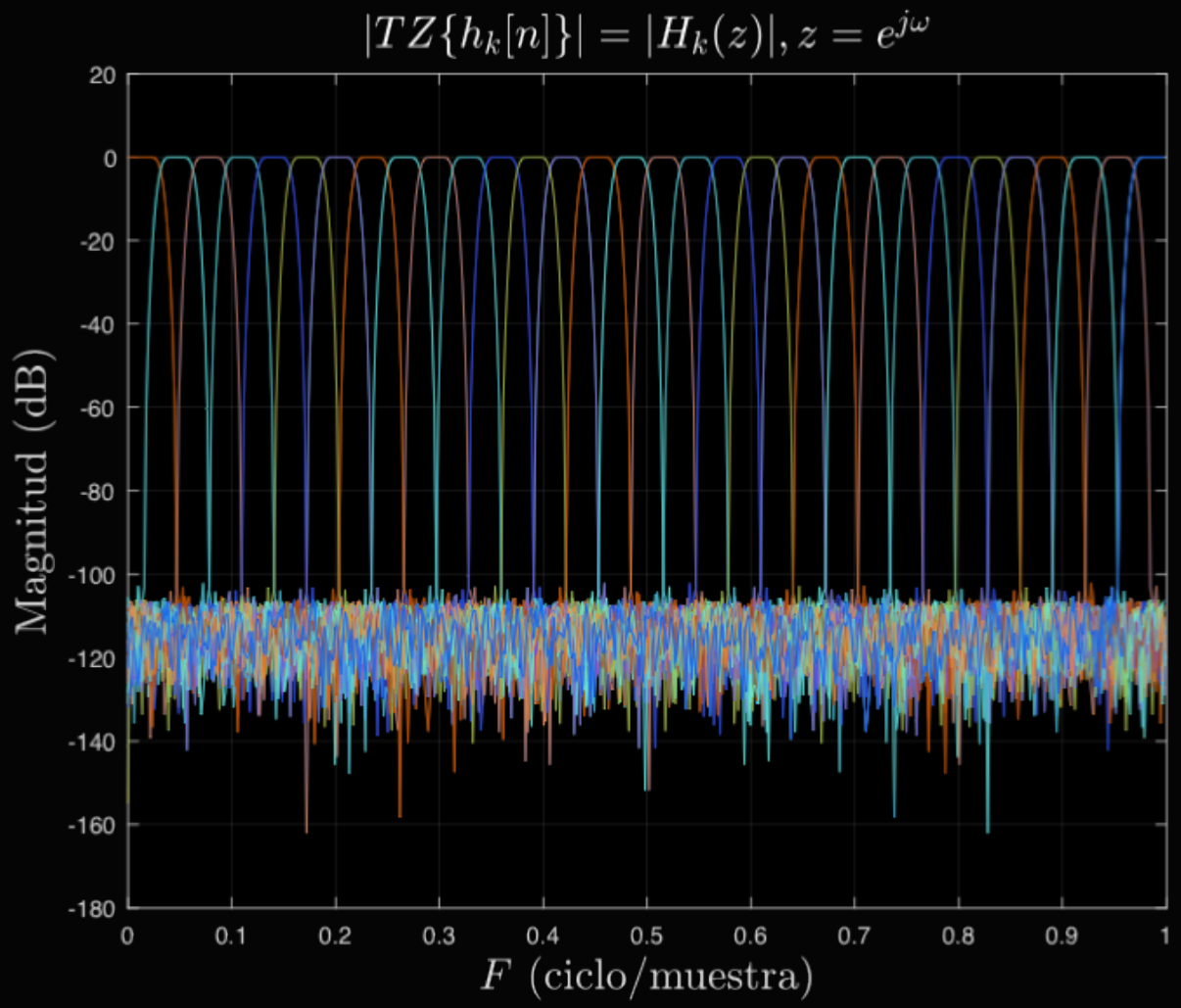
$h_k[n] = h_0[n] \cos\left[(2k+1)(n-16)\frac{\pi}{64}\right],$
$k = 0,1, \dots, 31, \quad n = 0,1, \dots, 511$

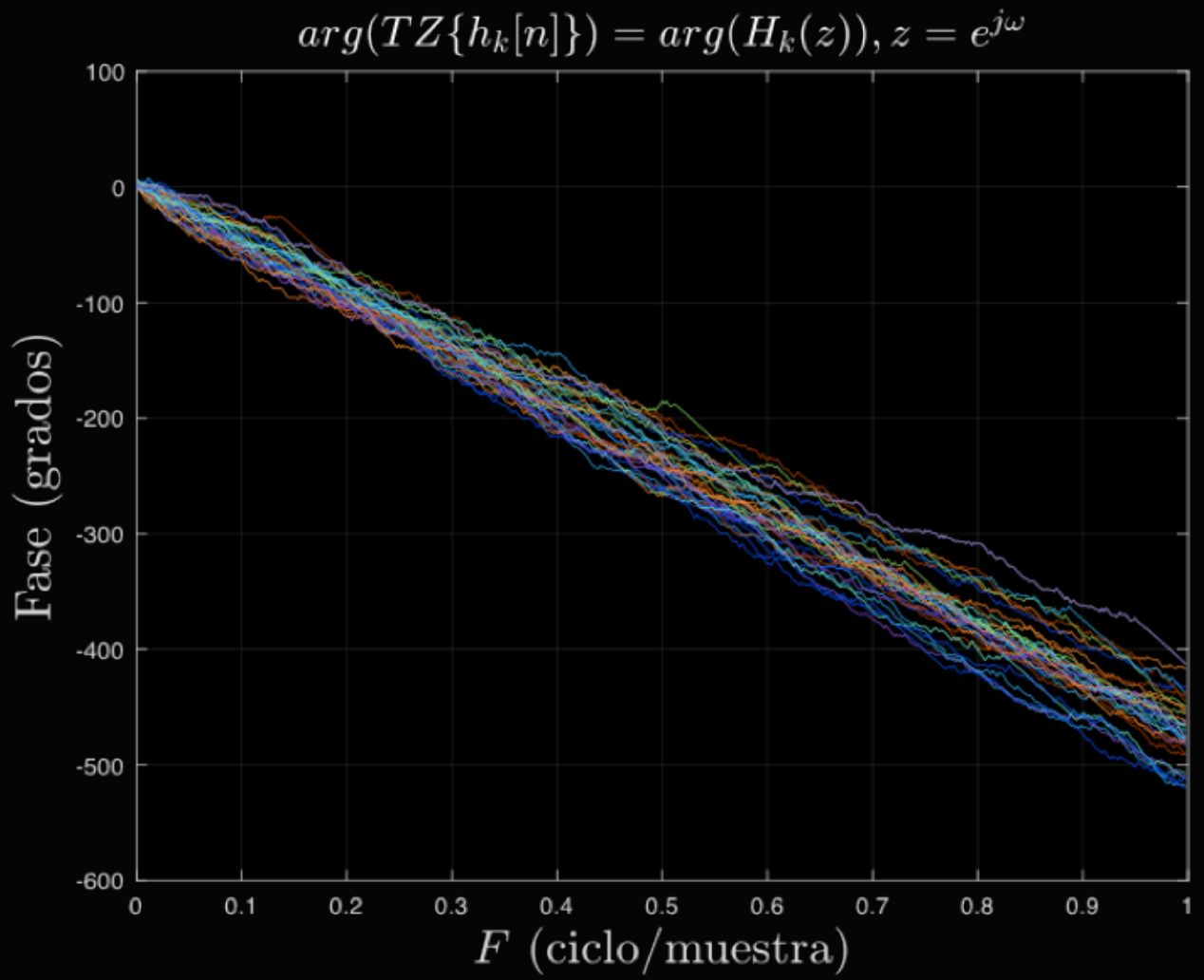
-Traslapo entre bandas de 512/32 = 16 muestras, genera un hop-size en el espectro digital de 32 muestras.

De igual forma $f_N/32 = 689$ Hz



$$h_k[n] = h_0[n]\cos\left[(2k+1)(n-16)\frac{\pi}{64}\right],\ k = 0,1,\ldots,31, \qquad n = 0,1,\ldots,511$$





Implementación teórica del algoritmo

Expresión analítica temporal del procedimiento:

$$y_k[n] = \sum_{m=0}^{511} h_k[m]x[32n - m], \qquad k \in \mathbb{Z}, \qquad k = 0, 1, \dots, 32, \qquad \text{y} \qquad n, m \in \mathbb{Z}, \qquad n, m = 0, 1, \dots, 511$$

Un implementación directa de la formulación anterior implica; 16384 multiplicaciones combinadas, con sumas, tan solo para entregar 32 salidas de $y_k[n]$ para cada bloque de 32 muestras de la señal de entrada $x[n]$.

Expresión analítica temporal del procedimiento:

$$y_k[n] = \sum_{m=0}^{511} h_k[m]x[32n-m], \qquad k \in \mathbb{Z}, \qquad k = 0,1,\dots,32, \qquad \text{y} \qquad n,m \in \mathbb{Z}, \qquad n,m = 0,1,\dots,511$$

Optimización para $y_k[n]$:

$$y_k[n] = \sum_{r=0}^{63}\sum_{q=0}^{7} h_k[64q+r]\cdot x[32n-64q-r], \qquad q,r \in \mathbb{Z}, \qquad q = 0,1,\dots,7, \qquad r = 0,1,\dots,63.$$

Matriz moduladora: $M[k,r] = \cos\left[\frac{(2k+1)(r-16)\pi}{64}\right]$, $k = 0,1,\dots,31, \qquad r = 0,1,\dots,64, \qquad q = 0,1,\dots,7$

Banco de filtros: $h_k[64q+r] = C[64q+r]\cdot M[k,r]$

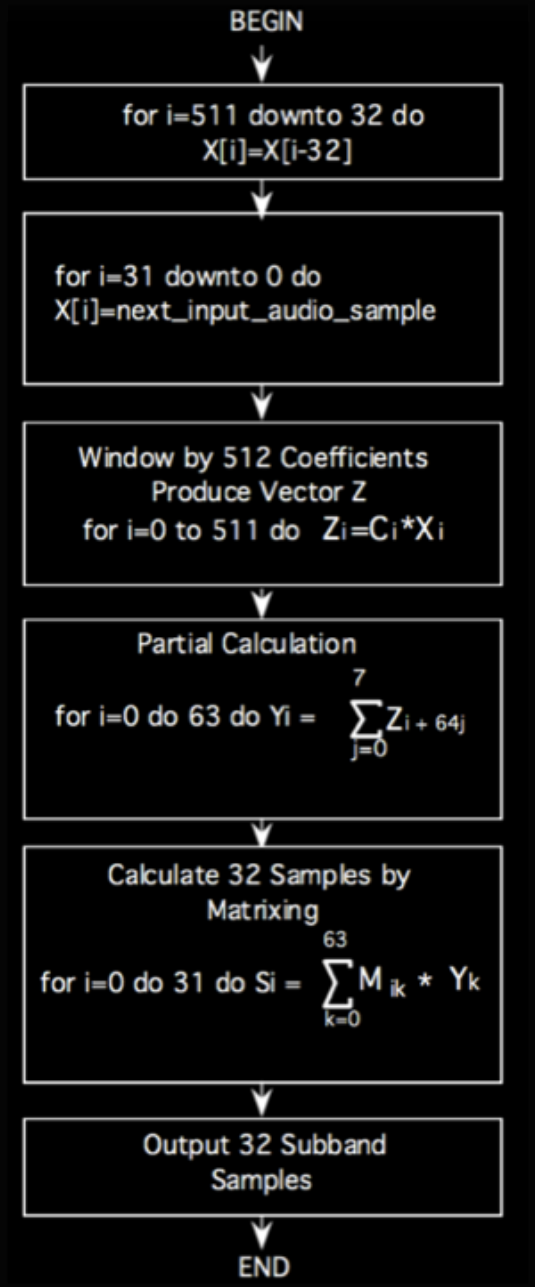
Formulación final de $y_k[n]$ que contiene 80 multiplicaciones y adiciones por muestra ($2\cdot 80\cdot 32 = 5120$ operaciones).
Se ventanea x con la ventana C, que luego será modulada por M:

$$y_k[n] = \sum_{r=0}^{63} M[k,r]\sum_{q=0}^{7} C[64q+r]\cdot x[32n-64q-r]$$

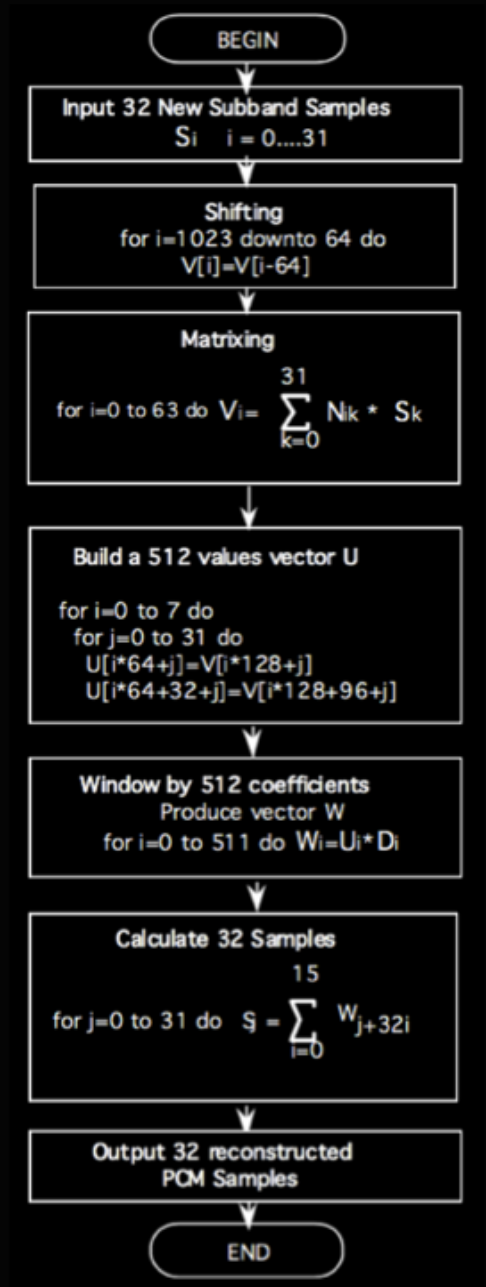
IMPLEMENTACIÓN EN JUCE

Pseudo-códigos propuestos por la normativa

PQMF:



IPQMF:



Imágenes tomadas de: ISO_IEC_11172-3_1993

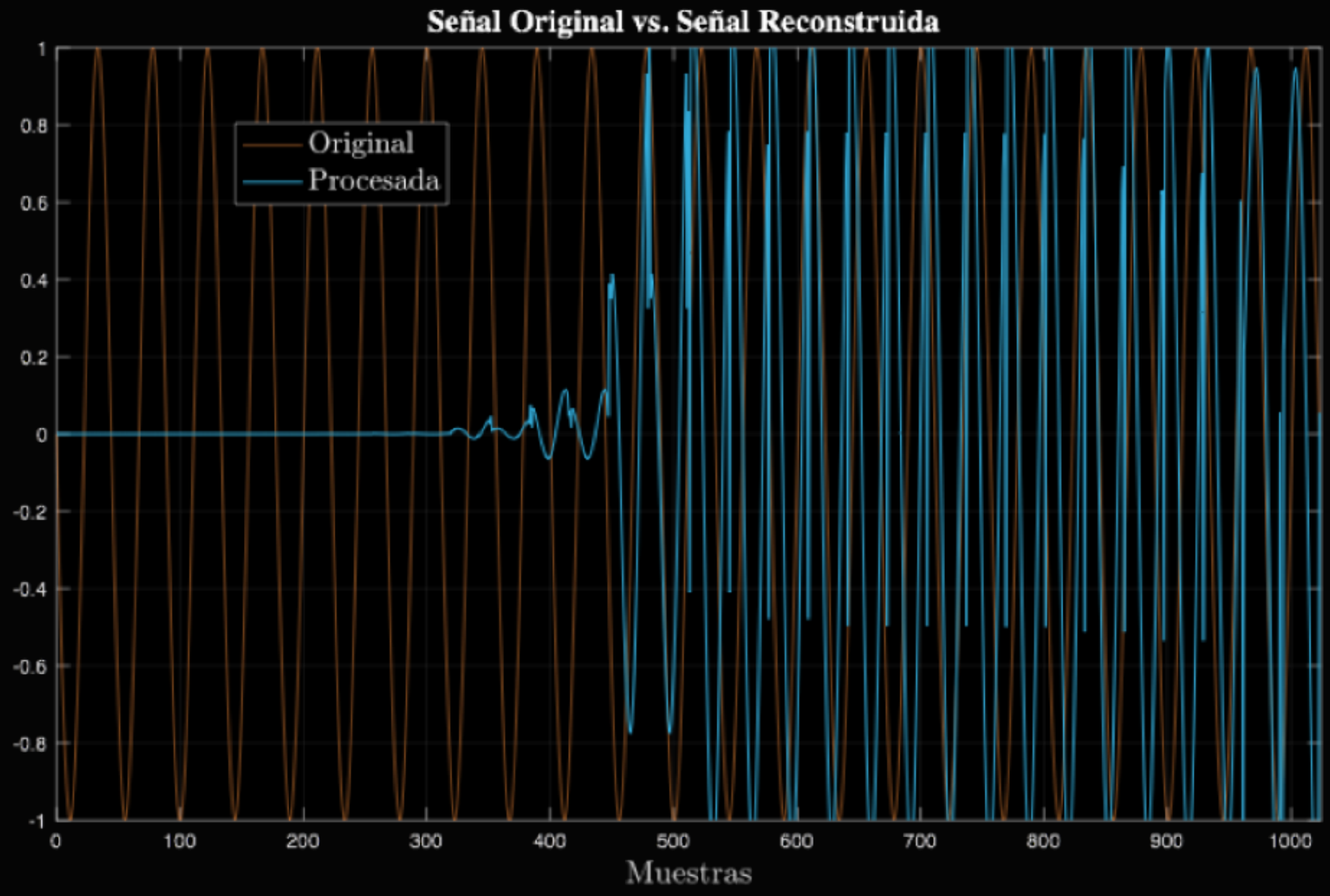
X[i]=[63, 62, 61, ... 31, 30, 29, ... 2, 1, 0, ... 0, 0, 0]
0, 1, 2, ... 32, 33, 34, ...61, 62, 63, ...509, 510, 511

Según la norma, las muestras ingresan de la primera a la última desde el primer hasta el último espacio en el arreglo de a bloques de 32 muestras.

X[i]=[0, 0, 0, ... 511, 510, 509, ... 479, 478, 477, ... 449. 448, 447]
0, 1, 2, ... 448, 447, 446, ... 480, 481, 482, ... 510, 511, 512

En el algoritmo implementado, se ingresan directamente las muestras de la última a la primera de forma inversa.

Reconstrucción de la Señal pasando por el PQMF e IPQMF sin el Algoritmo de Reintrepretación



BUFFER CIRCULAR INVERSO

```h
#include <stdio.h>
#include <iostream>
#include "math.h"

class CircularPQMF_Buffer{
public:

   CircularPQMF_Buffer();
   ~CircularPQMF_Buffer();

   void setBufferSize(int size);
   void clear();
   void addSample(float sample);

   float *buffer;

private:
   int index, i = 0;
   int bufferSize;
};
```

```cpp
void CircularPQMF_Buffer::setBufferSize(int size){
   if(buffer) {delete[]buffer;}
   bufferSize = size;
   index = bufferSize - 1;
   buffer = newfloat[bufferSize];
   clear();
}

void CircularPQMF_Buffer::clear(){
   memset(buffer, 0, bufferSize *sizeof(float));
}

void CircularPQMF_Buffer::addSample(float sample){
   buffer[index] = sample;
   if(index == 0){index = bufferSize;}
   index--;
}
```

CLASES DE PQMF E IPQMF

```cpp
#include<math.h>
#include<cstring>
#include<iostream>

extern float c[512];
```

Filtro base de análisis

```cpp
class PQMF_Analysis{
public:
   PQMF_Analysis(){
       memset(z,0,sizeof(z));
       memset(s,0,sizeof(s));
       part_sum=0;
       part_sum2=0;
       block_index=0;
       num_blocks=16;
       jj=0;

       //cálculo de la matriz moduladora M para el análisis
       for(int k = 0; k<32; k++){
           for(int r = 0; r<64; r++) {
               M_k_r[k][r] =cos((k+0.5)*(r-16)*(M_PI/32));
           }
       }
   };

   ~PQMF_Analysis(){};

   void setNumberBlocks(int);
   float *PQMF_Filtering(floatx[]);

private:
   int block_index, num_blocks, jj;
   float z[512];
   float s[64];
   float M_k_r[32][64];
   float part_sum, part_sum2;
};
```

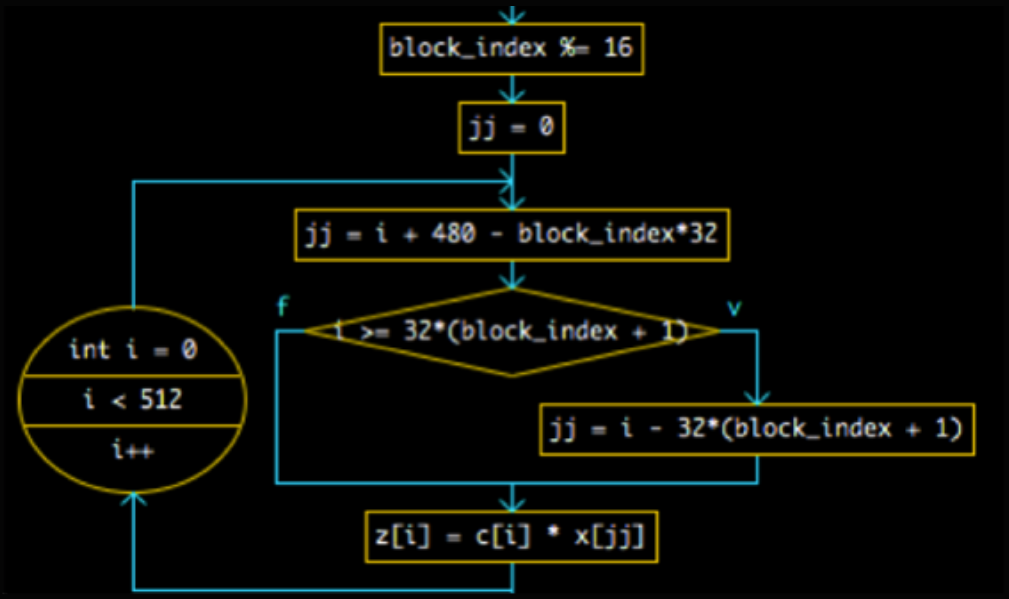
```cpp
#include<iostream>
#include<stdio.h>
#include<math.h>

extern float d[512];
```

Filtro base de síntesis

```cpp
class IPQMF_Synthesis{
public:

   IPQMF_Synthesis(){
       memset(v,0,sizeof(v));
       memset(u,0,sizeof(u));
       memset(w,0,sizeof(w));
       part_sum=0;
       part_sum2=0;

       //cálculo de la matriz moduladora N para la síntesis
       for(int k = 0; k<32; k++){
           for(int r = 0; r<64; r++) {
               N_k_r[k][r] =cos((2*k+1)*(r+16)*(M_PI/64));
           }
       }
   };

   ~IPQMF_Synthesis(){};

   float*IPQMF_Filtering(floaty[]);

private:
   float v[1024];
   float u[512];
   float w[512];
   float N_k_r[32][64];
   float part_sum, part_sum2;
};
```

PQMF

```
#include"PQMF_Analysis.h"

float *PQMF_Analysis::PQMF_Filtering(float x[]){

    block_index %= 16;

    jj = 0;
    for(inti = 0; i < 512; i++) {
        jj= i + 480 - block_index*32;
        if(i >= 32 * (block_index + 1)){
            jj = i - 32*(block_index + 1);
        }
        z[i] = c[i] * x[jj];
    }

    block_index++;

    for(auto r = 0; r < 64; r++) {
        part_sum = 0;
        for(auto j = 0; j < 8; j++){
            part_sum = part_sum + z[r + 64*j];
        }
        s[r] = part_sum;
    }

    static float y[32];

    for(auto k = 0; k < 32; k++){
        part_sum2 = 0;
        for(auto r = 0; r < 64; r++){
            part_sum2 += M_k_r[k][r] * s[r];
        }
        y[k] = part_sum2;
    }

    return y;
}
```



Reinterpretación del ingreso de las muestras de audio, para tomarlas como la norma indica.

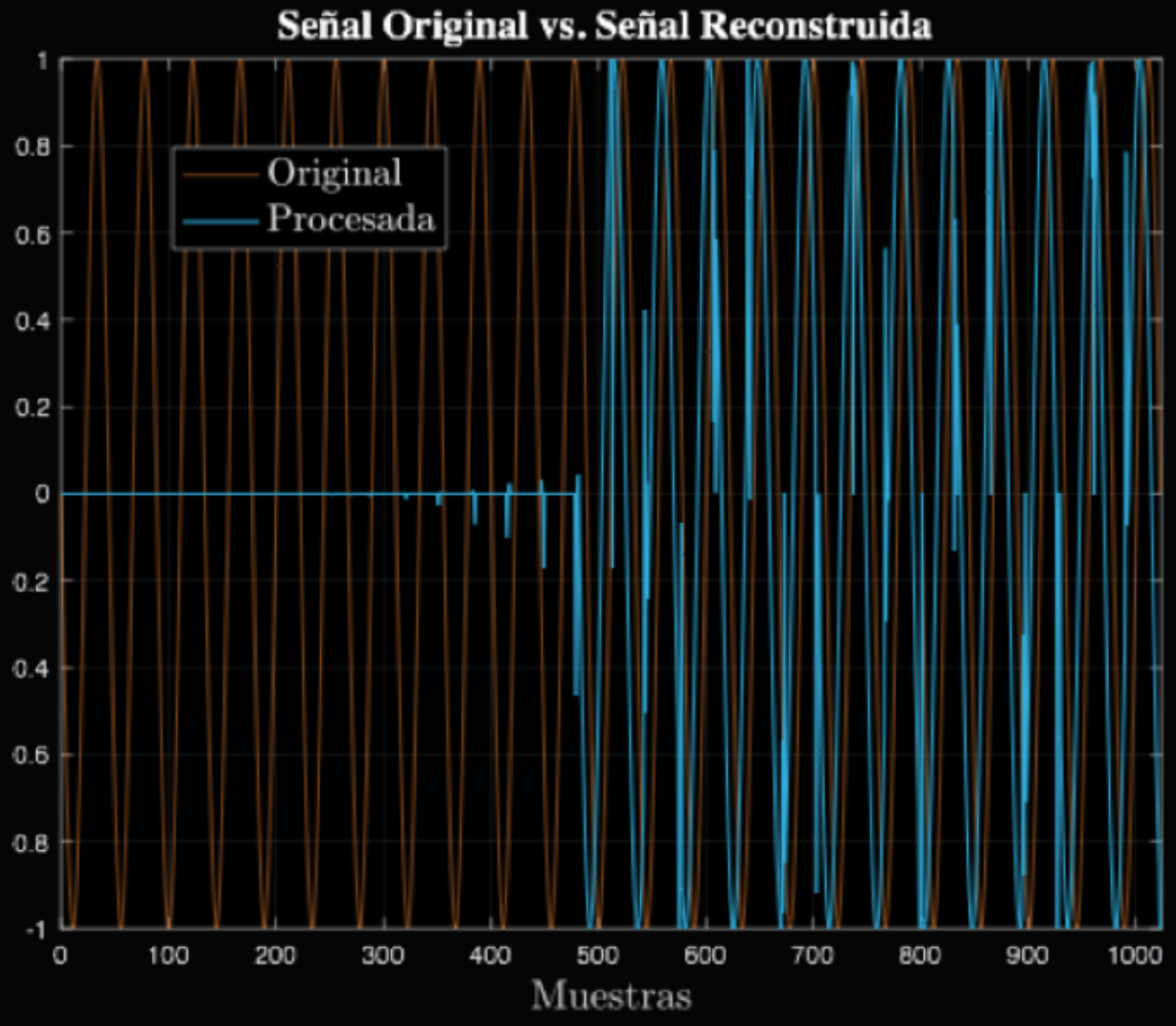
-Se van ingresando al revés 32 muestras de audio a un buffer de tamaño=512: x[n].

-Se ventanea dicho buffer x[n].

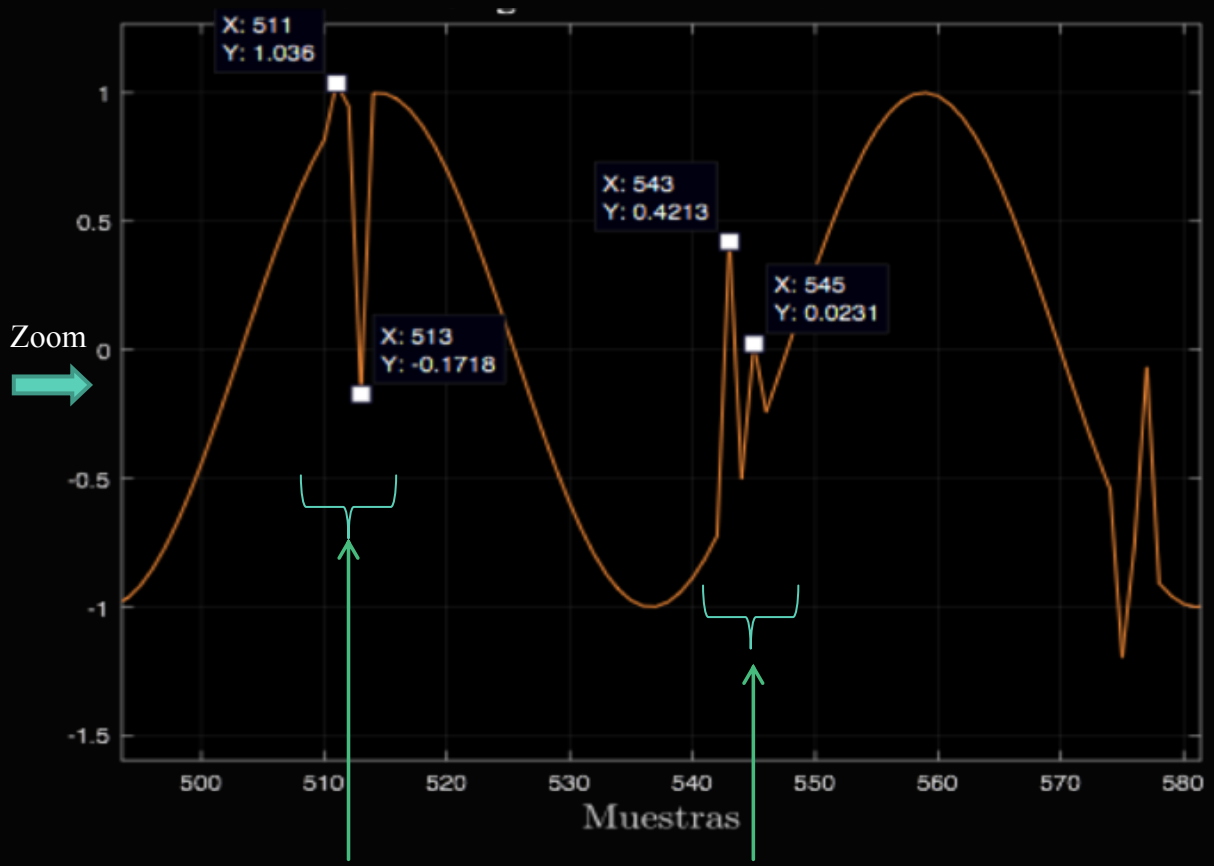
-Se modula x[n] ventaneado.

-Se entregan 32 bandas espectrales en y[k]

Reconstrucción de la señal pasando por el PQMF e IPQMF con el Algoritmo de Reinterpretación



Zoom



y[31], y[1]
Respectivamente
Bloque n

y[31+32], y[1+32]
respectivamente
Bloque n+1

IPQMF

```
float* IPQMF_Synthesis::IPQMF_Filtering(float x[]){

   for(int i = 1023; i >= 64; i--)
       v[i] = v[i - 64];

   for(auto i = 0; i < 64; i++) {
       part_sum = 0;
       for(auto k =0; k<32; k++) {
           part_sum += N_k_r[k][i] * x[k];
       }
       v[i] = part_sum;
   }

   for(int i = 0; i < 8; i++){
       for(int j = 0; j < 32; j++){
           u[64*i+j]   = v[128*i+j];
           u[64*i+32+j] = v[128*i+96+j];
       }
   }

   for(int i = 0; i < 512; i++){
       w[i] = d[i] * u[i];
   }

   static float y[32];

   for(int j = 0; j < 32; j++){
       part_sum2 = 0;
       for(autoi = 0; i<16; i++) {
           part_sum2 += w[j+32*i];
       }

       y[j] = part_sum2;
   }

   //corrección por promediación de la muestra 1
   y[1] = 0.5*(y[0] + y[2]);

   return y;
}
```

-Se ingresan 32 muestras espectrales x[k].

-Se modula x[k]

-Se ventanea con el filtro base a x[k] modulado.

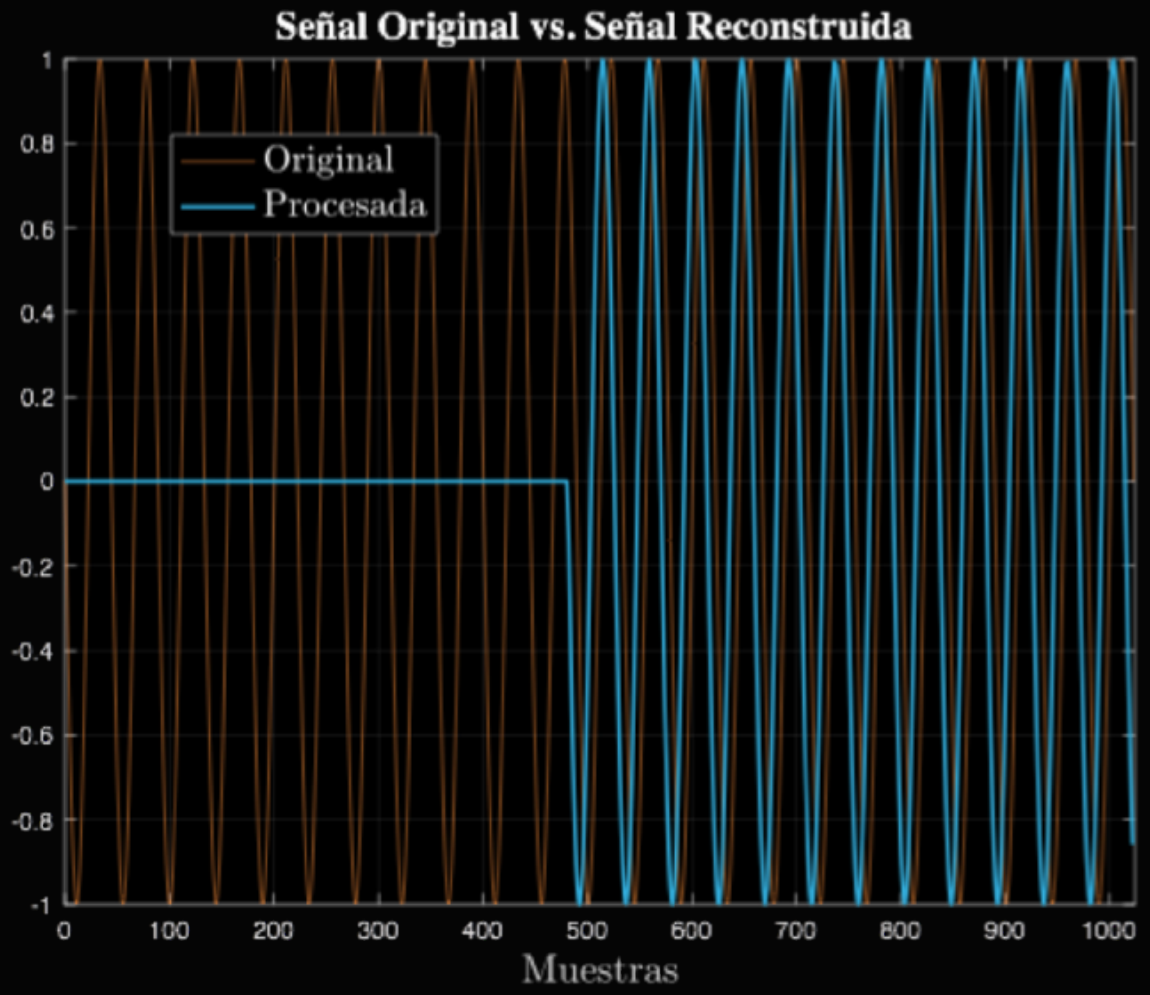
-Se entrega un arreglo y[n] de 32 muestras reconstruidas.

-Interpolación por promediación #1:
Corrección de la muestra 1, y[1]
Para cada bloque de 32 muestras

```
//Set del numero de bloques de a 32 a procesar:
       PQMF_processor.setNumberBlocks(buffer.getNumSamples()/32);

       for(intblockIndex = 0; blockIndex<buffer.getNumSamples()/32; blockIndex++){
           //Indexación de a 32 muestras en el buffer circular.
           for(int i = 0; i<32; i++)
               x.addSample(buffer.getSample(0, i + blockIndex*32));

           //Procesamiento del buffer circular con el PQMF y el IPQMF.
           output_sintesis = IPQMF_processor.IPQMF_Filtering(PQMF_processor.PQMF_Filtering(x.buffer));

           //Indexación de la señal reconstruida de a bloques de 32 hasta llenarse a bufferToFill.numSamples
           for(int n = 0; n < 32; n++)
               recons_signal[n + blockIndex*32] = *(output_sintesis+ n);
       }

       //Corrección por promediación de las últimas muestras de cada sálida y[32] de síntesis:
       for(int i = 31; i < buffer.getNumSamples(); i+=32){
           if(i == buffer.getNumSamples() - 1){
               recons_signal[i] = recons_signal[i-1];
               break;
           }else{
               recons_signal[i] = 0.5*(recons_signal[i-1] + recons_signal[i+1]);
           }
       }

        //Indexación de la señal reconstruida en el buffer de salida:
       for(int channel = 0; channel < totalNumInputChannels; ++channel){
           auto* channelData = buffer.getWritePointer(channel);
           for(auto n = 0; n < buffer.getNumSamples(); n++){
               channelData[n] = recons_signal[n];
               pushNextSampleIntoFifo(recons_signal[n]);
           }
       }
```

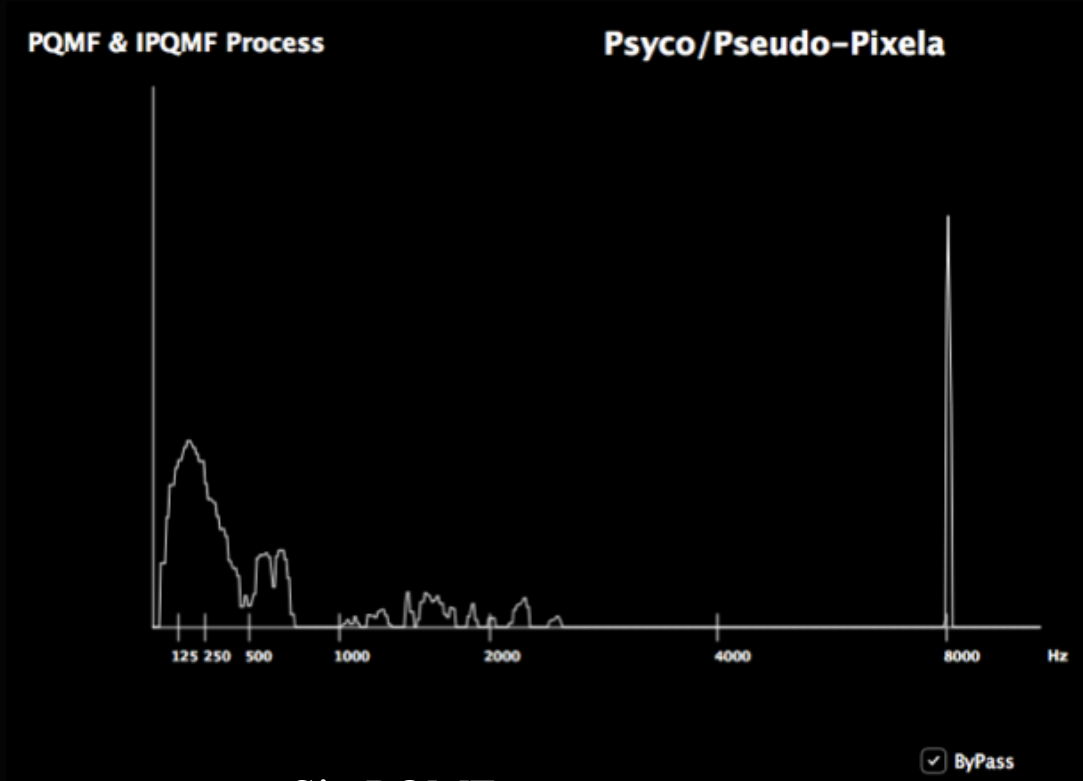
-Interpolación por promediación #2: Corrección de la muestra 31 para cada bloque de 32 muestras de cada salida de y[n]

Reconstrucción de la señal pasando por el PQMF con el algoritmo de reinterpretación y la interpolación completa.

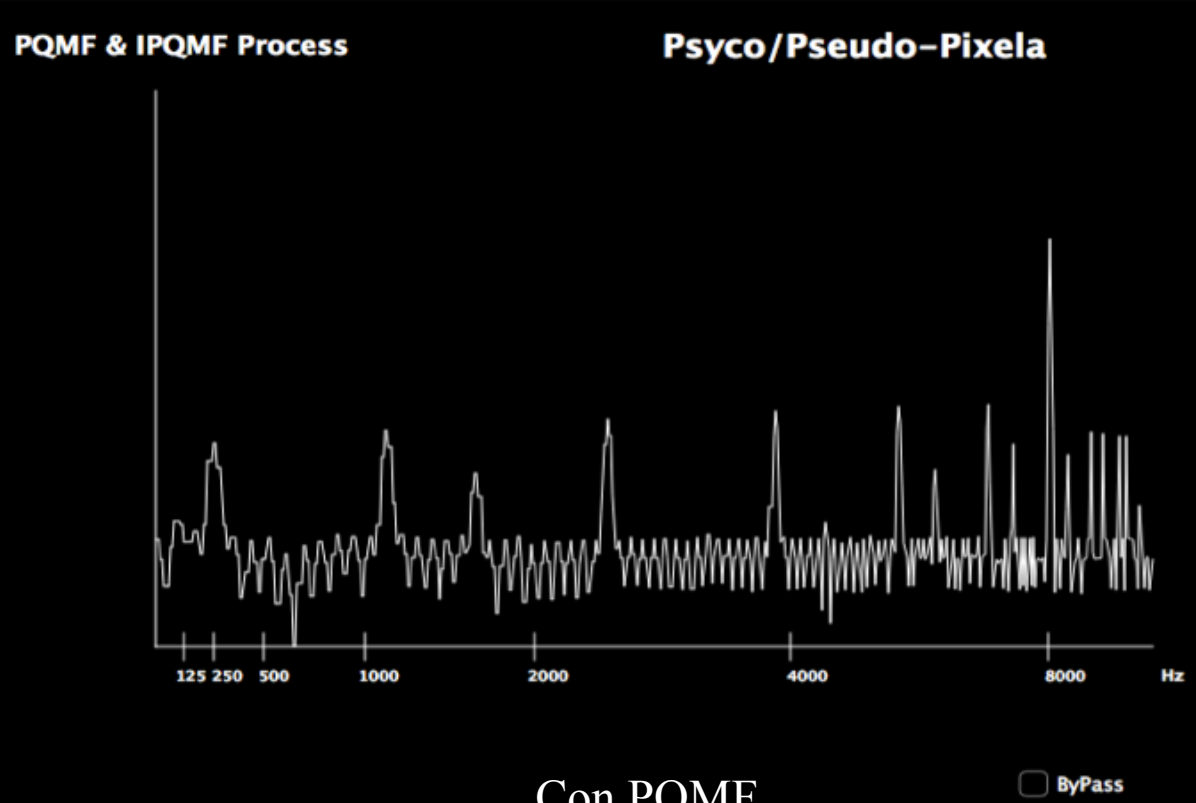


Sin embargo, siempre la última muestra de cada L = buffer.getNumSample no es interpolada, dado que se pierde la referencia, por lo que se genera discontinuidad cada L muestras y por tanto, distorsión armónica cada L/fs segundos.

Visualización de la Magnitud Espectral del Audio procesado con PQMF y sin procesamiento



Sin PQMF



Con PQMF

BIBLIOGRAFÍA

Bosi, M., & Goldbrg, R. E. (2003). *Didital Audio Signal Processing.* Boston: KLUWER ACADEMIC PUBLISHERS.

Meyer, F. G. (2015). *Lab 4.* Colorado: University of Colorado at Boulder.