123

Proyecto Final de Procesamiento Digital de Señales: Visualizador de Altura Musical

Mendoza Gutiérrez José Manuel, Santana Hernández Francisco Javier
González Armendáriz Cassandra, Martini Toriz Christophe

I. Introducción

PARA nuestro proyecto final del curso de ”procesamiento digital de señales” decidimos diseñar un programa que nos ayude a visualizar la altura de las notas en una pista de audio. Tomamos esta decisión principalmente porque pensamos que un programa que ayude a un músico a encontrar las notas musicales en una pista de audio le sería de gran ayuda. La segunda razón fue porque es un muy buen ejemplo de como se puede combinar la ciencia con el arte.

Comenzamos por plantearnos el problema de como podríamos diseñar este visualizador. Investigamos la teoría que se encontraba detrás del proyecto, determinamos que tan factible era de realizar, luego buscamos el entorno de desarrollo adecuado para realizar el programa y finalmente implementamos el código necesario.

Se decidió que el visualizador fuera ”el tetraedro de Sierpinski”, el cual cambiara de color conforme a las notas reproducidas de una pista mp3. Estos colores se basarían en la relación ”color-altura” establecida por el compositor simbolista ”Alexander Scriabin” quien diseñó un teclado ”Clavier à lumières”que proyectaba un color diferente en cuanto se tocaba una nota específica. Para lograr nuestro objetivo utilizamos el entorno de desarrollo ”processing” basado en el lenguaje ”java”. Processing fue diseñado específicamente para diseñadores visuales y posteriormente se crearon una gran cantidad de librerías para poder implementar sonido, expresiones matemáticas y animaciones 3D de alta resolución.

En este reporte primero presentaremos la teoría y el entorno de ”Processing”, posteriormente expondremos nuestro desarrollo, luego mostraremos nuestros resultados y finalmente cerraremos el reporte con una conclusión.

II. Marco teórico

A. Musical

En la música dodecafónica occidental existen 12 notas por cada octava. Estas notas son ”Do, Do♯, Re, ♯, Mi, Fa, Fa♯, Sol, Sol♯, La, La♯, Si”, ♯ se lee ”sostenido”. Sin entrar en detalles se puede decir que las notas ”sostenidas” son iguales a las notas ”bemoles (♭)” superiores. Este es el principio ”enarmónico”. Cada octava comprende estas doce notas, las cuales se siguen para pasar a la octava superior. Entre cada de estas notas se dice que existe un semi-tono. Esto es importante para la teoría musical y las relaciones ármonicas entre las notas.

Cada nota tiene una frecuencia específica y ésta define de que nota se trata y en que octava se encuentra. A causa de esto las notas llevan la notación siguiente (en física): x_i donde ”x” es el nombre de la nota e ”i” es el índice de la octava a la cual pertenece. ”i” puede ir de -1 hasta 10 en el espectro audible, esto es, de 20Hz a 20 000Hz. También se puede observar que $f(x_{i+1}) = 2f(x_i)$ donde ”f” es la frecuencia en hertz. Es decir, la frecuencia de la misma nota pero en la octava superior es igual a dos veces la frecuencia de la nota en la octava inferior. Frecuencias aproximadas en hertz correspondientes a cada nota en función de sus octavas (las primeras 8):

X;i	-1	0	1	2	3	4	5	6	7
Do	16.35	32.70	65.4	130.8	261.6	523.2	1046.4	2092.8	4185.6
Do♯	17.32	34.64	69.28	138.56	277.12	554.24	1108.5	2217	4434
Re	18.36	36.72	73.44	146.88	293.76	587.52	1175	2350	4700
Re♯	19.45	38.9	77.8	155.6	311.2	622.4	1244.8	2489.6	4979.2
Mi	20.6	41.2	82.4	164.8	329.6	659.2	1318.4	2636.8	5273.6
Fa	21.8	43.6	87.2	174.4	348.8	697.6	1395.2	2790.4	5580.8
Fa♯	23.1	46.2	92.4	184.8	369.6	739.2	1478.4	2956.8	5913.6
Sol	24.5	49	98	196	392	784	1568	3136	6272
Sol♯	25.95	51.9	103.8	207.6	415.2	830.4	1660.8	3321.6	6643.2
La	27.5	55	110	220	440	880	1760	3520	7040
La♯	29.1	58.2	116.4	232.8	465.6	931.2	1862.4	3724.8	7449.6
Si	30.9	61.8	123.6	247.2	494.4	998.8	1977.6	3955.2	7910.4

Hay que aclarar que la única nota que tiene una frecuencia

M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).
J. Doe and J. Doe are with Anonymous University.
Manuscript received April 19, 2005; revised August 26, 2015.

exacta es LA con una frecuencia de 27.5 Hz. Las demás fueron redondeadas. Esta es la razón por la cual el LA 440Hz se utiliza para afinar, a este LA se le llama LA 4 pero en este proyecto vamos a utilizar la notación europea por lo que todos los índices de las octavas se recorren una unidad hacia los negativos.

Por otro lado, los únicos instrumentos existentes que producen ondas sinusoidales (o cosenoidales) puras son los diapasones. Todos los demás instrumentos musicales producen una onda que es periodica pero no sinusoidal, por lo que cuando se toca en el instrumento una nota de frecuencia f, denominada frecuencia fundamental o a la altura, aparecen otras frecuencias denominadas armónicos. Los armónicos tienen frecuencias que son múltiplos de dos de la fundamental. Esto es debido a la propiedad de que las ondas periodicas se pueden descomponer en una suma de ondas sinusoidales (o cosenoidales). Este es el principio de la transformada de Fourier que se estudiará en la siguiente sección. Esto es fundamental para nuestro proyecto, ya que nosotros buscamos visualizar la frecuencia fundamental. Esto implica deshacernos de los armónicos que deben de ser tratados como ruido no deseado. Uno de los aspectos que simplifica este trabajo es que usualmente la frecuencia de la fundamental es la que tiene mayor amplitud, por lo que nos es fácil reconocerla.

Al momento de investigar acerca de nuestro proyecto también nos dimos cuenta de que es imposible encontrar las alturas de diversos instrumentos en una pista polifónica. Esto se debe a que los armónicos se mezclan con las fundamentales, por lo que es imposible discernir entre una frecuencia fundamental y una armónica y tambien es imposible saber que frecuencia le pertenece a que instrumento. Por esta razón nuestro proyecto solo funcionará con pistas monofónicas, es decir con una sola nota tocada a la vez.

Fig. 1. Un diapasón.

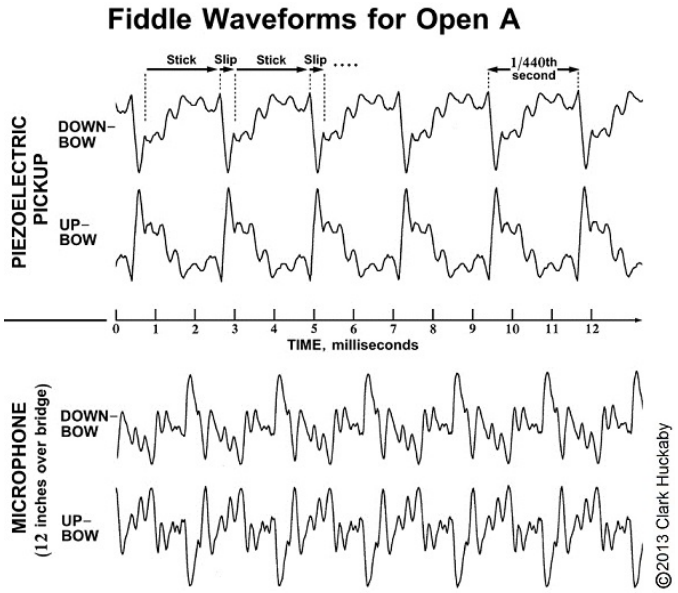

Fig. 2. Ondas sonoras de un violín en diferentes condiciones.

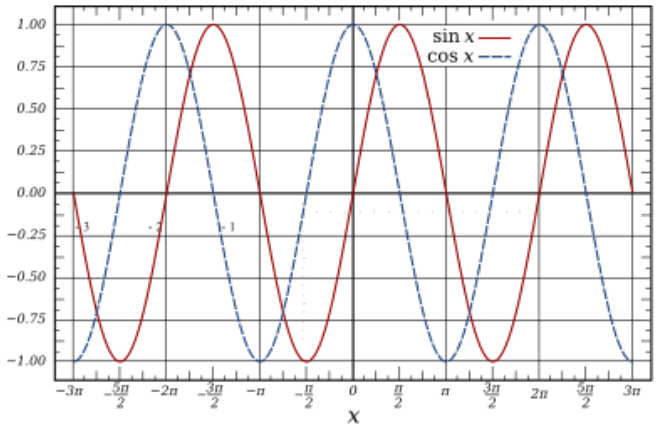

Fig. 3. Onda cosenoidal y onda sinusoidal

B. Procesamiento de señales

La transformada de Fourier deconstruye una representación en el dominio del tiempo de una señal en la representación del dominio de la frecuencia. Es una manera diferente de ver la misma señal.Un digitalizador muestrea una forma de onda y la transforma en valores discretos. Debido a esta transformación, la transformada de Fourier no funcionará con estos datos. En cambio, se usa la transformada de Fourier discreta (DFT), que se produce como el resultado de sus componentes de dominio de frecuencia en valores discretos.

La transformáda rápida de Fourier (FFT) es un algoritmo para el cálculo de la DFT basado en la división del tiempo, eliminando así gran parte de los cálculos repetitivos que hay que llevar a cabo si se desea resolver la DFT de forma directa.

Al aplicar una FFT para medir el componente de frecuencia de una señal, el análisis se basa en un conjunto finito de datos. La transformada FFT real asume que es un conjunto finito de datos, un espectro continuo que es un periodo de una señal periódica. Para la FFT, tanto el dominio del tiempo

como el dominio de frecuencia son topologías circulares, por lo que los dos extremos de la forma de onda de tiempo son interpretados como si estuvieran conectados entre sí. Cuando la señal medida es periódica y un número entero de periodos llena el intervalo de tiempo de adquisición, la FFT resulta bien, ya que coincide con esta suposición.

Sin embargo, muchas veces, la señal medida no es un número entero de periodos. Por lo tanto, la finitud de la señal medida puede dar lugar a una forma de onda truncada con diferentes características a la señal de tiempo continuo original y la finitud puede introducir cambios en la transición brusca en la señal medida. Las transiciones bruscas son discontinuidades.

Cuando el número de períodos en la adquisición no es un entero, los extremos son discontinuos. Estas discontinuidades se muestran en la FFT como componentes de alta frecuencia que no se presentan en la señal original. Estas frecuencias pueden ser mucho más altas que la frecuencia de Nyquist y son escalonadas entre cero y la mitad de su velocidad de muestreo. El espectro que se obtiene al usar una FFT, por lo tanto, no es el espectro real de la señal original, sino una versión distorsionada. Da la apariencia de que la energía en una frecuencia se fuga a otras frecuencias. Este fenómeno se conoce como fuga espectral, lo que hace que las líneas espectrales finas se difundan en señales más amplias.

Para minimizar los efectos de ralizar una FFT sobre un número no entero de ciclos se utiliza una técnica llamada ventaneo. Las ventanas reducen la amplitud de las discontinuidades en los límites de cada secuencia adquirida por el digitalizador. Consiste en miultiplicar la señal de entrada por una ventana de longitud finita que tiende hacia cero en los bordes. Esto hace que los extremos de la forma de onda se encuentren y por lo tanto, da como resultado una forma de onda continua y sin trancisiones bruscas.

Existen diferentes tipos de funciones ventana que se pueden aplicar dependiendo de la señal. Una gráfica real de una ventana muestra que la característica de frecuancia es un espectro continuo con un lóbulo principal y varios lóbulos laterales. El lóbulo principal se centra en cada componente de frecuencia de señal de dominio del tiempo y los lóbulos laterales se aproximan a cero. La altura de los lóbulos laterales indica el efecto que la función de ventanas tiene en las frecuencias alrededor de los lóbulos principales. La respuesta del lóbulo lateral de una señal sinusoidal fuerte puede dominar la respuesta del lóbulo principal de una señal sinusoidal débil cercana. Por lo general, los lóbulos laterales más bajos reducen la fuga en la FFT medida, pero aumentan el ancho de banda del lóbulo mayor. El rango de corte del lóbulo lateral es el rango de caída asintótica de los picos del lóbulo lateral. Al aumentar el rango de caída del lóbulo lateral, se reduce la fuga espectral.

Seleccionar una función ventana no es una tarea simple. Cada función de ventana tiene sus propias características y aptitud para diferentes aplicaciones. Para elegir una función ventana, se debe calcular obligatoriamente el contenido de la frecuencia de la señal.

- Si la señal contiene componentes de frecuencia de fuerte interferencia alejados de la frecuencia de interés, se elige una ventana con un rango alto de caída del lóbulo frontal. - Si la señal contiene fuertes señales de interferencia cerca de la frecuencia de interés, se elige una función ventana con un nivel bajo de lóbulo lateral máximo. - Si la frecuencia de interés contiene dos o más señales muy cerca una de la otra, la resolución espectral es importante. En este caso, es mejor elegir una ventana con un lóbulo principal muy estrecho. - Si la precisión de la amplitud de un solo componente de frecuencia es más importante que la ubicación exacta del componente en un contenedor de frecuencia determinado, se elige una ventana con un lóbulo principal amplio. - Si el espectro de la señal es más bien plano o de banda ancha en el contenido de frecuencia, se utiliza una ventana uniforme.

La preferencia entre una ventana u otra para determinada aplicación no tiene regla general, se recomienda aplicar diferentes tipos de ventanas y experimentar hasta encontrar la mejor para cada aplicación particular.

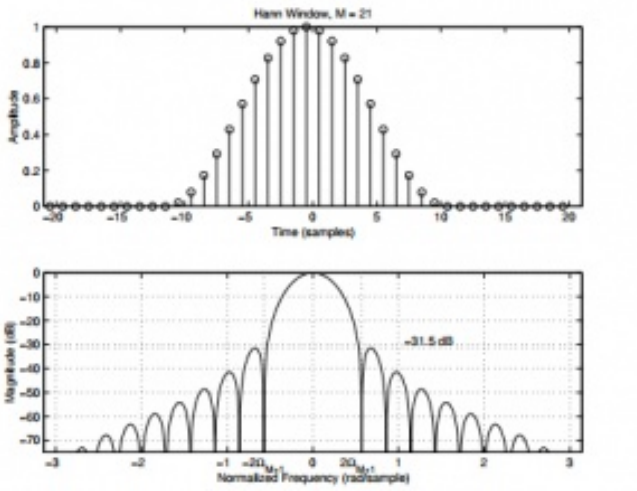

Fig. 4. Ejemplo de una ventana tipo Hann

C. Color

Para la relación color-frecuencia nos propusimos basarnos en la relación establecida por ”Alexander Scriabin” quien fuen un compositor simbolista de nacionalidad rusa, nacido el 6 de enero de 1872 y fallecido el 27 de abril de 1915. Lo especial de este compositor es que se dice que sufría de sinestesia. Es decir, de una ”enfermedad” ,la cual permite que percepciones sensoriales estimulen otro sentido. En su caso, Scriabin, al escuchar notas musicales podía percibir colores. Debido a esto, las obras tardías del compositor fueron basadas en la visualización de colores más que en la musicalidad. Así, Scriabin se propuso crear un intrumento denominado ”clavier a lumières” que significa ”teclado de luces”. Este instrumento tenía por función emitir una luz de un cierto color en cuanto se tocara una nota en específico en el teclado (que era identico al de un piano).

Ya que analizamos los colores correspondientes a cada nota, procedimos a investigar la teoría del color. Cada color tiene un código ”RGB” que significa Red-Green-Blue. La notación del código es la siguiente:

(R_x, G_x, B_x, Br_x), donde R_x, G_x, B_x son las cantidades de rojo, verde o azul respectivamente en el color al cual pertenece

la notación, Br_x no siempre es un parámetro presente pero representa el brillo del color, es decir la luminosidad.

Todos los parámetros pueden ir de 0 a 255, ya que esto es lo máximo que puede ser codificado dentro de un ”BYTE”. Esto implica que casi cualquier color del espectro visible puede ser representado mediante la notación ”RGB”. Las notas corresponden a los siguientes colores con sus respectivas notaciones:

X,C	R	G	B
Do	255	0	0
Do♯	128	0	128
Re	255	255	0
Re♯	255	224	189
Mi	245	245	255
Fa	90	11	10
Fa♯	0	191	255
Sol	255	128	0
Sol♯	200	162	200
La	0	255	0
La♯	255	0	128
Si	0	0	255

Con esto obtemos las relaciones:
Do: Rojo intenso
Do♯: Púrpura
Re: Amarillo
Re♯: Piel
Mi: Escarcha
Fa: Café rojizo
Fa♯: Azúl cielo
Sol: Naranja
Sol♯: Lila
La: Verde
La♯: Rosa
Si: Azúl rey

Fig. 5. Alexander Scriabin

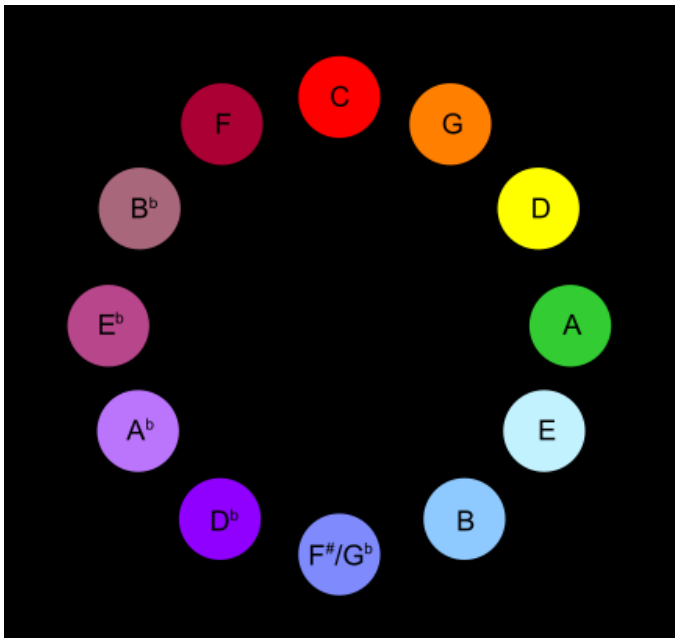

Fig. 6. El círculo de quintas según los colores de Scriabin

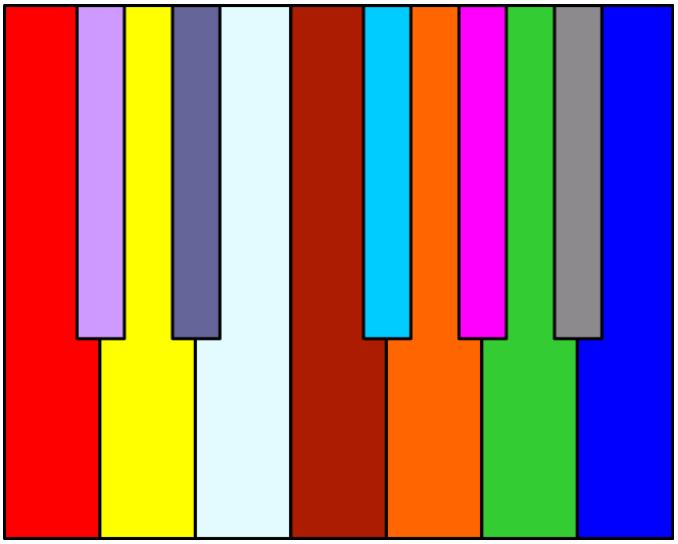

Fig. 7. Los colores correspondientes a cada nota en el ”clavier a lumières”

D. Tetraedro de Sierpinski

El ”tetraedro de Sierpinski” es una construcción matemática de tipo fractal. Un fractal es un objeto geómetrico que se construye con copias de el mismo en una escala mas pequeña, esto puede reproducirse hasta el infinito. Esta característica se le denomina ”autosimilitud”. El ”tetraedro de Sierpinski” es el ”triángulo de Sierpinski” aplicado a tres dimensiones. Este ”triángulo” se construye primero con un triángulo equilatero de catetos de longitud n. A este triángulo se le divide en tres triángulos de longitud n/2 ubicados en los vértices del triángulo original, esto provoca un triángulo central invertido el cual es borrado. Finalmente se realiza este procedimiento m veces con los triángulos internos.

El ”tetraedro de Sierpinsky” se construye de forma similar pero con tetraedros equiláteros en lugar de triángulos.

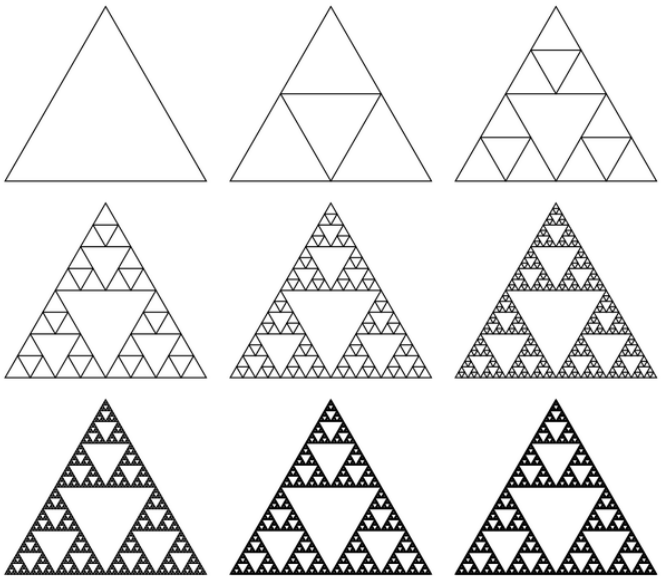

Fig. 8. Construcción del ”triángulo de Sierpinski”

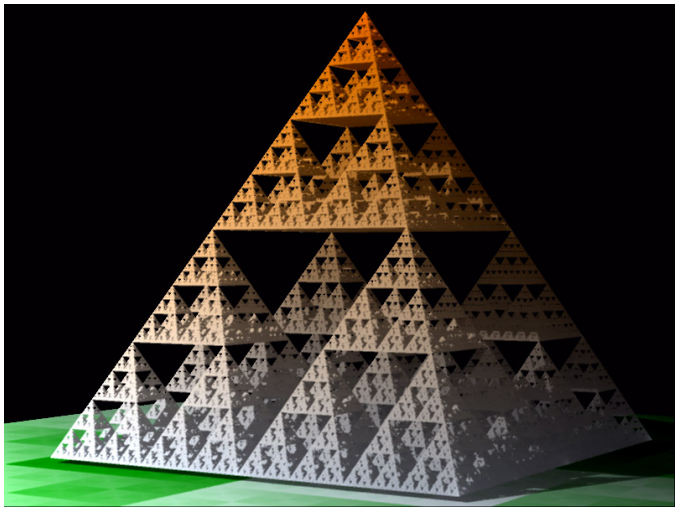

Fig. 9. ”Tetraedro de Sierpinski”

Para lograr definir los fractales, el Matemático Feliz Hausdorff logró determinar una ecuación para calcular la dimensión de estas construcciones geométricas. Estas dimensiones no son enteras si no que son fraccionarias. Esta dimensión clasifica la complejidad del fractal y se determina por:

$$D = \frac{ln(n)}{ln(\frac{1}{s})}$$

Donde D es la dimensión, s es el factor de escala al que se reproduce el fractal y n es el número de piezas que lo componen. La dimensión de ”el tetraedro de Sierpinski” es de dos. Esto implica que es una pirámide sin ”volumen”.

E. Processing

Processing es un entorno de desarrollo integrado ”open source” basado en el lenguaje ”Java”. Processing fue diseñado específicamente para diseñadores digitales por Ben Fry y Casey Reas del MIT. El program esta basado en dos funciones príncipales ”setup” y ”draw”. La primera sirve para realizar calculos previos a la ejecución del programa y diseñar los valores predeterminados para la ventana de visualización. La segunda función sirve para dibujar figuras mediante funciones predeterminadas, o para cambiar parámetros y realizar cálculos mientras se está ejecuntando el código.

Para utilizar processing orientado fue necesario estudiar e implementar una librería llamada ”minim”. Esta librería es capaz de realizar transformadas discretas de fourier, manejar casi todos los objetos asociados con ella, aplicar ventanas y reproducir archivos de música, entre otras cosas.

De la misma forma processing es capáz de utilizar vectores, álgebra, y cálculo básico. Tambien es capáz de realizar diseños en tres dimensiones en tiempo real y contiene todos los colores asociados con el sistema ”RGB”.

```
sketch_151117a
1  void setup(){
2    size(500, 500);
3  }
4
5  void draw(){
6    line(250, 20, 250, 50);//Antena
7
8    rect(220, 50, 60, 60);//Cabeza
9
10   rect(180, 110, 135, 60); //Torso
11
12   quad(315, 110, 345, 110, 355, 140, 315, 140);//HombroIz
13 }
```

Fig. 10. Ejemplo de funciones ”setup” y ”draw”

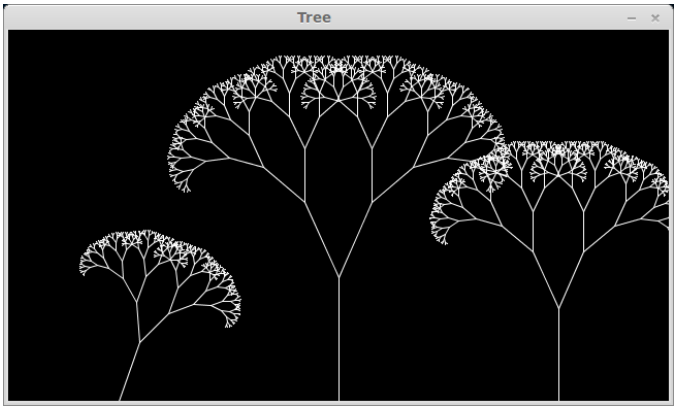

Fig. 11. Fractal creado con processing

III. Desarrollo

Para el desarrollo lo primero que hicimos fue estudiar el entorno de processing. Para esto estudiamos el algorítmo de un estudiante de programación que diseñó un visualizador de música[1]. Pero este visualizador nada más representaba la amplitud de las ondas sonoras y dividía las frecuencias altas, bajas y medias. Este visualizador se enfocaba en crear cubos en movimiento. Gracias a esto logramos entender mejor como se realizaban proyectos orientados al sonido en processing.

Luego de esto buscamos como podríamos diseñar la ”pirámide de Sierpinski”. Encontramos un código[2] que la realizaba gracias a una serie de puntos. Implementamos este código en el nuestro para crear el fractal. Posteriormente estudiamos los elementos de este código para poder entenderlo y que es lo querealizaba paso por paso (esto lo explicaremos más adelanto en cuanto hayamos abarcado la explicación de nuestro código). Finalmente estudiamos todas las funciones de la librería minim que nos ayudarían a realizar nuestro proyecto.

Ahora explicaremos el diseño de nuestro código. Primero empezamos por inicializar las librerías y seguimos por declarar las variables y los objetos. Los objetos que declaramos fueron la ”FFT”, un objeto ”minim”, el reproductor de audio, y una cámara movil de la librería ”peasy”.

[1]este código se puede encontrar en: https://github.com/samuellapointe/ProcessingCubes

[2]este código se puede encontrar en: http://williamchyr.com/tag/sierpinski-tetrahedron/

```
vis
1   //inicializamos las librerías
2  import ddf.minim.*;
3  import ddf.minim.analysis.*;
4  import ddf.minim.effects.*;
5  import peasy.*;
6  import peasy.org.apache.commons.math.*;
7  import peasy.org.apache.commons.math.geometry.*;
8
9  //inicializamos las variables de las librerías
10 FFT fft;
11 Minim minim;
12 AudioPlayer song;
13 BandPass bpf;
14 PeasyCam cam;
15
16 //inicializamos las variables a utilizar
17 float x,y,z;//son los ejes coordenados
18 float[] a;//arreglo que recibirá las amplitudes de la fft
19 float b=0;//variable utilizada para la ordenación de a
20 int fo=0;//el índice de a con mayor fft
21 float f;// frecuencia con mayor fft
22 float fundamental;//fundamental en el piano
```

Fig. 12. Inicialización de las librerías y declaraciones de Variables y objetos

Después en la fución ”setup” creamos una pantalla 3D. Luego cargamos el archivo de audio y le aplicamos la ”fft” con el tamaño del buffer y la frecuencia de muestreo del archivo de sonido. Posteriormente reproducimos el archivo de sonido e inicializamos la cámara movil con unas coordenadas iniciales aleatorias para la pirámide.

```
22 void setup(){
23   size(1000,1000,P3D);
24 minim= new Minim(this);//declaramos la utilización de la librería
25 song=minim.loadFile("prelude.mp3");//cargamos la pista mp3
26 //creamos la FFT con el tamaño del buffer y la frecuencia de muestreo de la canción
27 fft=new ddf.minim.analysis.FFT(song.bufferSize(),song.sampleRate());
28 song.play();//Reproducimos la canción
29 cam = new PeasyCam(this,100);//creamos un objeto de cámara móvil
30   //creamos las coordenadas iniciales del punto de pártida de la cámara móvil
31    x=random(-1000,1000);
32    y=random(-1000,1000);
33    z=random(-1000,1000);
34
35 }
```

Fig. 13. Función ”main”

Lo segundo fue diseñar nuestra función ”draw”, en la cual establecímos un fondo negro y un color inicial blanco para el tetraédro. Después aplicamos la ”fft” a todo tiempo de reproducción de la pista sonora. A esta ”fft” le aplicamos una ventana ”gausseana” para disminuir el ruido. Lo siguiente fue diseñar un código que encontrara el valor máximo de la ”fft” en el instante t. Este código nos regresa el índice de la banda de frecuencia donde se encuentra el máximo de la ”fft”. Luego aplicamos un filtro IIR pasabanda a la frecuencia con la amplitud mas grande y con ancho de banda igual a su frecuencia, ya que los ármonicos se presentan en múltiplos de dos de la frecuencia original. Con una pequeña ecuación logramos recuperar la frecuencia a la cual corresponde este índice. $f = \frac{if_s}{BS}$, donde f es la frecuencia correspondiente a la banda ”i”, fs es la frecuencia de muestreo y BS es el tamaño del buffer de la pista de audio (el tamaño a lo largo del cual se realiza la ”fft”).

Lo penúltimo fue crear el algorítmo de asociación altura-color. Para esto transformamos la frecuencia en el numero de una tecla de un teclado de piano, utilizando como punto de partida la nota ”La” de 27.5Hz, la cual es la nota más

```
//encontrar maximo-processar el arreglo de frecuencias
a= new float[fft.specSize()];
//guardamos la transformada en un arreglo
for(int j=0;j<fft.specSize();j++){
a[j]=fft.getBand(j);

//float a[]=song.mix.toArray(); pudimos usar esta función en lugar del for

}

//encontramos la banda procesada con mayor amplitud
   b=a[0];
   for(int k=1;k<a.length;k++){
 if(a[k]>b){
   b=a[k];
 fo=k;
 }
}
  bp = new BandPass(fo,fo,song.sampleRate());

  bp.process(a);
   //Procesamos el arreglo con un filtro pasaband para reducir el filtrado de armónicos
 //con un ancho de banda igual a la frecuencia ya que los armónicos se presentan en frecuencias multiples de dos de la original
  //recuperamos la frecuencia correspondiente al índice de la banda
 f=(float)fo*song.sampleRate()/song.bufferSize();
```

Fig. 14. Primera parte de la función ”draw”

baja en un teclado de piano(excluyendo al ”Bosendorfer Imperial”). De manera a lograr esto implementamos una ecuación logarítmica:

$n = \frac{ln(\frac{f}{27.5})}{ln(\sqrt[12]{2})}$ Luego redondeamos este resultado a un entero.

n es el número de la tecla de piano (n empieza en 0) y f es la frecuencia en Hz con mayor amplitud en el espectro. Finalmente empleamos un código que asigne cualquier tecla+12n a su correspondiente color. Y asignamos la magnitud de la frecuencia al brillo del color para que se pueda observar el cambio de matíz.

Para terminar implementamos el código para crear el ”tetraédro de Sierpinski” por medio de puntos que se situán cada uno a la mitad de la distancia creada por dos puntos anteriores. Se empieza por puntos que describen los vértices de la pirámide.

```
for(int m=0;m<9;m++){
if(fundamental==(m*12)){//verde la
  stroke(0,255,0,50*fft.getBand(fo));
  }
  if(fundamental==(1+m*12)){//rosa la sostenido
    stroke(255,0,128,50*fft.getBand(fo));
    }
    if(fundamental==(2+m*12)){//azul si
    stroke(0,0,255,50*fft.getBand(fo));
    }
    if(fundamental==(3+m*12)){//rojo do
    stroke(255,0,0,50*fft.getBand(fo));
    }
    if(fundamental==(4+m*12)){//purpura do sostenido
    stroke(128,0,128,50*fft.getBand(fo));
    }
    if(fundamental==(5+m*12)){//amarillo re
    stroke(255,255,0,50*fft.getBand(fo));
    }
    if(fundamental==(6+m*12)){//piel re sostenido
    stroke(255,224,189,50*fft.getBand(fo));
    }
    if(fundamental==(7+m*12)){//frost para mi
    stroke(245,245,255,50*fft.getBand(fo));
    }
    if(fundamental==(8+m*12)){//rojo-cafe para fa
    stroke(90,11,10,50*fft.getBand(fo));
    }
    if(fundamental==(9+m*12)){//azul cielo para fa sostenido
    stroke(0,191,255,50*fft.getBand(fo));
    }
    if(fundamental==(10+m*12)){//naranja para sol
    stroke(255,128,0,50*fft.getBand(fo));
    }
    if(fundamental==(11+m*12)){//lila para sol sostenido
    stroke(200,162,200,50*fft.getBand(fo));
    }
  }
```

Fig. 15. Función de asociación de color

```
109   //tetraedro
110   int n=0,p;
111   point(800,0,-800/sqrt(2));
112 point(-800,0,-800/sqrt(2));
113 point(0,800,800/sqrt(2));
114 point(0,-800,800/sqrt(2));
115 point(x,y);
116 while(n<20000)
117   {
118     p=int(random(1,5));
119 if(p==1)
120     {
121     x=(x+800)/2;
122     y=(y)/2;
123     z=(z-800/sqrt(2))/2;
124     }
125 else if(p==2)
126     {
127     x=(x-800)/2;
128     y=(y)/2;
129     z=(z-800/sqrt(2))/2;
130     }
131 else if(p==3)
132     {
133     x=(x)/2;
134     y=(y+800)/2;
135     z=(z+800/sqrt(2))/2;
136     }
137 else
138     {
139     x=(x)/2;
140     y=(y-800)/2;
141     z=(z+800/sqrt(2))/2;
142     }
143 point(x,y,z);
144     n++;
145   }
146 }
```

Fig. 16. Función para el ”tetraédro de Sierpinski”

IV. Resultados

Nuestros resultados fueron bastante complacientes. Obtuvimos un ”tetraedro de Sierpinski” movible con el mouse que cambiaba de color conforme a la frecuencia de las notas. Realizamos varias pruebas y los colores fueron los adecuados con las notas de prueba. Lo que observamos fue la pirámide colorearse de varios colores (armónicos) pero con el color predominante de la fundamental. Esto fue en su mayoría, ya que en algunos casos la pirámide se coloreó de otro color principal,esto fue debido a que los armónicos tenían una amplitud mayor a la de la fundamental. A consecuencia de esto nos dimos cuenta que la ventana ”gausseana” no fue suficiente para quitar la parte no deseada del espectro. También pudimos observar en cuanto le aplicamos el visualizador al preludio de la suite n1 para chello en sol mayor de Bach, que la pirámide en las cadencias principales resaltaba el sol, o re lo cual implicó que nuestro visualizador si da una cierta noción de la altura tocada.

Por otro lado tambien pudimos destacar que el cambio de color llevaba un cierto retrazo. De la misma manera esta cambio a veces era muy rápido para poder ser observado.

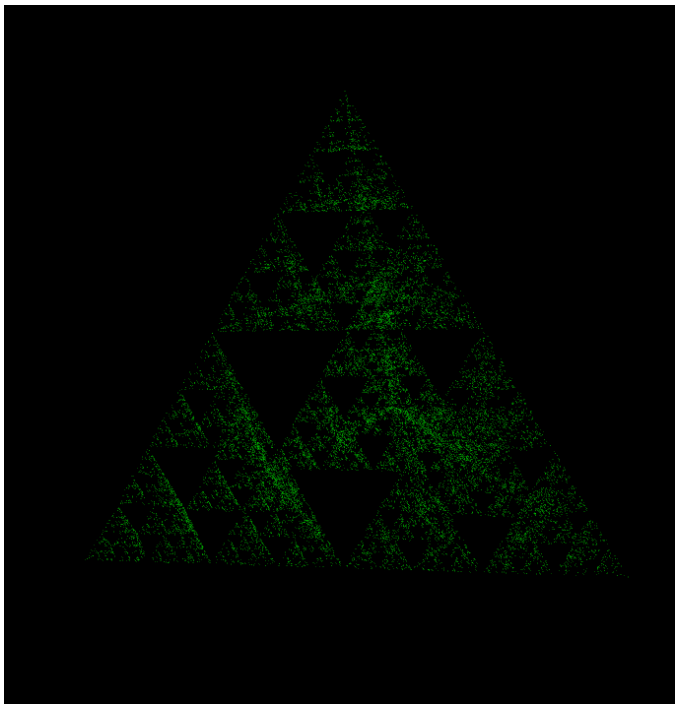

Fig. 17. Visualizador reflejando un La

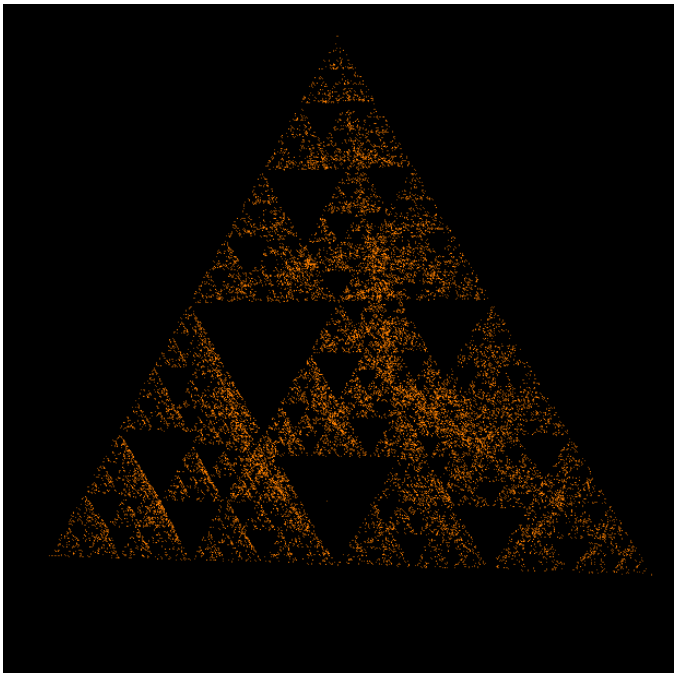

Fig. 18. Visualizador reflejando un Sol

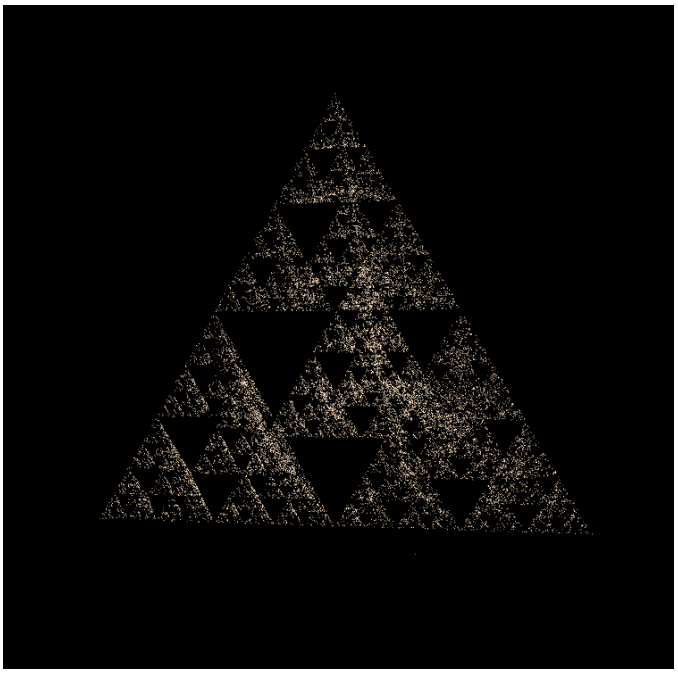

Fig. 19. Visualizador reflejando un Mi

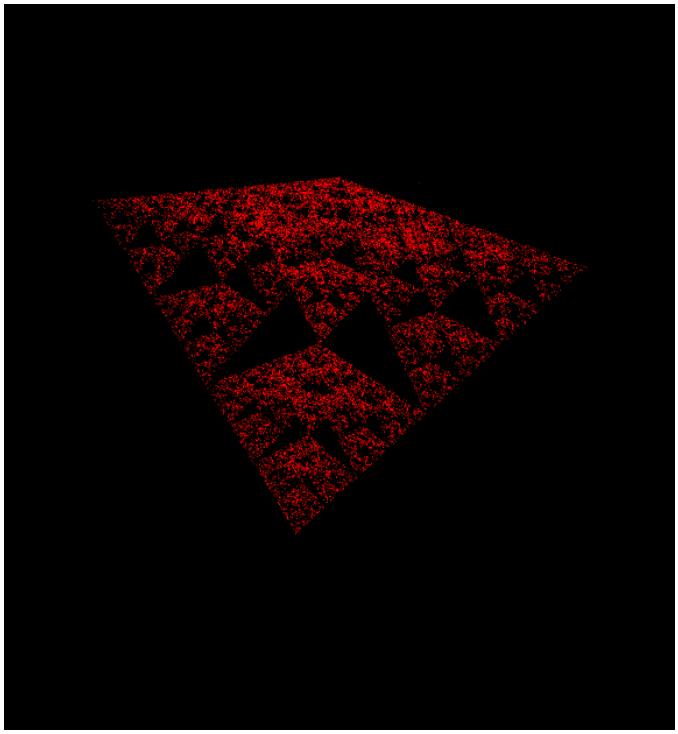

Fig. 20. Visualizador reflejando un Do

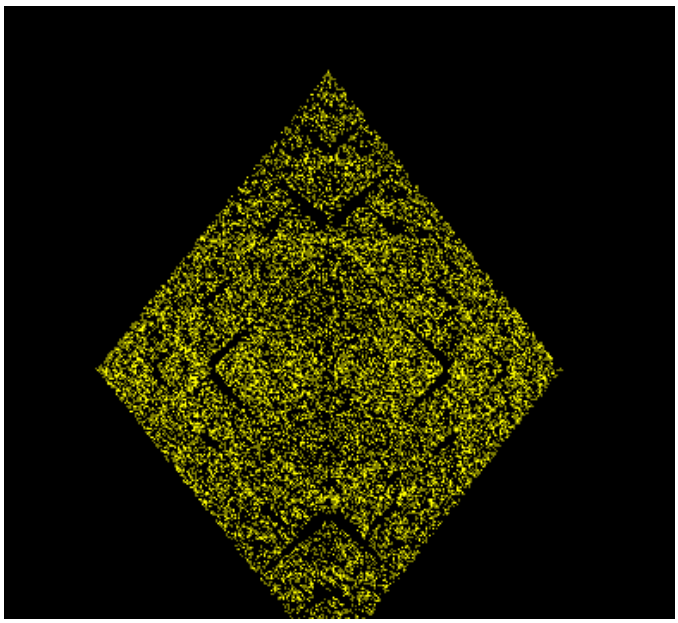

Fig. 21. Visualizador reflejando un Re

V. Conlusión

Finalmente podemos concluir que la visualización de alturas musicales no es una tarea fácil. Existen muchas incertidumbres que pueden modificar el resultado. La única forma de acercarnos a un resultado aceptable es disminuír el paso de frecuencias en el espectro. Pero mas allá de eso es prácticamente imposible diferenciar un ármonico de la fundamental. Esto solamente se puede determinar por la alta probabilidad de que la fundamental tenga una amplitud mayor en frecuencia, pero no siempre es así. Otro problema que se tiene al momento de visualizar la altura musical es que conforme la duración de la nota es más grande, esta arrastra mas armónicos cuando se va disipando la onda sonora.

De la misma forma la magnitud de los armónicos aumenta según como se toca la nota en el instrumento musical. Lo que hace inexacta su determinación. Por ejemplo el tocar mas rápido una nota en el piano libera mas armónicos así como tocar una nota con una arqueada mas larga en el violín. Para disminuír este efecto, nos dimos cuenta que fue posible implementar un filtro pasabanda que nos acercó mas al resultado deseado, logrando reducir el cambio rápido de color de la pirámide, de la misma forma no pudimos observar un retraso causado por el filtro, lo que significa que este no fue considerable. Esto era de esperarse ya que el filtro implementado fue uno de tipo IIR.

A pesar de todas estas incertidumbres logramos acercarnos bastante a un resultado adecuado. Pero como se dijo antes observamos que entre más larga era la nota más la pirámide cambiaba de color para la misma nota. Entonces si se quiere confiar más, en nuestro resultado uno tendría que fijarse en el primer color que se forma. Esto aún sigue siendo del todo confiable ya que aún así podría tratarse de un armónico.

A esto, también debemos sumar otra incertidumbre debido a que para pasar la frecuencia a un entero que represente una tecla del piano es necesario redondear, y es por esto que la nota podría ser la superior o inferior. Se intentó hacerlo simplemente truncando el valor, pero como resultado obteníamos aproximadamente la misma cantidad de colores exactos que redondeando.

Por otro lado tambien pudimos observar que obtener resultados pertinentes en un programa que va detectando las frecuencias conforme avanza una pista sonora es muy dificil a causa del retrazo causado por el procesamiento de la señal y por el código.

References

[1] F.Alton Everest ”the master handbook of acoustics” 3rd edition.
[2] John.R Pierce ”Le son musical” édition: l’univers des sciences.
[3] Wahl.org ”Chapter 4: fractals”
[4] natureofcode.com
[5] processing.org
[6] Philip McLeod, Geoff Wyvill paper on visualization of musical pitch, university of Otago.
[7] ”Analog and Digital Signal Processing” Second Edition. Ashok Ambardar.