UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

*Trabajo Fin de Grado*

# Desarrollo de módulos para la plataforma Soundcool: conexión entre lenguajes visual y sonoro

---


Laura Calabuig Benítez

Tutor: Jorge Sastre Martínez

Cotutor: Stefano Scarani


Valencia, 21 de abril de 2018

## RESUMEN

En este trabajo de fin de grado se ha planteado el diseño de módulos de representación gráfica sobre un análisis de audio para la plataforma Soundcool. Esta es una herramienta de software con módulos interconectables que, a través de móviles, tablets, ordenadores y Kinect, permite sintetizar y transformar cualquier tipo de sonido de forma colaborativa. En su versión 3 se han introducido módulos de vídeo además de audio, en el que el presente trabajo se engloba.

Para llevarlo a cabo se ha estudiado en profundidad el lenguaje Max, prestando especial atención a las herramientas de análisis de sonido y representación 3D y obteniendo como resultado tres diseños diferentes, en los que se han empleado distintas técnicas de análisis y representación. De esta manera, se logra una base sólida para generar infinidad de diseños sin necesidad de conocer a fondo el lenguaje.

Con este proyecto se desarrollan por primera vez módulos Soundcool que fusionan el audio y el vídeo, con una variable artística añadida. Así, los usuarios pueden no solo oír, sino también ver sus creaciones musicales.

## RESUM

En aquest treball de fi de grau s'ha plantejat el disseny de mòduls de representació gràfica sobre un anàlisi d'àudio per a la plataforma Soundcool. Aquesta és una ferramenta de programació amb mòduls interconectables que, a través de mòbils, tablets, ordinadors i Kinect, permet sintetitzar i transformar qualsevol tipus de so de forma col·laborativa. En la seua versió 3 s'han introduït mòduls de vídeo a més d'àudio, englobant el present treball.

Per fer-ho s'ha estudiat en profunditat el llenguatge Max, prestant especial atenció a les ferramentes d'anàlisi de so i representació 3D i , donant com a resultat tres dissenys diferents, en els quals s'han empleat diferents tècniques d'anàlisi i representació. D'aquesta manera, es crea una base sòlida per generar infinitat de dissenys sense necessitat de conèixer a fons el llenguatge.

Amb aquest projecte es desenvolupen per primera vegada mòduls Soundcool que fusionen l'àudio i el vídeo, amb una variable artística afegida. Així, els usuaris poden no només sentir, sinó també veure les seues creacions musicals.

## ABSTRACT

In this thesis it has been proposed the development of the design of graphical representation modules based on an audio analysis for the Soundcool platform. This is a tool with interconnectable modules that, through mobiles, tablets, computers and Kinect, allows to synthesize and transform any type of sound in a collaborative way. In its third version, they are they have been introduced video modules in addition to audio modules, encompassing this project.

To carry it out, the Max language has been studied in depth, paying special attention to the tools of sound analysis and 3D creation and resulting in three different designs, in which different techniques of analysis and representation have been used. In this way, a solid base is managed to generate thousands of designs without the need to know the language in depth.

With this project, Soundcool modules that merge audio and video, with an added artistic variable, are developed for the first time. Thus, users can not only hear, but also see their musical creations.

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS

# 1. INTRODUCCIÓN

Hasta la fecha se han creado diferentes módulos de Soundcool para la modificación del audio y vídeo en tiempo real, pero poco se ha investigado sobre el vínculo entre estos dos parámetros para la creación artística.

A partir de mis conocimientos artísticos sobre el sonido, por poseer el Título Superior de Música, y mi capacidad de aprendizaje y entendimiento de un nuevo lenguaje de programación, además de conocer la parte física del sonido, por cursar el Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación, he visto posible vincular el desarrollo de módulos de software Soundcool con el mundo artístico. Así, los mundos artísticos musical-sonoro y el visual se han unido a través del procesado de audio y vídeo en tiempo real. De esta manera, he podido atender a los parámetros musicales, como el tono o el ritmo, además de los puramente sonoros, como la frecuencia y la amplitud, para producir un vídeo artístico y en tiempo real íntimamente relacionado con la música y el sonido.

El acercamiento entre el mundo sonoro y el visual lo he centrado en elegir un mapeo artístico entre ambos. Esto consiste en tomar valores de los parámetros de uno y aplicárselos a los del otro, con la ayuda del especialista en instalaciones audiovisuales interactivas y video artista Dr. Stefano Scarani. Así, por ejemplo, la escala de volumen de un sonido podría verse como la diferente tonalidad de color que se dé en el vídeo, y viceversa.

En este trabajo he decidido traducir los parámetros sonoros a visuales, y no al revés, porque el proyecto Soundcool así lo exigía. El uso de móviles y tablets para crear la música con Soundcool crea unos espectáculos sonoros impactantes, pero visualmente no muy interesantes: los estudiantes controlan sus tablets y móviles con precisión para generar las obras, pero ello no genera los movimientos y expresividad visual presentes en un concierto con instrumentos tradicionales. Si no se introduce danza, imagen u otros elementos en los conciertos de Soundcool estos no suelen ser visualmente atractivos. Este proyecto viene a apoyar a Soundcool generando espectáculos visuales que sí sean expresivos directamente relacionados con la música que se crea en tiempo real.

Para elegir los diseños a desarrollar se han de abordar dos preguntas fundamentales: ¿Qué partes del audio se van a visualizar? Y ¿Cómo se va a representar visualmente cada una de esas partes? Durante el desarrollo de este trabajo se va a mostrar la respuesta a cada una de estas preguntas para cada proyecto. Por otro lado, he elegido generar gráficos 3D desde la nada, en lugar de modificar una grabación de vídeo, de manera que un alto porcentaje de los elementos

gráficos están estrechamente vinculados con el audio recibido.

Los proyectos *Drops*, *Attraction*, y *Mirrors* se han diseñado con la idea de abarcar, conjuntamente, el número de objetos y algoritmos más variado posible. En las siguientes páginas, pues, se va a explicar en qué consiste cada uno de ellos y qué elementos del lenguaje Max llevan integrados para recrear sus funcionalidades.

# 2. OBJETIVOS

El objetivo del presente trabajo consiste en el diseño de tres proyectos en el lenguaje de programación Max [Cycling74] que representen el material sonoro en forma visual artística, a partir de sus parámetros básicos como frecuencia, intensidad, etc. Este trabajo se encuentra inscrito en un proyecto de mayor envergadura llevado a cabo por el Grupo de Artes Performativas y Tecnología con sede en la Universitat Politècnica de València, Soundcool (soundcool.org), por lo que los objetivos de este trabajo están ligados a los propios de dicho proyecto, que se pueden encontrar en [1].

Soundcool es un sistema gratuito para la creación sonora y visual colaborativa por medio de móviles, tabletas, Kinect y Max / MSP / Jitter. Es desarrollado por el Technology research group (PerformingARTech) formado por personal y estudiantes en la UPV, en colaboración con el grupo de Roger Dannenberg, director del Computer Music Group en la Carnegie Mellon University (CMU). Este proyecto comienza en 2013 con una estancia de su director, Jorge Sastre, en el grupo de Computer Music de la CMU. Soundcool tiene actualmente el apoyo para realizar un proyecto europeo en el que se prevé ampliar las capacidades sonoras y de vídeo creación del sistema actual, tanto para el público en general como para personas con necesidades especiales, así como difundir los resultados obtenidos en revistas y congresos de prestigio internacional.

Soundcool es un proyecto en continua evolución; por este motivo requiere la creación constante de nuevos módulos que contribuyan a ampliar las funcionalidades del sistema. El equipo de la UPV que permanentemente se encuentra trabajando en ello, así como algunos estudiantes de nuestra escuela que han contribuido con sus TFGs, han investigado y colaborado para complementar dicho software. Así pues, la finalidad del presente Trabajo de Fin de Grado es contribuir al desarrollo de esta investigación, concretamente implementando módulos de análisis de audio y creación artística de vídeo en relación con el sonido y la música conjuntamente.

Por ello, mi objetivo se ha centrado en realizar el diseño de tres proyectos desarrollados con el mismo lenguaje de programación utilizado por Soundcool, para que posteriormente puedan ser integrados fácilmente en su plataforma y formar nuevos módulos. De manera más específica, se espera cumplir con los siguientes objetivos:

- Explotar a fondo el lenguaje de programación Max para ofrecer nuevas funcionalidades a Soundcool.

- Aunar en un mismo proyecto análisis de audio y generación de vídeo, un sistema totalmente nuevo e innovador en el sistema Soundcool.
- Crear animaciones estéticamente interesantes para acompañar al sonido en conciertos Soundcool, apoyando al mismo en la realización de espectáculos con procesado en tiempo real, no sólo interesantes desde el punto de vista musical/sonoro sino también artístico visual.
- Ayudar a los usuarios Soundcool a entender sus creaciones musicales al permitirles representar con animaciones lo que están creando, es decir, mostrar de forma artística frecuencias, variaciones de amplitud, etc.
- Evocar sensaciones y experiencias diferentes a los usuarios y espectadores de conciertos Soundcool.
- Aportar las bases para que el equipo de Soundcool pueda crear infinidad de diseños adecuados a sus necesidades en cada momento.

# 3. PREÁMBULOS

## 3.1. Metodología

La metodología seguida para el desarrollo de este proyecto consta de dos vertientes: por un lado, se mostrará la estructura de su planteamiento, vinculado directamente con la estructura de esta memoria. Por otro, se explicará en qué ha consistido cada etapa y se mostrará la distribución de tareas realizadas para la consecución del trabajo mediante un diagrama temporal.

### 3.1.1. Estructura del proyecto

Para la óptima comprensión de los elementos involucrados en el presente trabajo se ha decidido dividir la memoria en tres apartados vinculados entre sí.

En el primero se presentan cada uno de los tres proyectos. Se describe el funcionamiento de *Drops*, *Attraction*, y *Mirrors* tal y como lo puede apreciar el usuario, sin entrar en detalles técnicos.

El segundo apartado se subdivide en varias secciones y subsecciones, en las que se presentan los elementos que se pueden usar para, por un lado, analizar el audio, y por otro, representar el vídeo. Para ello se emplea lenguaje específico de Max y se explican tanto objetos de este lenguaje incluidos en el código desarrollado, como pequeños algoritmos creados con Max para recrear algunas de las funcionalidades que se requieren, como por ejemplo los filtros.

El tercer apartado aúna los dos anteriores, explicando cómo se usan los elementos presentados en el segundo apartado para realizar los tres diseños presentados en el primero. Es decir, se define la relación de parámetros auditivos con parámetros visuales y se muestra cómo se aplican para crear el efecto percibido al ejecutar el programa.

### 3.1.2. Distribución en tareas

Las etapas para la realización del proyecto han sido las siguientes:

1. Estudio de Soundcool: Con el fin de crear un módulo que sirva a esta herramienta, hemos de conocerla en profundidad. Para ello esta fase consistió en descargar el programa, realizar los tutoriales, investigar sobre su uso en diferentes campos y creaciones realizadas con él y jugar con los diferentes módulos para experimentar como usuario. También en esta etapa entré en

contacto con los principales desarrolladores actuales de Soundcool para obtener información específica a la hora de crear módulos homogéneos y compatibles con el sistema.

2. Estudio del lenguaje de programación por módulos Max [Cycling74]: La realización de este trabajo ha supuesto adquirir y aprender un nuevo lenguaje de programación. Esto ha sido posible mediante la asistencia virtual a un curso enfocado en Max 7 ofrecido por la universidad de Stanford a través de la plataforma Kadenze. También ha servido a este propósito la visualización y lectura de numerosos tutoriales y documentación relativos a dicho lenguaje.

3. Estudio de jitter: Este parche de Max para la generación y procesado de vídeo en tiempo real, ha sido estudiado en mayor profundidad dadas la complejidad de su uso e importancia en el presente proyecto. Una parte fundamental de jitter, physics, también ha sido estudiada de manera independiente, principalmente a través de tutoriales oficiales de Max.

4. Programación: Con unas bases de conocimiento sobre el lenguaje a utilizar, se ha procedido a construir el primer módulo. El desarrollo se ha dividido en tres fases: análisis del sonido, conversión artística de parámetros sonoros a visuales y representación de la imagen.

5. Ampliación y optimización: Tras realizar un primer proyecto (*Drops*), se decide ofrecer más posibilidades creando un segundo (*Attraction*) y finalmente un tercer (*Mirror*) proyecto, en los que se usan diferentes elementos tanto para el análisis del sonido, como para la representación visual con physics.

6. Documentación del código: Se comenta y agrupan adecuadamente las diferentes secciones de los proyectos realizados, con el objetivo de que las personas que quieran entenderlo, modificarlo o copiar partes, puedan hacerlo sin complicación.

7. Redacción: Por último, esta etapa recoge la recopilación de la información utilizada a lo largo del desarrollo del trabajo, la redacción de la memoria y la realización de la presentación para la defensa del proyecto.

Ha de mencionarse también el estudio de las Transformadas de Fourier (en este caso FFTs) tanto su revisión teórica física como su uso en el lenguaje Max. Se deben tener en cuenta dada la importancia que supone en el análisis del sonido, elemento básico para el funcionamiento de los diferentes proyectos desarrollados. Finalmente se ha decidido no hacer uso directo del análisis por medio de FFTs, utilizando en su lugar el objeto externo de Max *Analyzer~* realizado por Miller Pukette. *Analyzer~* se basa en el mismo proceso de análisis, las transformadas rápidas de Fourier, generando los datos necesarios para las aplicaciones realizadas. Analiza el audio recibido y da a su

salida parámetros útiles como el nivel de ruido, el brillo, la sonoridad o su composición frecuencial en diferentes formatos (escala Bark, parciales de frecuencia con su amplitud, *pitch* detectado...). Por su facilidad de uso y ser una herramienta creada por desarrolladores experimentados destinada precisamente al análisis de audio, se ha decidido usar directamente *Analyzer~* en lugar de las FFTs.

### 3.1.3. Diagrama temporal

A continuación, podemos ver la temporalización de las tareas llevadas a cabo mediante el siguiente diagrama:

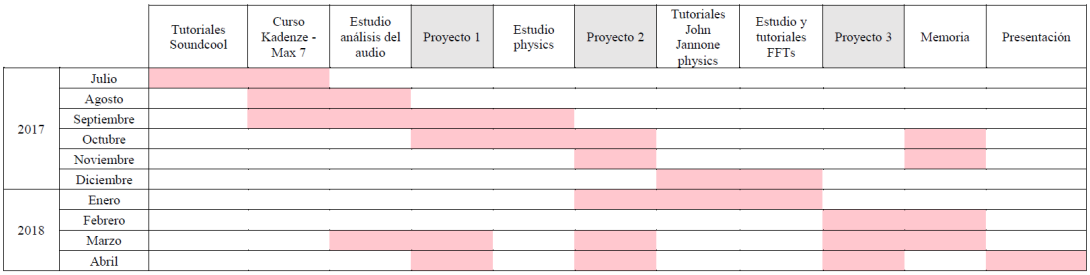



Fig. 3.1. Diagrama temporal

Como se puede apreciar, se han desarrollado las tareas en el orden descrito en el apartado 3.1.2., sin embargo en los meses de marzo y abril se ha vuelto a repasar cada uno de los proyectos para crear su versión final. En esta, se ha ordenando y optimizando el código e incluído comentarios para su mejor comprensión. La memoria se ha ido desarrollando durante todo el proceso de creación de proyectos, ya que debía explicar aquello que se codificaba. Al igual que con los diferentes diseños, en los últimos meses se ha terminado de escribir y corregir el texto del presente documento.

## 3.2. Herramientas

Un sistema de música interactivo es una configuración de hardware y/o software que nos permite cumplir una tarea relacionada con la música, normalmente en tiempo real, a través de cierta interacción. Estos sistemas típicamente tienen una serie de controles, en hardware o software, como interruptores, llaves, botones y sensores, por los cuales los elementos musicales como la armonía, ritmo, dinámicas y timbre pueden ser manipulados en tiempo real a través de la interacción con el usuario [2]. Para crear este tipo de sistemas se debe usar un lenguaje de programación que permita el tratamiento de audio y crear una interfaz sencilla a través de la cual el usuario final

pueda jugar con los parámetros musicales.

Soundcool es un sistema de música interactivo, que recientemente ha estado incluyendo tratamiento de vídeo, además de audio. En este apartado se va a explicar en qué consiste esta plataforma y por qué Max resulta ideal como lenguaje de programación para la misma.

### 3.2.1. Soundcool

Soundcool es un sistema innovador gratuito de educación musical y creación colaborativa mediante móviles, tabletas, Kinect, Open Sound Control (OSC) y MAX/MSP/JITTER creado en la Universitat Politècnica de València (UPV) y financiado por La Fundación Daniel y Nina Carasso, La Cátedra Telefónica de la UPV, la Generalitat Valenciana (España), el Ministerio de Educación, Cultura y Deporte y la UPV. Ha sido realizado con la colaboración del grupo de Computer Music de Roger Dannenberg, creador del software de audio Audacity y director de este Computer Music Group en la Carnegie Mellon University (Pittsburgh, USA), uno de los primeros Departamentos de Computer Science en los rankings internacionales [3].

Soundcool ha sido adoptado en centros de educación primaria, secundaria y escuelas de música en diversos países de Europa a través de proyectos europeos Erasmus+. También se ha introducido en la CMU y en el Instituto Tecnológico de Estudios Superiores de Monterrey (ITESM, México), y han expresado su interés en el sistema la Nayang University of Tecnology de Singapur, la University College London, la New York University, etc., todas ellas universidades de gran prestigio internacional.

Soundcool ha recibido el premio NEM Art de la asociación New European Media de industrias creativas, el Premio del SIMO Educación a la Mejor Experiencia en Programación y Robótica, el Premio Bankia al Talento Musical al mejor grupo innovador educativo ExperimentArts que inició las pruebas con Soundcool desde sus comienzos, y varios premios también al grupo de Soundcool dedicado a su uso para diversidad funcional.

El director del proyecto Soundcool, Jorge Sastre, ha obtenido el Premio Bankia al Talento Musical como Mejor Investigador por su trabajo con Soundcool. Junto a Roger Dannenberg, Sastre ha compuesto una ópera que utiliza el sistema Soundcool para generar todos los efectos sonoros, estrenada en el Palau de les Arts de Valencia en 2016, con fecha prevista de estreno en versión en inglés en el ITESM en 2018. Actualmente (abril 2018) se están realizando pruebas de implantación en 29 centros de primaria y secundaria de acuerdo con la Conselleria de Educación, Investigación, Cultura y Deporte.

Actualmente se utiliza principalmente en diferentes instituciones educativas, con objeto de promover la creación de sonido y música por los propios estudiantes. La arquitectura pedagógica de Soundcool está basada en tres escenarios de educación musical que permiten la interacción entre los diferentes agentes involucrados en la clase. Las diferentes obras musicales creadas en Soundcool por los estudiantes son controladas por ellos mismos a través de dispositivos móviles y el protocolo Open Sound Control (OSC) [3].

Para poder crear y educar en música colaborativamente, este sistema está diseñado a partir de una serie de módulos que se pueden conectar entre sí y funcionan en un ordenador Mac o PC. Los diferentes módulos se pueden controlar directamente con el ratón del ordenador, o a través de WIFI con móviles y tablets android e iOS, o mediante los movimientos del cuerpo con la interfaz de videojuegos Kinect de Xbox. Estos módulos permiten crear, intercambiar, transformar o transponer sonidos, introduciendo efectos y diferentes señales, entre otros [4]. En la fase en la que el proyecto se encuentra actualmente, su versión 3, se pretende trabajar especialmente en el tratamiento de vídeo, añadiendo nuevas funcionalidades a esta herramienta musical interactiva.

### 3.2.2. Max

Max es un lenguaje de programación y entorno de desarrollo gráfico de tratamiento de audio y vídeo en tiempo real que viene siendo utilizado en el ámbito musical y multimedia por ingenieros y artistas interactivos durante más de 15 años. Desarrollado por Cycling74 se define como un kit de herramientas para expresiones audiovisuales/multimedia que no requiere demasiado conocimiento previo sobre programación. De hecho, Max es un lenguaje de programación gráfico, donde las conexiones entre diferentes elementos se hacen introduciendo cajas de objetos en el proyecto, y vinculándolas a través de cables [5].

A lo largo del presente trabajo vamos a referirnos con la palabra *patch* a proyectos creados en Max, que constan de diferentes objetos interconectados entre sí. Estos objetos cuentan con un número de entradas, situadas en la parte superior de la caja del objeto, y un número de salidas, en su parte inferior. Un *patch* puede contener a su vez *subpatches*, donde se desarrollan partes del programa y que resultan muy útiles a la hora de entender cómo se ha creado, o de replicar cierto algoritmo en diferentes situaciones, entre otros ejemplos.

Max tiene una versión gratuita, sin licencia, Max Player, que permite usar los módulos pero no modificarlos, y una versión completa con licencia donde se pueden modificar y crear diferentes proyectos. Esto lo hace también ideal como plataforma para soportar Soundcool, ya que su objetivo es que los usuarios puedan hacer uso de esta herramienta de forma gratuita, sin modificar su

estructura [29].

MSP y Jitter son dos parches adheridos a Max para tratar el audio y el vídeo respectivamente. El hecho de utilizar señales de alta frecuencia o de alta precisión temporal, especialmente el audio, nos lleva a usar MSP, mientras que Jitter fue creado para trabajar con información multidimensional – matrices – y especialmente con material de vídeo. Para identificar objetos tipo MSP, el nombre de este irá seguido del símbolo ~, como, por ejemplo, *delay~*. En el caso de Jitter, el nombre irá precedido de *jit.*, por ejemplo *jit.gl.render*.

Además de bregar con datos de vídeo, Jitter también ofrece una interfaz para OpenGL y Physics. La primera es una herramienta para generar gráficos de forma dinámica, que en lugar de procesar vídeo directamente, representa imágenes a partir de un escenario que contiene objetos, texturas e iluminación. La segunda se puede usar para generar gráficos complejos y que reaccionan en tiempo real [14]. Mediante objetos *jit.phys*, además es posible hacer que los elementos creados con OpenGL interactúen simulando leyes físicas.

Así pues, el lenguaje de programación Max en este trabajo resulta ideal, debido a su aspecto y estructura intuitiva, que representa un diagrama de flujo de datos y facilitará enormemente cumplir con los objetivos que se plantean por su orientación al tratamiento de sonido y vídeo.

# 4. PROYECTOS

En esta sección se describen los tres proyectos diseñados: *Drops*, *Attraction* y *Mirrors* sin entrar en detalles técnicos. Estos diseños se dan en un entorno de 3 dimensiones, cuyos ejes *x*, *y* y *z* quedan definidos en la Fig. 4.1.

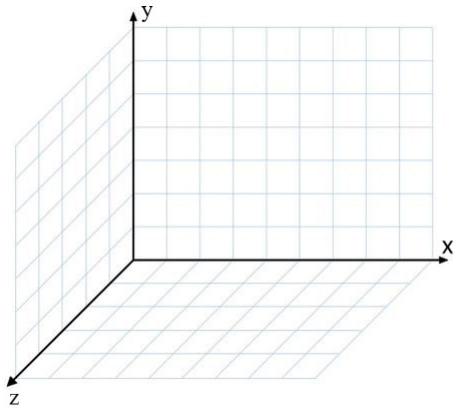



Fig. 4.1. Ejes *x, y, z*

## 4.1. Diseño 1: *Drops*

En este módulo se genera una visualización en la que partículas de agua van apareciendo en la pantalla en una posición determinada por el sonido que se esté reproduciendo (Fig. 4.2.). Grupos de estas partículas se encuentran distribuidos de igual manera a lo largo del eje *x* del espacio del entorno gráfico, tomado como eje de frecuencias, por lo que las partículas más a la izquierda representan las frecuencias más graves, y las partículas más a la derecha a las más agudas. Del mismo modo se ha hecho una distribución gradual del color, desde azul celeste para las primeras frecuencias, hasta magenta para las más agudas. El entorno es iluminado por un foco de luz central, que será más tenue o más fuerte según la intensidad del sonido.

Mientras el módulo reciba audio, las partículas aparecerán en una posición determinada cada vez que se produzca un pequeño aumento en el volumen del sonido, que denominamos ataque. La posición *y* de las gotas en el *mundo*, entorno 3D donde se desarrolla la acción, está determinada por la amplitud media de cada rango de frecuencias representado. Con posterioridad caerán por efecto de la fuerza de gravedad, fijada por el ritmo del sonido, por lo que audios de ritmo agitado provocarán una caída rápida de las gotas, mientras uno lento la ralentizará.

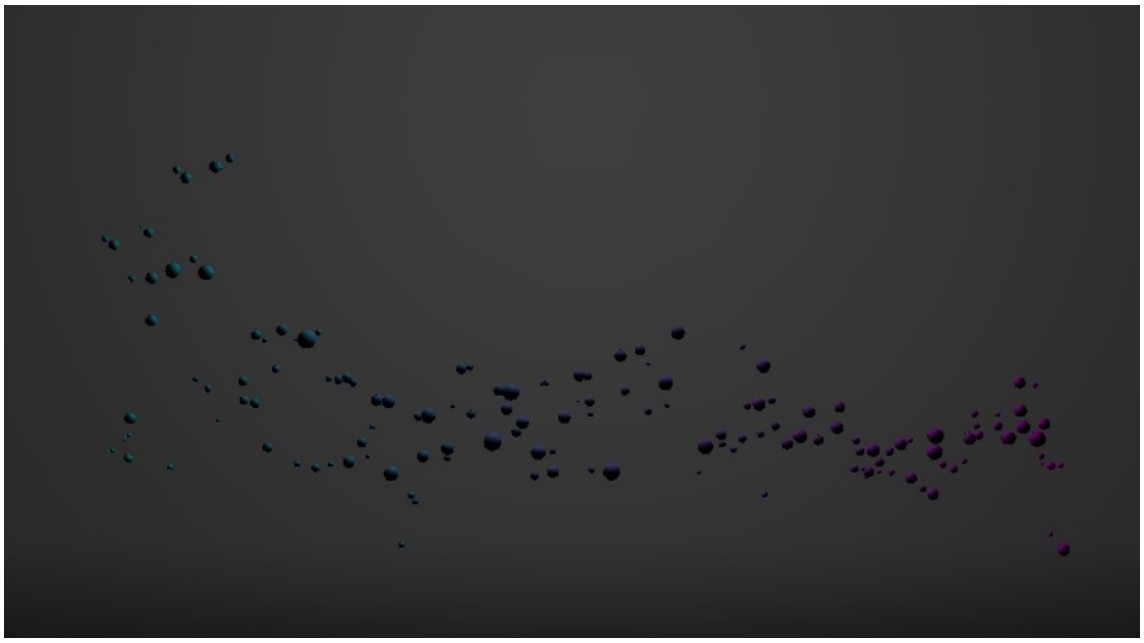


Fig. 4.2. Representación de *Drops*

En este diseño, los elementos configurables manualmente son: valor de la gravedad, color de la luz del ambiente y color y forma de las gotas.

### 4.2. Diseño 2: *Attraction*

En el proyecto *Attraction* un grupo de 150 objetos son atraídos por una esfera central (Fig. 4.4.). Cuando se produce un ataque en el audio, los objetos reciben una fuerza instantánea de repulsión desde este mismo centro, o cada dos segundos si esto no sucede. Su forma cambiará en el momento en que se rompa un silencio de más de 2 segundos, o cada minuto si esto no ocurre. Esta será seleccionada aleatoriamente de entre las siguientes figuras: esfera, cilindro, cubo, cono, círculo, cuadrado o cápsula.

El tamaño de la esfera central será directamente proporcional a la intensidad del sonido recibido. De este modo, el objeto no será visible en momentos de silencio, y tendrá grandes dimensiones en los instantes en que la señal de audio tome su máxima amplitud.

Los objetos de este gran grupo son de tres colores y tamaños diferentes: rojos y pequeños aquellos que representan las frecuencias altas, verdes y de mediano tamaño los elementos que representan las frecuencias medias y azules y algo más grandes aquellos que definen el rango de frecuencias más bajo. La intensidad de sus colores dependerá de la amplitud del sonido en el respectivo rango de frecuencias (bajas, medias o altas), que en total abarca los límites de la audición humana.

Las paredes de la estancia donde sucede la acción (izquierda, techo, derecha y suelo) muestran una representación visual de la señal de audio que se está recibiendo, desde el momento en que este se activa hasta que cesa. Puede elegirse si este dibujo se sobrepone al anterior una vez se hayan recorrido las cuatro paredes, o si, ocurrido esto, el dibujo se borra para volver a empezar. El perímetro de un cuadrado hace de eje temporal. Su base y altura son los ejes *x* e *y* del espacio, situado en $z = 0$, y limita con el techo, suelo y paredes laterales de este espacio (Fig. 4.3.).

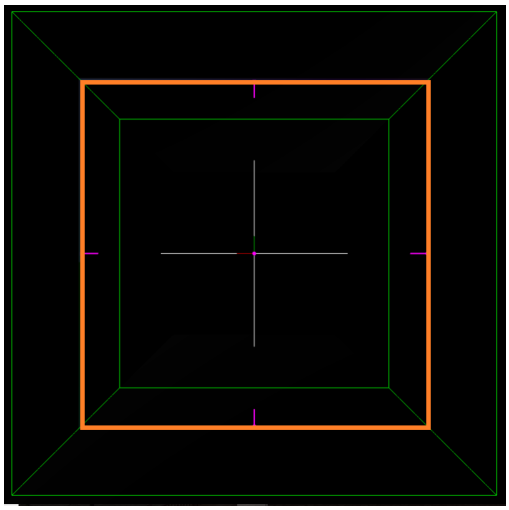


Fig. 4.3. Eje temporal: perímetro del cuadrado

La amplitud del audio aparece como un conjunto de picos en sentido de atrás a adelante para el espectador, recorriendo este eje temportal (representación temporal del sonido). Además, el color de las líneas de amplitud en el tiempo cambiará aleatoria y gradualmente.

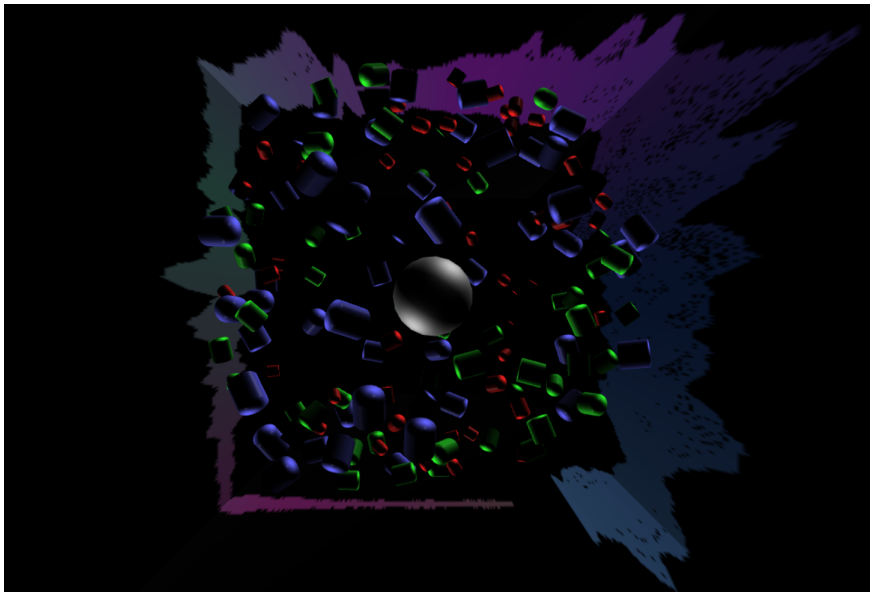


Fig. 4.4. Representación de *Attraction*

Aquí el usuario puede modificar la forma de los objetos manualmente, aplicarles impulsos, modificar los colores de la representación temporal de la amplitud, si se decide que estos no varíen aleatoria y gradualmente, y si estos dibujos van a desaparecer de las paredes o no cada vez que hayan completado una vuelta.

### 4.3. Diseño 3: *Mirrors*

El nombre de este diseño hace referencia a la reflexión de la imagen sobre las paredes del entorno, simulando una habitación de espejos (Fig. 4.5.).

En este caso se parte de dos grupos de objetos que se mueven aleatoriamente en el plano horizontal *XZ* definido en la Fig. 4.1.. El primero, formado por objetos de mayor dimensión, tomará su posición en el eje *y* atendiendo a la frecuencia del primer parcial detectado en el audio de entrada. Así, si esta frecuencia es alta los objetos se moverán en un plano más cercano al techo de la estancia, e irán bajando hasta llegar al suelo si esta disminuye. Además, su forma y dimensiones cambiarán de manera similar a como lo hacían los objetos del proyecto *Attraction*. Por un lado la forma varía en momentos definidos por la duración de los silencios (Sección 4.2.), y tomará la forma de una de las siguientes figuras, escogida de manera aleatoria: Tetraedro, cubo, octaedro, dodecaedro o icosaedro. Su tamaño cambia como lo hacía la esfera central de *Attraction*.

El segundo grupo, formado por octaedros de menor tamaño, define su altura en el espacio tal como lo hace el primer grupo, escalando la frecuencia del segundo parcial detectado. En este caso los objetos no cambian de forma, pero sí modifican su tamaño según la amplitud.

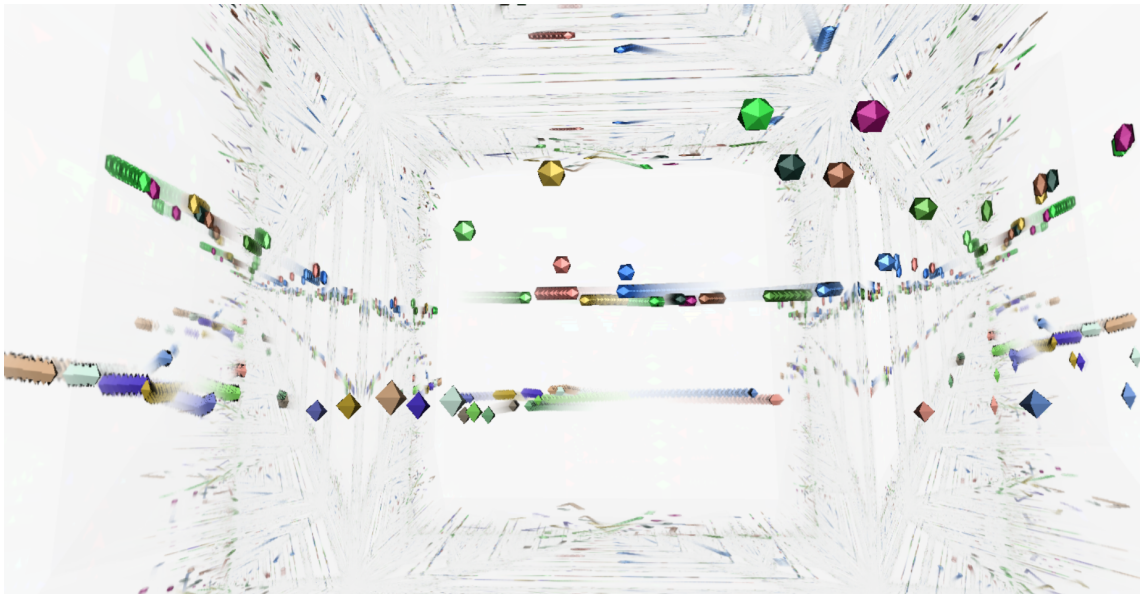


Fig. 4.5. Representación de *Mirrors*

En este diseño se puede seleccionar manualmente el color del entorno, la forma de los diferentes objetos que componen la pantalla y se puede crear una nueva combinación aleatoria de los colores asignados a cada elemento.

# 5. DESARROLLO

## 5.1. Sonido

Los proyectos desarrollados pretenden ofrecer la posibilidad de ser usados tanto a partir de sonido en directo, recogido a través de un micrófono o de instrumentos MIDI, como de pistas pregrabadas. Al crear el módulo Soundcool de los diferentes diseños la entrada de audio se hará a través de entradas *in* compatibles con las *out* estándar en esta herramienta, que permiten conectar unos módulos con otros. Sin embargo, como los diseños presentados no se han integrado en módulos Soundcool aún, sino que son prototipos, se ha decidido dar la opción al usuario de introducir manualmente si se quiere usar un micrófono como entrada de audio, o pistas pregrabadas, con la opción de seleccionarla.

Una vez se ha elegido el medio a través del cual se obtendrá la señal de audio, esta se analiza para obtener diferentes parámetros. Entre estos, se han elegido frecuencia, amplitud y ataque por ser características audibles y mesurables.

### 5.1.1. Frecuencia

- *Analyzer*

  *Analyzer~* (Fig. 5.4.) es un objeto del lenguaje Max que analiza un audio recibido, ofreciéndonos el nivel de ruido del mismo, su composición frecuencial o la detección de ataques, entre otras características. Este objeto debe explicarse en detalle, ya que facilita enormemente la obtención de parámetros del análisis auditivo y constituye uno de los fundamentos del presente trabajo.

  Como elemento artístico se hace uso del tono, definido por una nota musical. La escala musical occidental divide la octava (el intervalo musical asociado con un ratio 2:1) en doce subintervalos iguales. La escala por tanto, se compone por doce subintervalos, o semitonos, que corresponden a un ratio de $2^{\frac{1}{12}}$ . La escala logarítmica de tonos más usada es la escala MIDI, en la cual la nota 69 está asignada a una frecuencia de 440 ciclos por segundo. Usando esta nota como referencia, la escala se construye multiplicando o dividiendo por $2^{\frac{1}{12}}$ para obtener el semitono inmediatamente superior o inferior respectivamente. Así, para convertir tonos MIDI, $m$, a frecuencias en ciclos por segundo, $f$, se aplican las fórmulas de conversión

tono/frecuencia [6]:

$$m = 69 + 12 \times \log_2 \frac{f}{440} \tag{5.1}$$

$$f = 440 \times 2^{\frac{m-69}{12}} \tag{5.2}$$

La detección del tono en *analyzer~* está fundamentado en *fiddle*, definido como un detector de tono. Creado por Miller Puckette, *fiddle* devuelve una estimación de tonos percibidos y su correspondiente amplitud a partir de un sonido de entrada. Esta puede darse de dos maneras diferentes: una lista con la sucesión de notas MIDI estimadas según la frecuencia detectada y la salida continua del valor absoluto de esta frecuencia. Opcionalmente, *fiddle* ofrece una lista con los picos de la señal sinusoidal detectados y empleados para determinar el tono. Al hacer uso de transformadas de Fourier, los argumentos necesarios para crearlo son el tamaño de la ventana de análisis, el máximo número de tonos simultáneos que se van a intentar detectar (en caso de polifonía), el número de picos en el espectro a considerar durante el análisis y, de estos, cuántos se van a dar a la salida. Es posible que el tono, medido en unidades MIDI, sea declarado nulo, significando que no ha sido posible identificar ninguno [6].

Tanto *fiddle* como *analyzer~* basan sus cálculos en las Transformadas de Fourier. De manera general, una Transformada de Fourier es un procedimiento matemático que mapea cualquier señal analógica estacionaria a una serie infinita de sinusoides, cada una de ellas con una determinada amplitud y fase. Considerando la magnitud de estas, es posible detectar las frecuencias predominantes, y así, el tono.

Para adaptar el análisis de Fourier al dominio de las señales muestreadas, de duración finita y variantes respecto al tiempo, se define la transformada discreta de Fourier localizada o a corto plazo (STFT). Para ello, se divide la señal en pequeñas porciones o segmentos, que formarán una secuencia de ventanas temporales. El contenido frecuencial o espectro se calcula en cada una de las tramas utilizando la transformada de Fourier. Esto tiene la desventaja de producir distorsiones en la medida del espectro, ya que el analizador de espectro no mide sólo la señal de entrada sino el producto de la misma por la ventana.

A continuación del enventanado, la STFT aplica la transformada de Fourier discreta (DFT) a cada segmento enventanado. Aquí se emplea la transformada rápida de Fourier (FFT), que es sencillamente una implementación eficiente de la DFT [25].

*Analyzer~* toma como argumentos el tamaño de búfer, dimensión de salto entre una FFT y la siguiente (implicando solape en el enventanado o no), tamaño de la FFT, tipo de ventana, retardo inicial, número de tonos simultáneos a extraer, como hacía *fiddle*, cantidad de picos a encontrar y de salida, y el tipo de formato para la salida *Bark*. La escala de Bark es una descomposición del modelo de espectro auditivo, cuyo número de salidas depende de la frecuencia de muestreo. En la versión actual de *analyzer~* esta frecuencia está fijada a 44.1 kHz, proporcionando 25 bandas de frecuencia. En la Fig. 5.1. se muestra una representación de la salida *Bark* cuando la señal de entrada es una sinusoide.

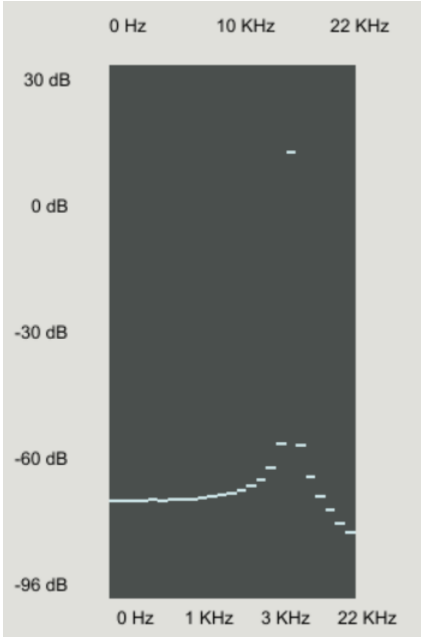



Fig. 5.1. *Bark* para una sinusoide

Para un resultado más cercano a la percepción humana, se escalan las diferentes frecuencias de acuerdo con la fisiología de la audición. Para ello se usa una escala logarítmica para las amplitudes y se separan las frecuencias de manera no lineal en fragmentos, asignándoles un descriptor relevante, como los coeficientes Bark.

El oído medio del ser humano atenúa los sonidos que recibe, haciendo de filtro, cuya respuesta frecuencial puede verse en la Fig. 5.2. El comportamiento del sistema de audición humano puede modelarse con una sucesión de filtros de banda crítica. La escala de bandas Bark proporciona uno de los modelos más usados [7].

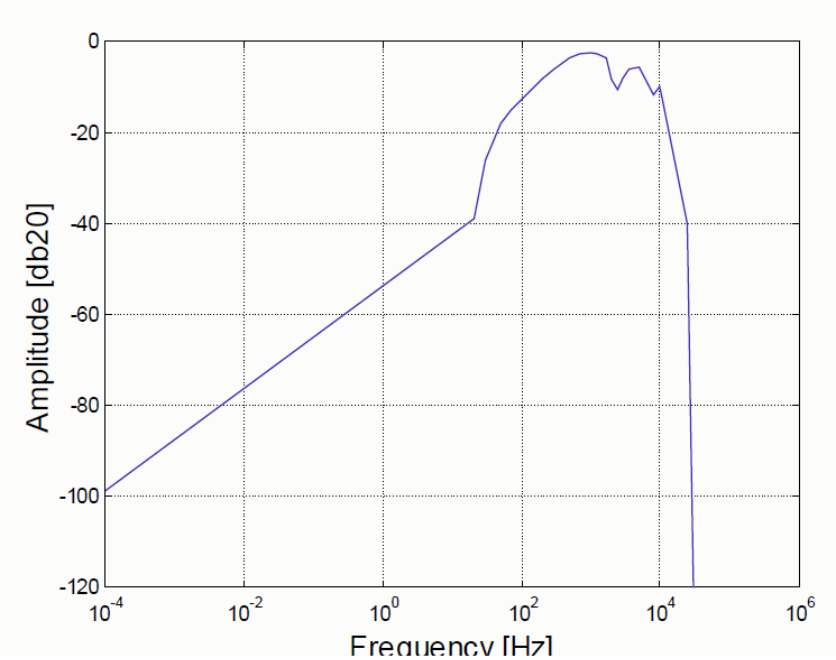



Fig. 5.2. Respuesta frecuencial del filtro atenuante del oído medio [7]

Basado en los resultados de diversos experimentos psicoacústicos, la escala Bark [10] está definida de manera que las bandas críticas de la audición humana tienen un ancho de 1 Bark [8]. Esta escala (Fig. 5.3.) contiene 24 Barks [1], que corresponden con las primeras 24 bandas críticas de audición. La fórmula 5.3. [7] muestra la relación entre la frecuencia expresada en Bark ($B$) y en Hz ($f$).

$$B = 13 \times atan\frac{f}{1315{,}8} + 3{,}5 \times atan\frac{f}{7518} \tag{5.3}$$

[1] La escala Bark se define sólamente hasta los 15.5 kHz, por lo tanto la máxima frecuencia de muestreo, determinada por el límite de Nyquist, es 31 kHz. *Analyzer* da una 25ª banda Bark, que supera los 19 kHz dado que la frecuencia de muestreo de este software está fijado a 44.1 kHz. Esto se hace debido al límite de la audición humana, que generalmente se extiende hasta los 20 kHz [8].

TABLE I.

| Number | Center frequencies Hz | Cut-off frequencies Hz | Bandwidth Hz |
|---|---|---|---|
| | | 20 | |
| 1 | 50 | 100 | 80 |
| 2 | 150 | 200 | 100 |
| 3 | 250 | 300 | 100 |
| 4 | 350 | 400 | 100 |
| 5 | 450 | 510 | 110 |
| 6 | 570 | 630 | 120 |
| 7 | 700 | 770 | 140 |
| 8 | 840 | 920 | 150 |
| 9 | 1000 | 1080 | 160 |
| 10 | 1170 | 1270 | 190 |
| 11 | 1370 | 1480 | 210 |
| 12 | 1600 | 1720 | 240 |
| 13 | 1850 | 2000 | 280 |
| 14 | 2150 | 2320 | 320 |
| 15 | 2500 | 2700 | 380 |
| 16 | 2900 | 3150 | 450 |
| 17 | 3400 | 3700 | 550 |
| 18 | 4000 | 4400 | 700 |
| 19 | 4800 | 5300 | 900 |
| 20 | 5800 | 6400 | 1100 |
| 21 | 7000 | 7700 | 1300 |
| 22 | 8500 | 9500 | 1800 |
| 23 | 10 500 | 12 000 | 2500 |
| 24 | 13 500 | 15 500 | 3500 |

Fig. 5.3. Bandas Bark [11]

El objeto *analyzer~* hace una serie de cálculos para obtener valores que representan los tonos detectados en cada instante de tiempo, los ataques detectados en la envolvente de la amplitud, frecuencia y amplitud de los picos detectados mediante la FFT, la amplitud del audio en el dominio temporal y, opcionalmente, una secuencia de mensajes con los picos detectados, medido en Hertzios. Un análisis se hace tras un número de muestras igual a la mitad del tamaño de ventana de análisis definido, n, sobre las últimas n muestras.

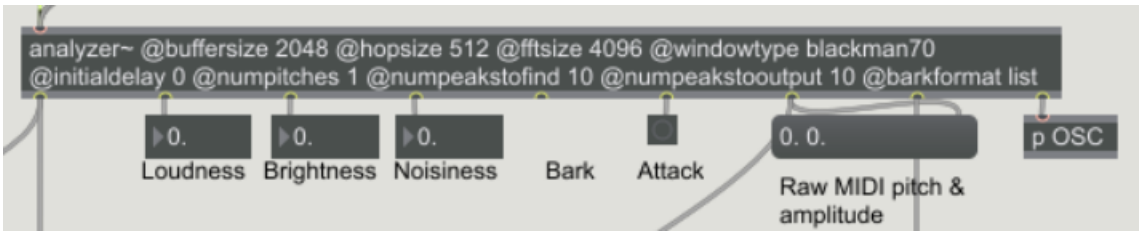



Fig. 5.4. *Analyzer*

Las salidas de *analyzer~* son, por tanto: una lista de tonos, tanto en frecuencia como su equivalencia en MIDI, sonoridad, brillo, ruido, descomposición del espectro como una lista de 25 bandas (*Bark*), detección del ataque, una lista de tonos MIDI con su amplitud y la descomposición sinusoidal de la señal de entrada, incluyendo hasta 100 picos detectados. La sonoridad mide la energía espectral de la señal en cada instante y el brillo está definido

por la relación entre las frecuencias que componen una señal y su magnitud (mayor o menor amplitud) [7].

Se elige una ventana de 2048 muestras, siendo 4096 el tamaño de la FFT en saltos de 512 muestras ya que son medidas estándar que aseguran un equilibrio entre la exactitud de los resultados obtenidos y el coste computacional de las operaciones, considerando que además, el retardo que suponen las operaciones es aceptable respecto del sonido, siendo este aproximadamente 5 ms ($\frac{2048}{44100}$).

- Filtros

  Otro modo en que se ha podido diferenciar entre los rangos de frecuencia ha sido a través de filtros (Fig. 5.5.). Para ello se ha empleado el objeto *biquad~*, que implementa un filtro con dos polos y dos ceros. Para facilitar la definición de los parámetros de este filtro, se le ha unido el objeto *filtercoeff~* que, al recibir los valores de frecuencia central, parámetro Q y ganancia, genera los coeficientes del filtro deseado. Por tanto, si se desea saber qué rango de frecuencias es predominante en la señal de audio, y no la frecuencia exacta, se puede dividir el espectro en varios segmentos aplicando un filtro para cada rango de frecuencias.


Fig. 5.5. Ejemplo de creación de un filtro

### 5.1.2. Amplitud

- *Analyzer*

  Obtener los valores de amplitud de cada frecuencia se puede hacer a través de *analyzer~*. Tanto en la salida de *Bark*, como en la de *polyphonic pitches* y *sinusoidal components*, este objeto ofrece una lista (Fig. 5.6.) con diferentes frecuencias y la intensidad con que cada una de ellas aparece en el espectro del audio analizado.

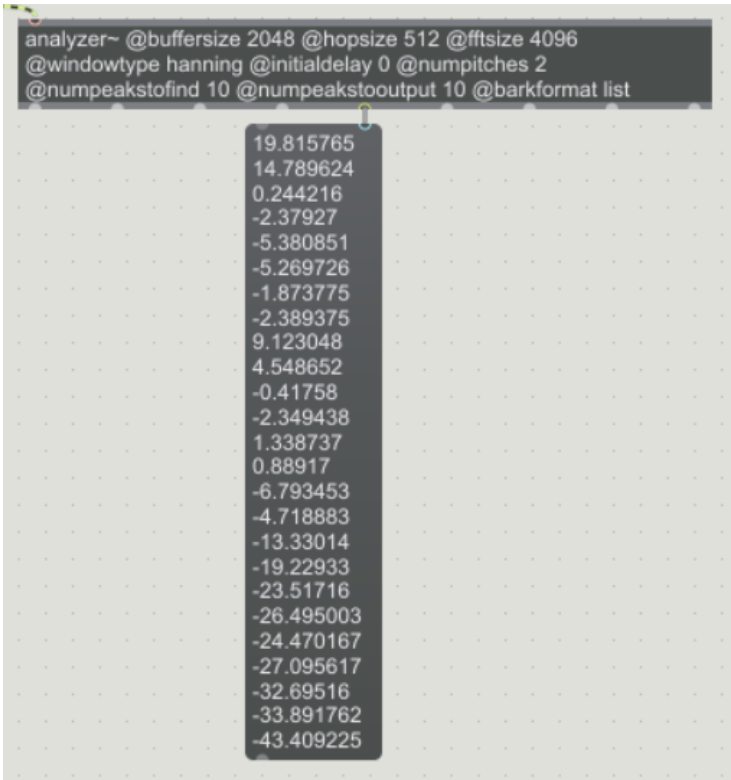



Fig. 5.6. Lista de valores a la salida *Bark*

- *Peakamp~*

  En Max 7 existen objetos, como *peakamp~*, que dan el valor máximo de amplitud de la señal en un intervalo de tiempo que podemos definir, en milisegundos, por lo que obtener un parámetro útil con estos valores es sencillo.

  En general en los proyectos desarrollados, en primer lugar se pasa el nivel de señal a decibelios, dada la naturaleza logarítmica de la audición humana [6]. Seguidamente se le suma cierta cantidad, ya que hemos de tener en cuenta qué niveles de audio son considerados máximos y mínimos. Por experiencia, este valor es de 40 dBs, ya que al medir la amplitud de la señal con *peakamp~*, se ha hallado que niveles superiores a -35 dBs son audibles, y el máximo siempre será 0 (el rango de valores de salida de *peakamp~* es de 0 a 1). Se quiere contar con valores únicamente positivos para facilitar los cálculos.

  En ocasiones también se ha añadido *line~*, que impide cambios abruptos en la sucesión de valores de amplitud, permitiendo posteriormente un movimiento fluido de los elementos visuales a representar. Generalmente, esta conversión se hace dentro del subpatch *ampMov* (Fig. 5.7.), que se emplea en varios de los diseños.


Fig. 5.7. Algoritmo *ampMov*

### 5.1.3. Ataque

- *Analyzer*

  *Analyzer~* también se ha empleado para obtener el ataque, que se ha definido en este contexto como un impulso que se recibe cuando la amplitud de la señal, en un periodo de 100 milisegundos, ha aumentado más de 4 dBs. Estos valores permiten que los ataques representen aproximadamente el ritmo del sonido analizado.

- Manual

  Es sencillo, sin embargo, reproducir la función de ataque (Fig. 5.8.) de *analyzer~* con objetos más básicos. Aquí se compara la amplitud de la señal en cada instante con esta misma señal retrasada un tiempo determinado, y si esta diferencia es superior al umbral fijado, el objeto *if~* que se emplea para ello da a su salida un impulso.

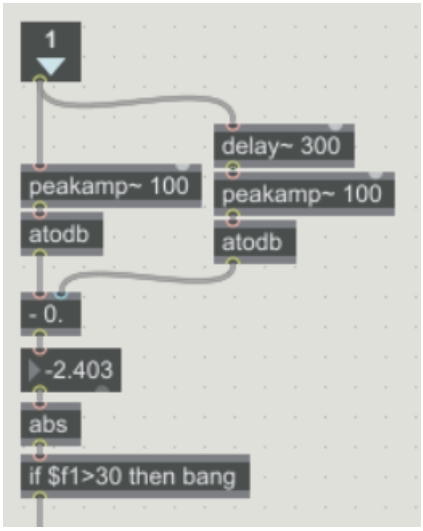



Fig. 5.8. Definición manual del ataque

## 5.2. Vídeo

Como se ha comentado en el capítulo “3: Herramientas”, en estos proyectos se va a hacer uso de los objetos de Jitter que ofrece Max. Dentro de Jitter, OpenGL permite generar gráficos 3D dinámicamente, y Physics los hace interactuar. Es con objetos de estos dos tipos con los que se va a constituir la pantalla.

### 5.2.1. El *mundo*

En este documento se va a hacer referencia en numerosas ocasiones a lo que se ha decidido llamar *mundo* (Fig. 5.9.). Este se refiere al espacio 3D generado por Max donde se van a representar todos los objetos y donde va a suceder toda la acción. Se llama así por el objeto *world* de Physics que se va a usar, que dota de características físicas a este entorno.

El primer paso para representar los gráficos es crear una pantalla *jit.window* y un objeto *jit.gl.render* vinculado, que dibuje en ella objetos Jitter OpenGL, gráficos 3D. A este último se ha unido el objeto *jit.phys.world*, que encapsula en un contexto de simulación física los elementos creados, de manera que se convierten en objetos rígidos que pueden colisionar y verse afectados por diferentes fuerzas, como la gravedad [14].

Funciones con opción de ser activadas o no se han añadido a todos los proyectos. Estas son *jit.gl.physdraw*, que, activado, dibuja los ejes de los elementos del mundo, *jit.gl.handle*, que modifica el ángulo de visión del mundo o atributos como *fullscreen*, que pone la ventana en modo de pantalla completa.

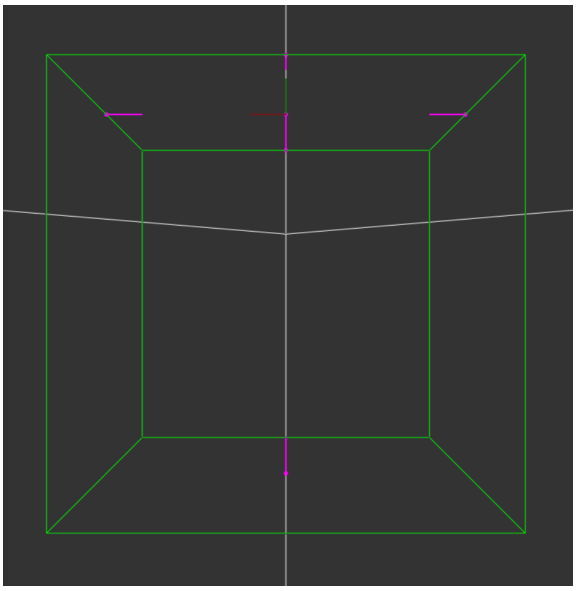
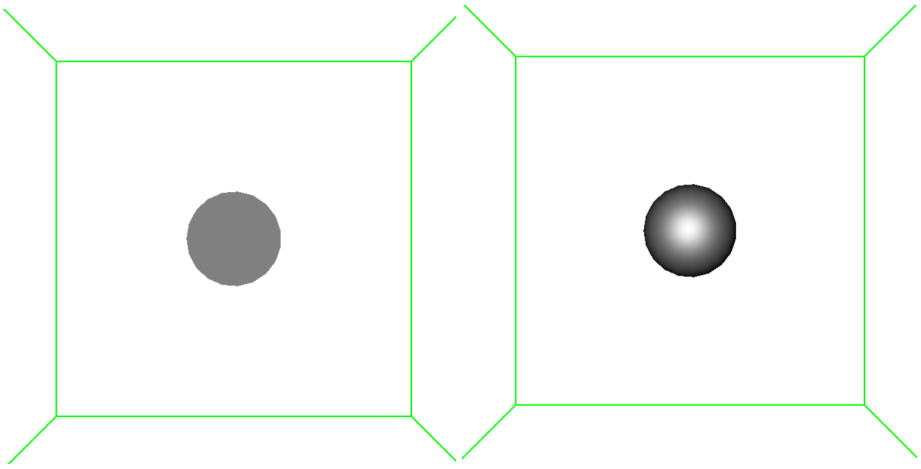


Fig. 5.9. Representación del *Mundo*

### 5.2.2. Objetos

Los cuerpos se crean en Max fundamentalmente con *jit.gl.gridshape*, como una malla de una de las siguientes formas: esfera, anillo, cilindro, cilindro abierto, cubo, cubo abierto, plano o círculo. A este se vincula el objeto *jit.gl.material* para cubrir la malla con una textura y darle apariencia de cuerpo rígido (Fig. 5.10.).



(a) Sin textura (b) Con textura

Fig. 5.10. *Gridshape*

Se pueden definir infinidad de atributos de estos cuerpos, como forma, escala, posición, rotación, número de líneas con que se va a crear la malla que forma su contorno, si se permite o no que el objeto se vea afectado por la iluminación, o que se difumine la sombra o muestre profundidad,

entre otros ejemplos.

En uno de los diseños se usa *jit.gl.plato* (Fig. 5.11.) en lugar de gridshape, que genera uno de los siguientes sólidos poliédricos: tetraedro, cubo, octaedro, dodecaedro o icosaedro, pero el funcionamiento es similar al de *gridshape*.

Fig. 5.11. Icosaedro creado con *jit.gl.plato*

A estos cuerpos se les puede vincular el objeto de Physics *jit.phys.body*, que representa un cuerpo rígido y la forma de colisión en una simulación física. Así, al objeto creado con *gridshape*, se le puede dar un valor de masa, elasticidad o par de torsión (producto entre una fuerza aplicada a un objeto y el radio de acción de dicha fuerza), entre otros atributos.

Se puede también generar una fuerza sin necesidad de vincularla a un objeto mediante *jit.phys.ghost* (Fig. 5.12.), que simplemente define de qué manera esta va a afectar a los *jit.phys.body* del entorno: valor y dirección de la fuerza de atracción o repulsión, desde qué puntos se aplica, cuánto espacio abarca, etc.

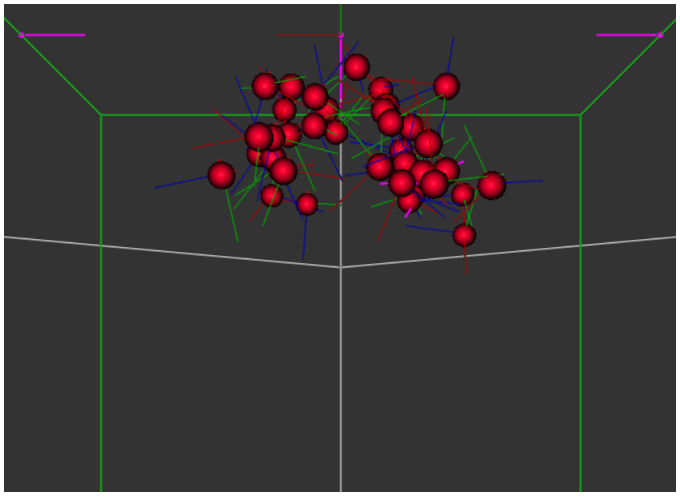
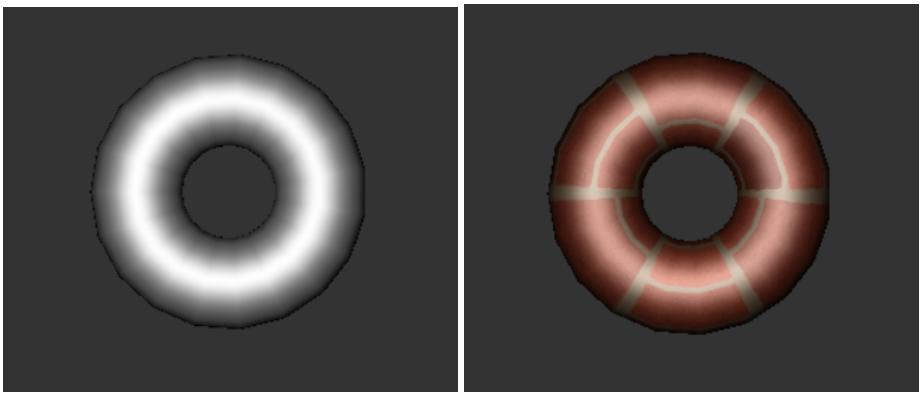


Fig. 5.12. Grupo de objetos afectados por una fuerza creada con *jit.phys.ghost*

Finalmente, en alguno de los diseños, en lugar de usar *jit.gl.gridshape* y *jit.phys.body* se han empleado *jit.gl.multiple* y *jit.phys.multiple*, generando un número definido de elementos independientes, pero de iguales características.

### 5.2.3. Texturas

Como se ha mencionado, *gridshape* tiene vinculado un material que dará cierta textura al cuerpo en cuestión. Esta textura puede ser simplemente una superficie lisa, establecida por defecto, una textura contenida en un archivo externo de imagen, o texturas en movimiento a partir de matrices (Fig. 5.13.).



(a) Textura estándar (b) Imagen aplicada

Fig. 5.13. Cuerpo con diferentes texturas aplicadas

En los diseños de este proyecto se ha hecho uso de todas ellas, y merecen atención ciertos objetos y algoritmos que se han creado para generar matrices (vídeo) a proyectar en la superficie de algunos de los objetos.

- *Poke*

  El objeto *jit.poke~* (Fig. 5.14.) escribe el nivel de señal de un audio recibido en una matriz. De la manera en que se va a usar aquí, se crea primero una matriz de tamaño $n \times m \times 3$ a la que se le da un nombre (*poked* en este caso). Se crearán tres objetos *poke*, cada uno de ellos definiendo uno de los tres planos de la matriz creada. Estos son los valores R (Rojo), G (Verde) y B (Azul) de cada pixel de una pantalla. *Poke* escribe el valor de su entrada 1 en la posición *m*, *n* especificada por sus entradas 2 y 3 respectivamente. Estas dos últimas son iguales para los tres objetos *poke*, por lo tanto el resultado es que en cada instante se pinta un pixel de la pantalla de un color definido por la primera entrada de los tres objetos (valores R, G y B).

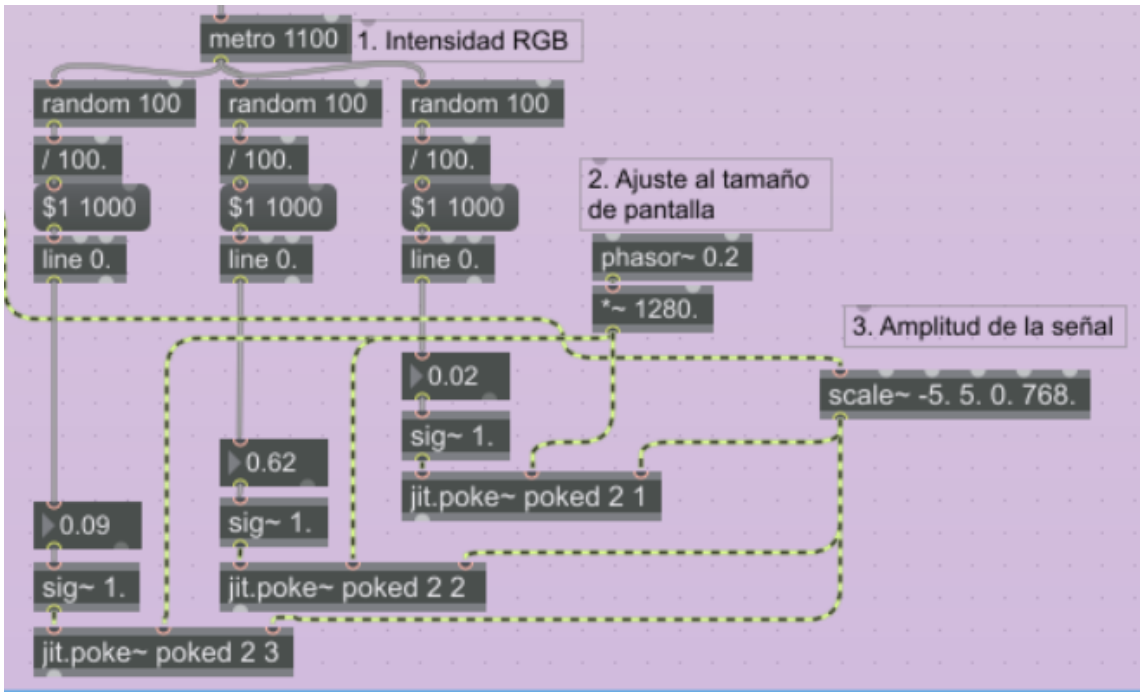



Fig. 5.14. Definición de los tres objetos *poke*

El eje *x* está definido por un fasor (señal diente de sierra). Este tipo de señales crece linealmente hasta alcanzar la máxima amplitud y cuando esto ocurre regresa inmediatamente a cero para volver a empezar. Un fasor multiplicado por el tamaño de pantalla resulta en que la pantalla sea recorrida de izquierda a derecha a la frecuencia definida por esta señal y al acabar vuelva al extremo izquierdo de la pantalla para hacerlo de nuevo. El eje *y* es el nivel de señal recibido, el sonido. Por lo tanto, el resultado es la representación de la amplitud de señal de audio en el dominio temporal (Fig. 5.15.). En el caso del sonido, la amplitud representa la variación de la presión atmosférica respecto al tiempo. Se representa a partir del valor 0 (posición de equilibrio o valor medio de la presión) hasta el punto de máxima amplitud de la forma de onda [25].

Es fácil a partir de aquí convertir esta pantalla en una textura por medio de *jit.gl.material*.

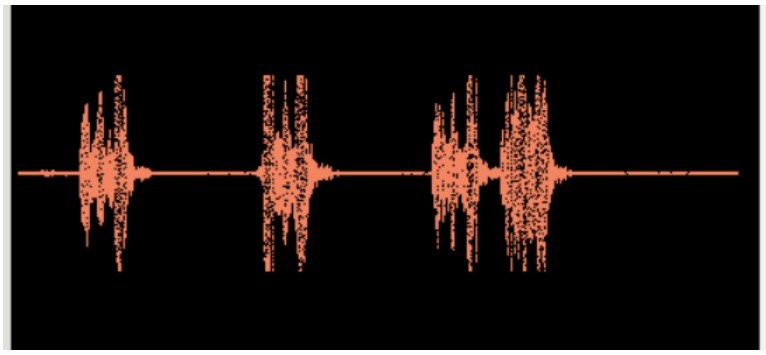


Fig. 5.15. Representación del audio en el dominio temporal, generado por *poke*

- Espejos

  Para dar este efecto el escenario está rodeado de *cámaras* que proyectan la imagen que capturan en cada pared [13]. El espejo (Fig. 5.16.) se ha conseguido gracias a los objetos *jit.gl.camera* y *jit.gl.videoplane*. El primero establece las propiedades necesarias para definir la perspectiva en OpenGL. Esto incluye un campo de visión desde una posición y ángulos determinados, el corte de los planos capturados (las paredes en nuestro caso) y la perspectiva o modo de proyección ortográfica, capturando los lados, sombras, focos de luz, etc de los objetos del entorno. Por su lado, *jit.gl.videoplane* expone el vídeo que la cámara recoge. El vínculo entre los dos se hace a través de *jit.gl.slab*, que procesa la textura que la cámara va creando [14].

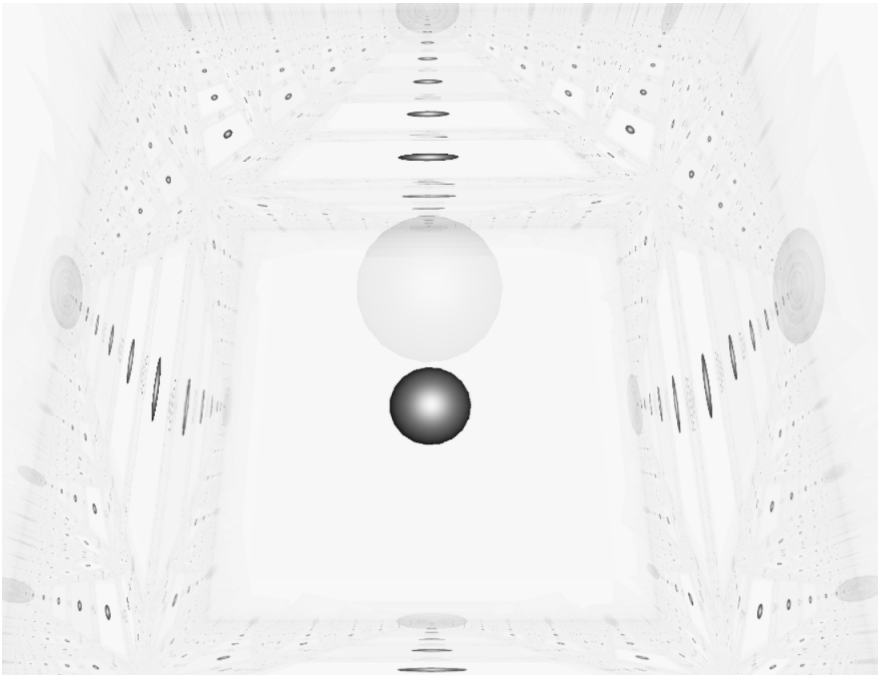


Fig. 5.16. Efecto de espejo

Por último, se ha de mencionar que los colores se definen en Max con 4 números comprendidos entre 0 y 1. Los tres primeros corresponden a los valores R (rojo), G (verde) y B (azul), siendo por tanto 0, 0, 0, el color negro, y 1, 1, 1, el blanco. El cuarto valor corresponde a la componente alfa, la opacidad.

# 6. APLICACIÓN

## 6.1. *Drops*

Para el análisis del audio, en este proyecto se usa el objeto *analyzer~* del cual podemos extraer como interesantes las salidas de *Attack* y *Bark*. La señal de ataque se da cuando en un periodo de 100 milisegundos se ha producido un cambio en el nivel de sonido mayor a 4 dBs.

La escala de *Bark* surge de *analyzer~* como una lista de 25 valores de amplitud, que se escalan para mostrarse representados como el valor $y$ de la posición de las gotas que se exhiben, tomando de nuevo como referencia el sistema de coordenadas que muestra la Fig. 4.1. Los valores $x$ y $z$ serán siempre los mismos.

Las dimensiones del mundo son $10m \times 10m \times 10m$ y se establece como punto de origen 0, 0, 0, el centro de este espacio. El eje $z$ tomará el valor cero para los 25 grupos de gotas. El eje $x$ se ha dividido en 25 partes iguales, desde la posición -5 del eje hasta la 5, definiendo los 25 puntos equidistantes donde se va a representar cada uno de los grupos de gotas. Así, las esferas aparecerán formando una línea a lo largo del eje $x$ en el centro del *mundo*. También, del mismo modo, se ha definido el color de cada grupo de gotas, asignándoseles ascendentemente en la escala Bark un valor entre 0 y 1 de color rojo, de 1 a 0 verde y 1 de azul en el valor RGB que especifica la tonalidad de los objetos. Es decir, las gotas de la primera banda de Bark aparecen pegadas a la pared izquierda de un tono celeste, y las de la última banda, la $25^a$, junto a la pared derecha en color magenta.

Esta representación se dará gracias al objeto *poly~* (Fig.6.1.), que encapsula un *subpatch* dentro de una caja objeto. Como se ha descrito en el apartado 3.2., el *subpatch* es un tipo de encapsulado de *patches*, que facilita la comprensión y el manejo de algoritmos y diseños presentes en un proyecto. *Poly~* requiere que se introduzca el número de veces que queremos que el objeto definido en este nuevo *subpatch* aparezca en nuestro proyecto. Cada una de las copias puede modificarse independientemente, si la seleccionamos a través del mensaje *target*, que apunta a un elemento específico de dicho grupo. Así, mandando el mensaje *target 1* a *poly~*, los datos de entrada a este objeto solo afectarán al primer elemento, *target 2* al segundo, y así sucesivamente. La excepción es *target 0*, que se usa cuando se quiere que todos los elementos del grupo reciban estos datos de entrada de igual manera.

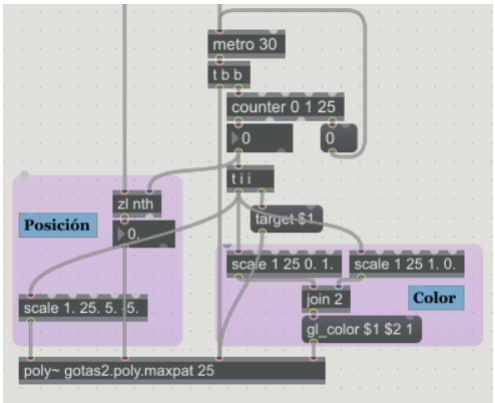



Fig. 6.1. Uso de *Poly~*

El objeto creado con *poly~* representará un grupo de gotas (Fig. 6.2.), cuya definición deberá de hacerse por medio de matrices, al tratarse de un conjunto de elementos y no uno solo. A través de operaciones con la matriz de creación y los parámetros de posición arriba definidos, se crea el grupo de esferas, que aumentarán y disminuirán de tamaño, para simular el efecto de las gotas al caer. En este caso las matrices son necesarias ya que en el grupo cada gota tendrá una posición y un tamaño, y deberá actuar independiente de las demás a las fuerzas del *mundo*, de ahí la necesidad de usar *jit.phys.multiple*.

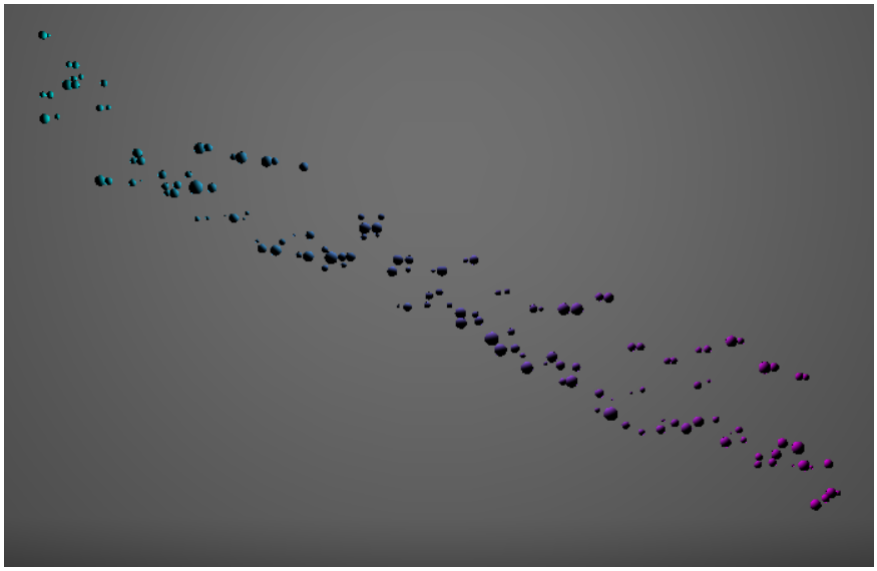


Fig. 6.2. Representación de las gotas

La sucesión de ataques en el sonido activará la representación de las gotas. En el instante en que se producen, la matriz de posiciones se actualiza con los valores para el eje *y*, que varía en el tiempo, más las posiciones *x* y *z*, que son constantes. Así, los veinticinco grupos de gotas reaparecen en la pantalla, según la amplitud de cada banda Bark, para volver a caer al suelo por efecto

de la gravedad. Para evitar que las gotas desaparezcan antes de llegar al suelo, se ha replicado este sistema tres veces (Fig. 6.3.), por lo tanto habrá 75 grupos de gotas, tres por cada posición en el eje *x*. Estas réplicas se activarán de forma alterna a cada impulso de la salida *attack*, así, mientras un grupo aún no ha terminado de caer, puede aparecer otro si se produce un ataque, en una nueva posición *y*, sin que se desvanezca el anterior.

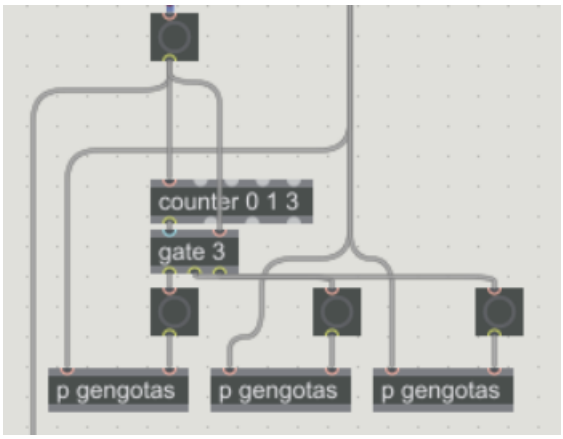



Fig. 6.3. Sistema de generación triplicado

El *subpatch ritmomedio* contiene un algoritmo que da a su salida el tiempo medio entre ataque y ataque equivalente a diez ataques consecutivos. Este tiempo es escalado para transformarlo en un valor de fuerza de atracción de la gravedad entre $-12m/s^2$ y $-1m/s^2$.

El ataque se usa también en el *subpatch susto*. Sin utilizar la salida de *analyzer~*, el ataque se define en este *subpatch* como un impulso al detectar una diferencia superior a 30 dBs en instantes separados por 300 ms, como se ha descrito en el apartado 5.1.3. En este caso se emplea este impulso (de mayores dimensiones al definido en *analyer~*) para esparcir por todo el espacio la totalidad de las esferas generadas, obteniéndose así una diferenciación entre el ritmo marcado por el sonido y los ataques de alto nivel repentinos que en este se puedan producir. Estos podrían corresponder a unos platillos de crash, o una explosión, un trueno o golpes bruscos y muy fuertes de sonido que merecen que en la imagen haya un cambio inesperado y grande. Al mismo tiempo, la gravedad se fija a un valor alto para que los objetos no queden suspendidos excesivamente en el tiempo después de que se produzcan este tipo de ataques.

La amplitud de audio de entrada se emplea por otro lado para regular la intensidad de la luz en el *mundo*, a través de *ampMov*. Este algoritmo hace que el silencio sea representado por la completa oscuridad, otorgando un 0 a los valores R, G y B del color de difusión de la luz, definida en *jit.gl.light* (Fig. 6.4). Para obtener máxima claridad en el entorno, este color debe ser blanco (valor 1, 1, 1, para los parámetros RGB del color de difusión de la luz, como se ha

explicado en el apartado 5.2.), que coincide con el nivel máximo de sonido registrado de amplitud (aproximadamente 40 dBs, fijados según la descripción de *peakamp~* del apartado 5.1.). El cambio es gradual gracias a *line~* y no se aceptan números que superen estos límites, ya que el objeto *zmap* asigna un rango de valores de entrada a uno de salida, sin que puedan ser sobrepasados.

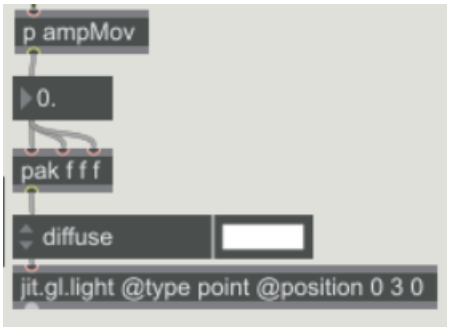
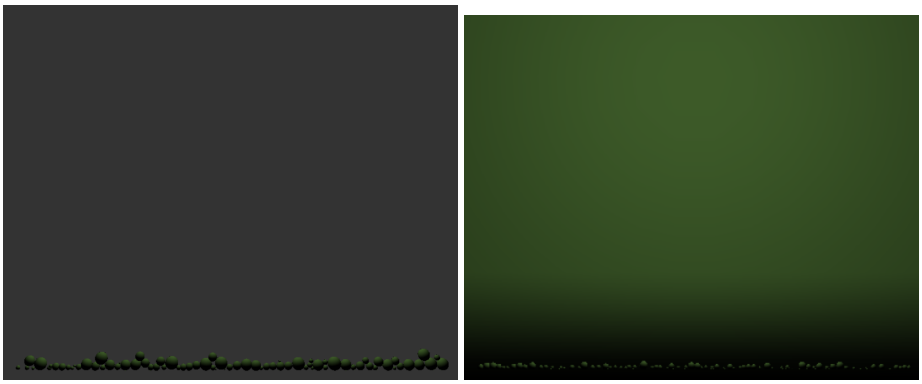



Fig. 6.4. Definición de la iluminación

Para que la difusión de la luz tenga efecto en el *mundo*, se han usado dos objetos planos *jit.gl.gridshape* para crear un suelo y pared de fondo donde se proyecte esta luz (Fig. 6.5.).



(a) Sin paredes (b) Con paredes

Fig. 6.5. Efecto de luz verde sobre el *mundo* sin (a) y con (b) paredes

## 6.2. *Attraction*

En este caso se ha optado por dividir el espectro en tres secciones creando filtros como se ha explicado en el apartado 5.1. La selección de los parámetros que los definen se ha hecho considerando la frecuencia de las notas de la escala musical. En la Tabla 1 se muestran las notas de la escala para las primeras nueve octavas con notación anglosajona, y su equivalencia en frecuencia. Cada nota dista de la anterior medio tono, como se ha explicado en el apartado 5.1.1., al definir los tonos MIDI.

| Octavas: nota y frecuencia (Hz) | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| $C_0$ | 16.35 | $C_1$ | 32.70 | $C_2$ | 65.41 | $C_3$ | 130.81 | $C_4$ | 261.63 | $C_5$ | 523.25 | $C_6$ | 1046.50 | $C_7$ | 2093.00 | $C_8$ | 4186.01 |
| $C^{\#}_0/D^{b}_0$ | 17.32 | $C^{\#}_1/D^{b}_1$ | 34.65 | $C^{\#}_2/D^{b}_2$ | 69.30 | $C^{\#}_3/D^{b}_3$ | 138.59 | $C^{\#}_4/D^{b}_4$ | 277.18 | $C^{\#}_5/D^{b}_5$ | 554.37 | $C^{\#}_6/D^{b}_6$ | 1108.73 | $C^{\#}_7/D^{b}_7$ | 2217.46 | $C^{\#}_8/D^{b}_8$ | 4434.92 |
| $D_0$ | 18.35 | $D_1$ | 36.71 | $D_2$ | 73.42 | $D_3$ | 146.83 | $D_4$ | 293.66 | $D_5$ | 587.33 | $D_6$ | 1174.66 | $D_7$ | 2349.32 | $D_8$ | 4698.63 |
| $D^{\#}_0/E^{b}_0$ | 19.45 | $D^{\#}_1/E^{b}_1$ | 38.89 | $D^{\#}_2/E^{b}_2$ | 77.78 | $D^{\#}_3/E^{b}_3$ | 155.56 | $D^{\#}_4/E^{b}_4$ | 311.13 | $D^{\#}_5/E^{b}_5$ | 622.25 | $D^{\#}_6/E^{b}_6$ | 1244.51 | $D^{\#}_7/E^{b}_7$ | 2489.02 | $D^{\#}_8/E^{b}_8$ | 4978.03 |
| $E_0$ | 20.60 | $E_1$ | 41.20 | $E_2$ | 82.41 | $E_3$ | 164.81 | $E_4$ | 329.63 | $E_5$ | 659.25 | $E_6$ | 1318.51 | $E_7$ | 2637.02 | $E_8$ | 5274.04 |
| $F_0$ | 21.83 | $F_1$ | 43.65 | $F_2$ | 87.31 | $F_3$ | 174.61 | $F_4$ | 349.23 | $F_5$ | 698.46 | $F_6$ | 1396.91 | $F_7$ | 2793.83 | $F_8$ | 5587.65 |
| $F^{\#}_0/G^{b}_0$ | 23.12 | $F^{\#}_1/G^{b}_1$ | 46.25 | $F^{\#}_2/G^{b}_2$ | 92.50 | $F^{\#}_3/G^{b}_3$ | 185.00 | $F^{\#}_4/G^{b}_4$ | 369.99 | $F^{\#}_5/G^{b}_5$ | 739.99 | $F^{\#}_6/G^{b}_6$ | 1479.98 | $F^{\#}_7/G^{b}_7$ | 2959.96 | $F^{\#}_8/G^{b}_8$ | 5919.91 |
| $G_0$ | 24.50 | $G_1$ | 49.00 | $G_2$ | 98.00 | $G_3$ | 196.00 | $G_4$ | 392.00 | $G_5$ | 783.99 | $G_6$ | 1567.98 | $G_7$ | 3135.96 | $G_8$ | 6271.93 |
| $G^{\#}_0/A^{b}_0$ | 25.96 | $G^{\#}_1/A^{b}_1$ | 51.91 | $G^{\#}_2/A^{b}_2$ | 103.83 | $G^{\#}_3/A^{b}_3$ | 207.65 | $G^{\#}_4/A^{b}_4$ | 415.30 | $G^{\#}_5/A^{b}_5$ | 830.61 | $G^{\#}_6/A^{b}_6$ | 1661.22 | $G^{\#}_7/A^{b}_7$ | 3322.44 | $G^{\#}_8/A^{b}_8$ | 6644.88 |
| $A_0$ | 27.50 | $A_1$ | 55.00 | $A_2$ | 110.00 | $A_3$ | 220.00 | $A_4$ | 440.00 | $A_5$ | 880.00 | $A_6$ | 1760.00 | $A_7$ | 3520.00 | $A_8$ | 7040.00 |
| $A^{\#}_0/B^{b}_0$ | 29.14 | $A^{\#}_1/B^{b}_1$ | 58.27 | $A^{\#}_2/B^{b}_2$ | 116.54 | $A^{\#}_3/B^{b}_3$ | 233.08 | $A^{\#}_4/B^{b}_4$ | 466.16 | $A^{\#}_5/B^{b}_5$ | 932.33 | $A^{\#}_6/B^{b}_6$ | 1864.66 | $A^{\#}_7/B^{b}_7$ | 3729.31 | $A^{\#}_8/B^{b}_8$ | 7458.62 |
| $B_0$ | 30.87 | $B_1$ | 61.74 | $B_2$ | 123.47 | $B_3$ | 246.94 | $B_4$ | 493.88 | $B_5$ | 987.77 | $B_6$ | 1975.53 | $B_7$ | 3951.07 | $B_8$ | 7902.13 |

Tabla 1. Notas y frecuencias de la escala musical [9]

Se ha decidido dividir el espectro en octavas, contando con las ocho primeras. Las octavas se cuentan de Do (C en el sistema anglosajón) a Do, abarcando doce semitonos. Siguiendo la Tabla 1, el filtro de frecuencias bajas contará con 3 octavas (C0 - C3), el de frecuencias medias con 2 octavas (C3 - C5) y el de las altas de nuevo con 3 (C5 - C8).

Deben darse al objeto de filtrado *filtercoeff~* (Fig. 5.2.) tanto la frecuencia central como el parámetro Q, que se define como el cociente entre la frecuencia central y el ancho de banda del filtro, como muestra la Ec. 6.1. El cálculo de los parámetros de los filtros a emplear queda de la siguiente manera:

| | Frec. Inicial (Hz) | Frec. Final (Hz) | Ancho de banda (Hz) | Frecuencia central (Hz) | Q |
|---|---|---|---|---|---|
| Bajas | 16,35 | 130,8 | 114,45 | 73,575 | 0,642 |
| Medias | 130,8 | 523,3 | 392,5 | 327,05 | 0,833 |
| Altas | 523,3 | 4186,01 | 3662,71 | 2354,655 | 0,642 |

Tabla 2. Parámetros de los filtros

$$Q = \frac{f_c}{\Delta f} \tag{6.1}$$

Así, los *subpatches bajas*, *medias* y *altas* contienen estos filtros, seguidos del algoritmo *amp-Mov* definido en el apartado 5.1., que escala la amplitud de la señal de salida del filtro a un valor entre 0 y 1. Este número decimal definirá la intensidad del color de los objetos que corresponden a cada filtro.

Estos tres *subpatches* (*bajas*, *medias* y *altas*) son esencialmente iguales, aunque para distinguir unas frecuencias de otras se ha elegido que los objetos tengan diferentes intensidades de color (Fig.

6.6.), siendo azul para las bajas, verde para las medias y rojo para las altas. Además, el tamaño de estos objetos estará también definido por el rango de frecuencias que cubre: de radio igual a 0,6 m para las bajas, 0,45 para las medias y 0,3 las altas.

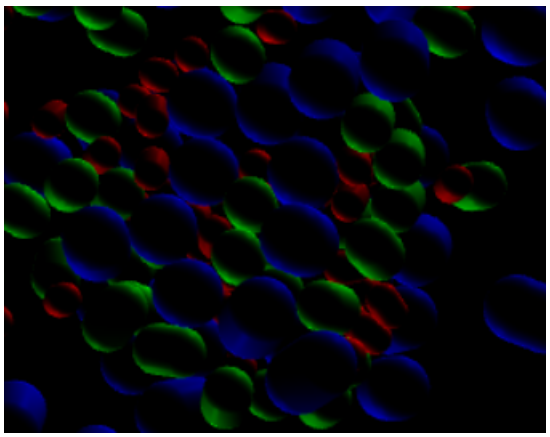


Fig. 6.6. Diferencia entre objetos según la zona del espectro que representan

La representación gráfica se hace a través de *jit.phys.multiple* nuevamente. Se crea primero una matriz de valores aleatorios a partir de tres planos y cincuenta elementos. Esto supone la creación de 50 objetos independientes en posición, escala y rotación, con la finalidad de que las fuerzas actúen sobre ellos por separado, no como un conjunto aglomerado. Estas cualidades físicas se definen con los *jit.phys.multiple* que siguen a *jit.noise*, generador de matrices de valores inicialmente aleatorios, y *jit.gl.gridshape* con lo que se consigue dar cuerpo a cada elemento físico, como se ha explicado en el apartado 5.2.

La forma de los elementos de este entorno se asigna de forma aleatoria, como describe el *subpatch cambioForma* (Fig. 6.7.). El momento en que se da el cambio de forma se recibe del *subpatch silencio*, que mide el tiempo que transcurre desde que comienza un silencio en el audio hasta el momento en que se vuelve a producir sonido. También se detecta si este tiempo es superior a un minuto, y se lanza el mensaje de cambio de forma en el instante en que esto ocurre.

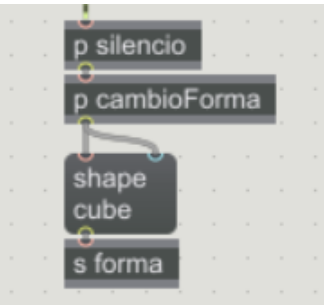



Fig. 6.7. Relación *silencio* y *cambioForma*

Por otro lado, se crea una esfera central (Fig. 6.8.), cuya función es representar el volumen del audio sin filtrar. Para ello, se recurre una vez más a *ampMov* encargado de convertir su salida en el valor del radio de este objeto.

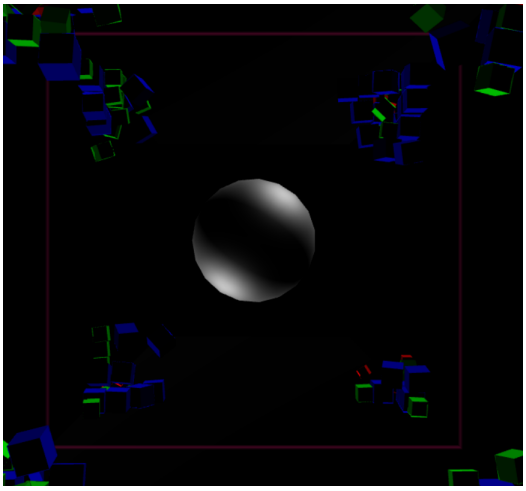


Fig. 6.8. Esfera central

Como se ha descrito en el apartado 4.2., cada vez que se produce un cambio repentino de volumen (ataque) en el audio, definiendo su ritmo, una fuerza proporcional al tiempo que ha pasado entre un ataque y el siguiente repelerá a los objetos del *mundo*. El algoritmo que define esto está contenido en el *subpatch ataque*, donde, además de establecer cómo se producen los ataques, compara el tiempo entre ataques y si en 2 segundos no se ha producido ninguno, también lanza un mensaje de activación. La salida del *subpatch* define la magnitud del atributo *central_impulse* de un *jit.phys.ghost*, que aleja los objetos del centro, de la manera que muestra la Fig. 6.9.

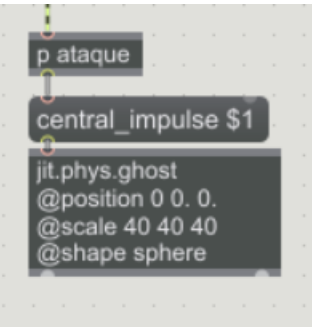



Fig. 6.9. Relación *ataque* y *jit.phys.ghost*

Para que los objetos tiendan a volver al centro después de los *ataques*, se crea otro objeto *jit.phys.ghost* que los atrae a la posición 0 0 0, manteniéndose activo de forma permanente.

Por otro lado, las paredes del entorno se han definido como un cubo a través de *jit.gl.plato*. Esto es así para poder aplicar a este entorno una textura que afecte a todas ellas como conjunto. Como se define en el apartado 5.2. se crea una matriz a través de *poke* que representa en una pantalla la señal de audio en el dominio temporal. Esta matriz se pasa a *jit.gl.material* que, aplicado a *jit.gl.plato*, ofrece el efecto de que la línea de amplitud del sonido vaya recorriendo las cuatro paredes progresivamente. Los valores R, G y B de los correspondientes objetos *poke* se mueven aleatoria y progresivamente entre 0 y 1, produciendo el efecto de color que se aprecia en la Figura 6.10.

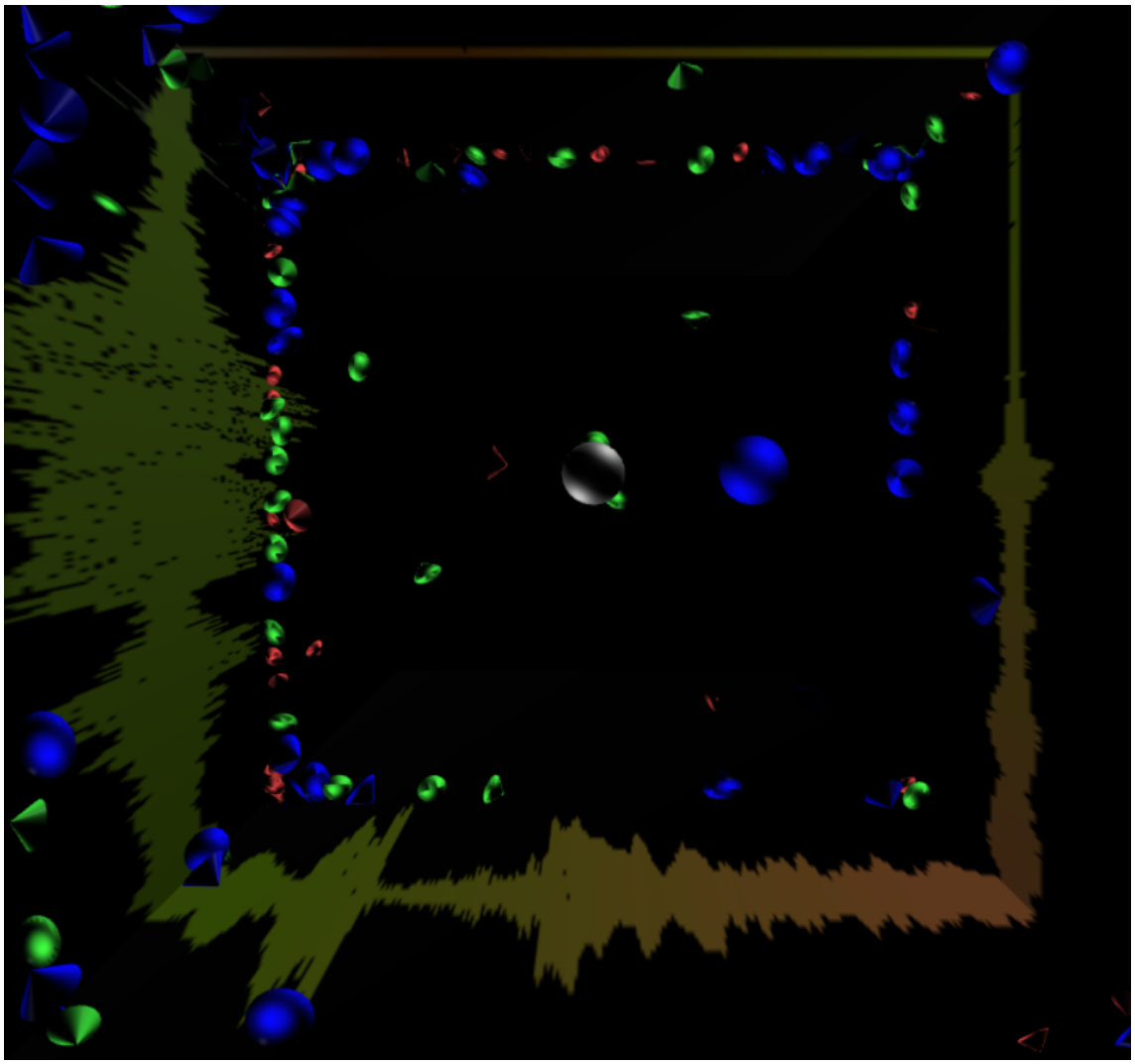


Fig. 6.10. Representación *poke* sobre las paredes del entorno

Finalmente, se añaden dos objetos *jit.gl.light* generadores de dos puntos de luz direccionales para iluminar estéticamente el entorno.

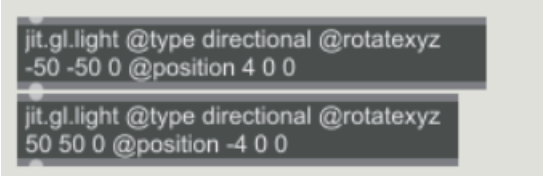



Fig. 6.11. Definición de los focos de luz

### 6.3. *Mirrors*

En este caso se ha usado, al igual que en *Drops*, el objeto *analyzer*, pero en lugar de usar la salida de *Bark*, se ha empleado *Polyphonic pitches*, que proporciona una lista con la frecuencia y

amplitud de las tres primeras frecuencias fundamentales detectadas, que idealmente se corresponderían con tres notas de un acorde. Se aprovechan los dos primeros, ya que experimentalmente se ha comprobado que son los que mejor representan la sensación percibida por nuestro oído.

Para estos dos parciales se crean los *subpatches formasParcial1* y *formasParcial2*, que son esencialmente iguales. Estos contienen un algoritmo que, en primer lugar, evitan que se dé a la salida un cero después de que *analyzer~* haya detectado correctamente un tono y su amplitud, y así reducir el error que este objeto genera. Como ya se ha mencionado anteriormente, *analyzer~* ofrece como salida el valor 0 si no detecta un tono, lo cual es frecuente cuando se usan estos proyectos debido al ruido del audio de entrada. Esto se ve en un objeto *multislider*, que representa en un monitor el tono que da la salida del analizador (Fig. 6.12.).

Fig. 6.12. *Multislider*

Seguidamente el valor de frecuencia se escala para entrar en el objeto *poly~* de creación de objetos como posición en el eje *y* de estos. Así, cierta altura en frecuencia determinará también una altura en el *mundo*. Lo mismo se hace con el valor de amplitud del tono detectado: se escala para que, en este caso, se transforme en el tamaño de los objetos tipo *plato* (apartado 4.3.) que representan dicha nota.

Añadido a *formasParcial1* se genera una segunda entrada vinculada a *cambioForma*, definido en el punto 6.2, al que precede del mismo modo *silencio*. La combinación de estos dos *subpatches* se encarga de seleccionar aleatoriamente una figura y aplicarla a los objetos cada vez que se producía un silencio largo, o después de un minuto, si esto no ocurre. *Silencio* recibe uno de los siguientes mensajes procedente de *cambioForma*: *tetra, hexa, octa, dodeca* o *icosa*, definiendo la forma de los objetos del primer parcial, igual que se hacía en *Attraction*.

En este caso, los objetos que representan la frecuencia y amplitud de cada parcial se definen dentro de un objeto *poly~*. El cuerpo en sí es un objeto *jit.gl.plato*, descrito en el apartado 5.2., cuyo color se establece de manera aleatoria al abrir el *patch* o cuando el usuario lo decida. La posición es aleatoria en los ejes *x* y *z*, el plano horizontal, mediante un sencillo algoritmo denominado

*smoothrandom* (Fig. 6.13.). La posición en el eje *z* está definida por la frecuencia, como se acaba de comentar, junto con la forma y la escala (Fig. 6.14.). Los objetos que toman su altura según la frecuencia del segundo parcial detectado no cambiarán de forma, a menos que el usuario la seleccione manualmente.

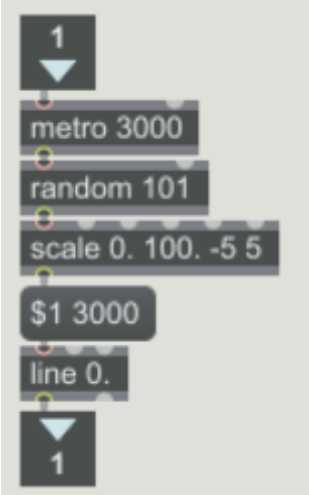



Fig. 6.13. *smoothrandom*

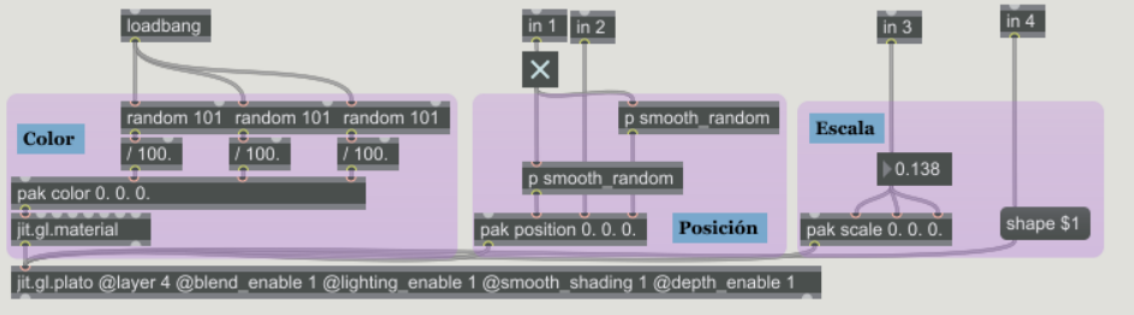



Fig. 6.14. *aleatorio.poly*

El aspecto más destacable de este diseño es la presencia de espejos (Fig. 6.15.), explicados en el apartado 5.2. Para recrearlos, se han construido cinco *patches* que representan las cinco paredes del entorno. Todas ellas han sido creadas de la misma manera, diferenciándose entre ellas únicamente por la posición y ángulos desde donde la cámara *graba* y la posición de su representación en el *mundo*.

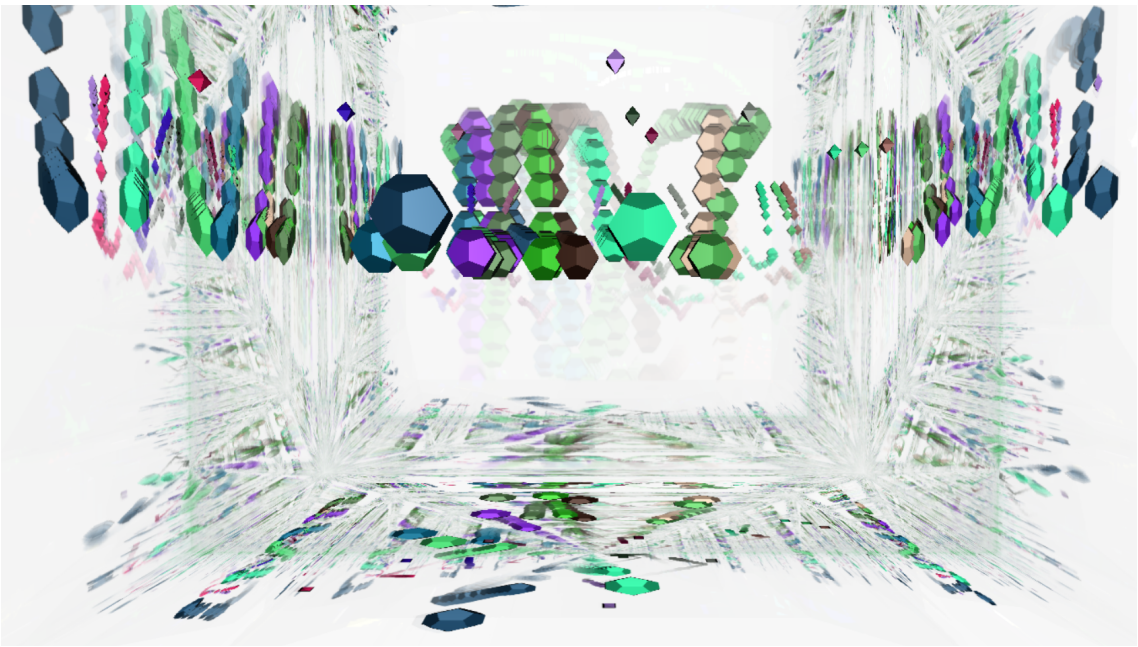


Fig. 6.15. *Mirrors*

# 7. CONCLUSIONES Y TRABAJO FUTURO

Con este trabajo se han creado tres proyectos en lenguaje Max que generan una representación gráfica y artística en vídeo en tiempo real a partir del análisis de un audio de entrada. Estos proyectos son prototipos diseñados para formar módulos independientes de Soundcool, aportándole nuevas funcionalidades, obteniendo no solo la posibilidad de emplear estos tres diseños, sino la oportunidad de hacerlo con todos aquellos que se pueden derivar de ellos.

En este sentido, el equipo de Soundcool podría crear nuevos módulos en los que participen variables diferentes a las propuestas en estos. En estos diseños se ha incluído la opción de que el usuario modifique algunos de los parámetros, pero las posibilidades son infinitas. A través de un panel de control, el usuario podría cambiar los colores, las formas, la transformación de ciertos parámetros auditivos en fuerzas que se aplican a los cuerpos, su tamaño, velocidad o posición, por ejemplo, todo lo cual sería muy sencillo de modificar.

La imagen que cada diseño aporta, habiéndose elegido frecuencia (o tono), amplitud (o volumen) y ataques (o ritmo) como parámetros auditivos, representa efectivamente lo que el usuario escucha, evocando una experiencia multisensorial. Así, se ha pensado que podría proponerse como trabajo futuro un desarrollo más exhaustivo del análisis de audio, para que personas con deficiencias sensoriales, como la sordera o la ceguera, pudieran oír la imagen o ver la música. Este trabajo iría en la línea de la rama de Soundcool dedicada a la diversidad funcional (Grupo Emosons). Para ello, se debería llevar a cabo un análisis frecuencial de alta complejidad con el fin de obtener las notas que forman cada acorde en la armonía del audio analizado en cada instante. Así, podría representarse más fielmente lo que cada acorde evoca, basándonos en los estudios psicológicos sobre la percepción de las diferentes armonías [26]. Del mismo modo, y con un desarrollo muy diferente, sería posible también crear material sonoro a partir de la imagen recibida, a fin de cumplir este mismo objetivo.

Durante el desarrollo del trabajo se ha intentado hacer este análisis frecuencial mediante Transformadas de Fourier (FFTs), sin embargo, se ha decidido usar en su lugar simplemente el objeto *analyzer~*, ya que está basado en este mismo algoritmo y ha sido creado por desarrolladores profesionales. No obstante, sería interesante intentar reproducir su funcionamiento para poder manejar desde dentro este objeto y conseguir los resultados propuestos previamente, pudiéndose analizar un audio más complejo y enriquecer la experiencia sensorial del usuario.

Por último, ha de hacerse mención a la sencillez con que Max permite crear este tipo de

proyectos. Visualizadores de audio como los de Windows Media Player o iTunes están basados en algoritmos muy complejos que proporcionan una visualización atractiva del sonido pero difícil de modificar. Mediante el desarrollo de este trabajo se ha comprobado que es posible construir herramientas similares, con elementos incluso más vistosos, modificables por el usuario a través de una interfaz y sin necesidad de ser un programador experto.

El presente Trabajo Fin de Grado forma parte del proyecto de la Fundación Daniel y Nina Carasso AC 16-AC-2016 (2016-2019). Aunque los proyectos desarrollados sean únicamente prototipos, los nuevos módulos se incluirán como parte del sistema, una vez modificadas ciertas funciones, como las opciones de cargar y guardar *patch* o las entradas y salidas de audio y vídeo, para hacerlos compatibles con el resto de módulos Soundcool.

# BIBLIOGRAFÍA


[1] Serrano Comes, J. E., *Nuevas Tecnologías e Interfaces para la Educación Musical: SoundCool* Trabajo de Fin de Carrera, Universitat Politècnica de València, 2013.

[2] V.J. Manzo, *Max/MSP/Jitter for Music: A Practical Guide to Developing Interactive Music Systems for Education and More*, Oxford University Press, 2014.

[3] *Soundcool OSC App*, PerformingARTech - UPV, URL = `https://play.google.com/store/apps/details?id=org.soundcool.upv.oscapp`, 2016, Último acceso: 11/10/2017

[4] *Soundcool, el innovador método colaborativo que busca otra enseñanza musical*, EFE-Valencia, eldiario.es, 2016.

[5] Lechner, P., *Multimedia Programming Using Max/MSP and TouchDesigner*, Packt Publishing, 2014.

[6] Puckette, M., *The Theory and Technique of Electronic Music*, World Scientific Publishing Co. Pte. Ltd., 2006.

[7] Peeters, G., *A large set of audio features for sound description (similarity and classification) in the CUIDADO project*, Ircam, 2004.

[8] Smith III, J. O., Abel, J. S., *Bark and ERB Bilinear Transforms*, IEEE Transactions on Speech and Audio Processing, 1999.

[9] `http://pages.mtu.edu/~suits/notefreqs.html`, *Frequencies for equal-tempered scale, A4 = 440 Hz*, Michigan Technological University. Último acceso: 10/2/2018

[10] Havelock, D., Kuwano S., Michael Vorländer, M., *Handbook of signal processing in acoustics*, Springer, 2008.

[11] *Perceptual Noise Reduction for Voice Quality Enhancement*, `https://www.vocal.com/noise-reduction/perceptual-noise-reduction/`, VOCAL Technologies, Ltd. Último acceso: 23/3/2018

[12] *Programming Max: Structuring Interactive Software for Digital Arts*, `https://www.kadenze.com/courses/programming-max-structuring-interactive-software-for-digital-arts-info`, Standford University. Último acceso: 1/10/2017.

[13] *Max tutorials*, https://cycling74.com/tutorials, Cycling74. Último acceso: 8/11/2017.

[14] *Max Online Documentation*, https://docs.cycling74.com/, Cycling74. Último acceso: 9/11/2017.

[15] *Cycling '74 Forums*, https://cycling74.com/forums, Cycling74. Último acceso: 15/10/2017.

[16] *Simple Introduction to the Jitter jit.gl.multiple Object*, https://www.clairesteger.com/home/simple-intro-to-jitglmultiple, Claire Steger. Último acceso: 30/10/2017.

[17] *MAX msp 7: Playing with jit.gl.gridshape, audio and textures!*, https://www.youtube.com/watch?v=Qlgrx_ZVjiw, Último acceso: 20/10/2017.

[18] *Max7 Tutorials*, https://www.youtube.com/channel/UCID7NkS7oLCT-iEqq15zL9w/videos, John Jannone. Último acceso: 3/2/2018.

[19] Charles, J. F., *A Tutorial on Spectral Sound Processing Using Max/MSP and Jitter*, Computer Music Journal, Massachusetts Institute of Technology, 2008.

[20] Dannenberg, R. B., *Interactive Visual Music: A Personal Perspective*, Computer Music Journal, Massachusetts Institute of Technology, 2005.

[21] Rowe, R., *Machine Musicianship*, MIT Press, 2001.

[22] Schichler, D., *A Vision of sound: A 3D visualization of pipe organ music*, Tesis, Rochester Institute of Technology, 2011.

[23] Hauer, A., *Physics-based music visualization*, Trabajo de Fin de Grado, Technische Universität Wien, 2016.

[24] Alfayate De la Iglesia, D., *Sinestesia: música y color*, Trabajo de Fin de Master, Universitat Politècnica de València, 2013.

[25] Gómez Gutiérrez, E., *Representación de señales de audio*, Departament de Sonologia, Escola Superior de Musica de Catalunya, 2009.

[26] Cytowic, R. E., *Synesthesia: A Union of the Senses* (2nd edition), MIT press, 2002.

[27] Yang, Y. et al., *Light up! Creating an Interactive Digital Artwork Based on Arduino and Max/MSP Design*, IEEE journal, 2010.

[28] Eidelman, E., Mokhov, S. A., *Alchymical Mirror: Real-time Interactive Sound- and Simple Motion-Tracking Set of Jitter/Max/MSP Patches*, Tesis, Concordia University, 2005.

[29] Pelejero Ibáñez, E. R., *Expansió del Sistema Soundcool per a donar suport a videocreació col-laborativa*, Trabajo de Fin de Grado, Universitat Politècnica de València, 2017.

[30] García Talavera, L., *Soundcool: Smatphones, Tablets y otras interfaces para la creación audiovisual.*, Trabajo de Fin de Grado, Universitat Politècnica de València, 2016.