Universidad Politécnica de Valencia, Facultad de Bellas Artes, Departamento de Pintura

Programa de Doctorado-Posgrado: Artes Visuales e Intermedia

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

TESIS DOCTORAL:

Generación de imagen sintética en tiempo real basada en muestras de sonido digitalizado en el campo del arte digital.

Presentada por:

Carlos García Miragall

Dirigida por:

Dra. María José Martínez de Pisón Ramón

Dr Francisco Javier Sanmartín Piquer

Valencia, enero 2016

Agradecimientos

En primer lugar agradecer las aportaciones de los evaluadores Diego Díaz García, María Blanca Montalvo Gallego y Eduardo Marin Sanchez y de los miembros del tribunal José Maldonado Gómez y Miguel Molina Alarcón.

Me gustaría también agradecer el apoyo recibido de todas mis compañeras y compañeros de mi grupo de investigación, el Laboratorio de Luz, desde que me acogieron me he sentido como en casa, arropado y cuidado. En especial a mis tutores María José y Paco, que han aguantado lo suyo.

También me gustaría agradecer el apoyo recibido de mis compañeros de batallas multisensoriales, el colectivo PDP11, Deco y David.

Dedico también este trabajo a mis padres Enrique y Mari, que a lo largo de los años me han recordado que había que hacer una tesis, ya está, también a mis hermanos Nuria y Quique, y a mi familia cercana.

También me gustaría agradecer el apoyo recibido a mis amigos, Nacho, Juan, Jacob, Iñaki, Javier, Raúl, Carli, Sara, Astor, Diego y a la pandilla de los Vivos.

Quiero también agradecer a Paloma el haberme aguantado durante tantos años el agobio de tener que hacer esta investigación.

En especial dedico este trabajo a mi amigo Gonzalo que desgraciadamente nos dejó hace muy poquito, un beso fuerte compadre.

Y sobre todo dedico este trabajo a mis dos lucecitas que me alegran la vida todos los días del año, mis hijos Hugo y Saúl.

“Como prolongados ecos que de lejos se confunden
en una tenebrosa y profunda unidad,
vasta como la noche y como la claridad,
los perfumes, los colores y los sonidos se responden.”

Charles Baudelaire

RESUMEN (castellano)

Generación de imagen sintética en tiempo real basada en muestras de sonido digitalizado en el campo del arte digital

Este trabajo de investigación se enmarca dentro del campo de la visualización del sonido, que en el contexto del arte explora la síntesis entre imagen y sonido, las representaciones físicas y gráficas del sonido y su evolución en el arte contemporáneo.

Cuando el sonido se estructura con una finalidad artística, como en la música, además de las propiedades físicas o psicoacústicas, como intensidad y frecuencia, percibimos cualidades estéticas y conceptuales. Una interpretación visual en este caso debería al menos recoger parte de esas cualidades físicas, estéticas y conceptuales y aportar nuevas ideas obtenidas de su fusión.

En este contexto, la investigación la hemos centrado en la generación en tiempo real de imagen a partir del sonido; proponiendo un método de visualización de sonido basado en las muestras numéricas, obtenidas como resultado de su digitalización. Este método está ideado para un entorno en el que la música y la generación de imágenes se desarrollan en tiempo real, y se perciben de forma conjunta, siendo adecuado para eventos audiovisuales en directo.

Si tenemos en cuenta que un sonido digitalizado no es más que un conjunto de números ordenados en el tiempo, denominados muestras y una imagen no es más que un conjunto de números ordenados en el espacio, denominados pixeles, el método que proponemos toma las muestras de sonido y las ubica espacialmente generando los pixeles de la imagen. Con el paso del tiempo se irá construyendo una secuencia de imágenes. En función de la estrategia de ubicación espacial de muestras, el número de muestras (presentes y pasadas) y

el número de fuentes sonoras a incorporar, obtenemos diferentes familias de algoritmos de visualización con características estéticas propias.

El objetivo que perseguimos con el método de visualización es encontrar interpretaciones visuales, generadas en tiempo real, que recojan parte de las propiedades físicas, estéticas y conceptuales del sonido, bajo la hipótesis que *la estructura interna de la señal sonora contiene propiedades perceptuales independientes del canal sensorial*. Para alcanzar este objetivo y corroborar la hipótesis, hemos desarrollado una aplicación informática para generar las imágenes a partir de las muestras del sonido y hemos analizado los resultados dentro del campo del arte digital.

El método de visualización propuesto se fundamenta en el hecho que desde el punto de vista informático el sonido y la imagen en movimiento son secuencias de números y simplemente lo que planteamos es construir las imágenes en función de los números del sonido. A través del diseño de unos casos de estudio, hemos experimentado con diferentes composiciones sonoras y con diferentes estrategias de ubicación espacial de las muestras de sonido. Del análisis efectuado hemos concluido que incluso con algoritmos de ubicación muy sencillos se aprecia la estrecha correlación que se produce entre el sonido y la imagen. Cuando se dividen las líneas de sonido en canales de color o en imágenes, se ven claramente los instrumentos. Cuanto más periódico es el sonido más estructurada y ordenada es la imagen resultante, incluso con algoritmos de ubicación que no mantienen estrictamente el orden secuencial de las muestras de sonido. En general el resultado estético lo podemos enmarcar en la corriente de arte abstracto, corroborando la naturaleza esencialmente abstracta del sonido.

Como futuros trabajos se plantea profundizar más en los algoritmos de ubicación espacial, dotándolos de inteligencia artificial, en la hibridación de dichos algoritmos, en la posibilidad de incorporar transformadas matemáticas basadas en el dominio de la frecuencia, y en la extensión de la investigación a un ámbito más general de visualización de datos.

Palabras-clave: AUDIOVISUALIZACIÓN, SONIDO ESTRUCTURADO, MÚSICA VISUAL, VISUALIZACIÓN DE DATOS, SOFTWARE ART, TIEMPO REAL, MUESTRAS DE SONIDO, SONIDO EXPERIMENTAL.

SUMMARY (English)

Real time synthetic image creation, based on digitised sound samples in the field of digital art

The present research is framed within the field of sound visualization, which within the context of fine art examines the synthesis between image and sound, the physical and graphical representations of sound and its evolution in contemporary art.

When sound is structured with an artistic purpose, as in music, in addition to physical or psycho-acoustic properties such as intensity and frequency, we perceive aesthetic and conceptual qualities. In this case, a visual interpretation should at least gather some of those physical, aesthetic and conceptual qualities, whilst the merger process should allow for innovative ideas.

In this context, the research focuses on real-time image production drawn from the sound; proposing a sound visualization method based on numerical samples obtained as a result of digitisation. This method is designed for an environment in which music and image production are developed in real time, and are perceived jointly, being suitable for audiovisual live events.

If we consider that a digitised sound is nothing but a set of numbers arranged in time, called samples, and an image is nothing more than a collection of numbers arranged in space, so-called pixeles, the method we propose takes the numerical values of the sound samples and then spatially locates them to create the pixels of an image. Over time it will generate a sequence of images.

Depending on the employed strategy of the spatial placement of the samples, the number of samples (past and present) and the number

of sound sources to be embedded, we obtain different families of visualization algorithms with their own aesthetic properties.

The objective of our display method is to uncover visual interpretations that developing in real time allow to collect part of the physical, aesthetic and conceptual properties of sound, under the hypothesis that the internal structure of the sound signal contains perceptual properties that are independent of the sensory channel.

In order to achieve this objective and seeking to corroborate the hypothesis, we have developed software to create images from sound samples and we have analyzed the results in the field of digital art.

The proposed display method is based on the fact that from the data-processing point of view sound and moving images are sequences of numbers and therefore we simply suggest creating the images in accordance with the numbers of the sound.

Through the design of case studies we have experimented with different sound compositions and with different strategies of spatial placement of the sound samples. After the analysis we have been able to conclude that even under the conditions of a very simple algorithms placement a close correlation between sound and image can be appreciated.

When the sound lines divide into colour channels or images, the instruments can be clearly seen. The more periodical is the sound the more structured and organized is the resulting image, even with placement algorithms that do not strictly maintain the sequential order of the sound samples.

In general, the aesthetic result can be framed within the context of abstract art, confirming the essentially abstract nature of sound.

Future work will consider the possibility of expanding the subject of algorithms of spatial location, providing them with artificial intelligence, as well as of the further study of hybridization of these algorithms, the possibility of incorporating transformed maths based on the frequency domain and the extension of the investigation into more general data visualization.

Key words: AUDIOVISUALIZATION, STRUCTURED SOUND, VISUAL MUSIC, DATA VISUALIZATION, SOFTWARE ART, REAL TIME, SOUND SAMPLES, EXPERIMENTAL SOUND.

RESUM (Valenciá)

Generació d'imatge sintètica en temps real basada en mostres de so digitalitzat en el camp de l'art digital

Aquest treball de recerca s'emmarca dins el camp de la visualització del so, que en el context de l'art explora la síntesi entre imatge i so, les representacions físiques i gràfiques del so i la seua evolució en l'art contemporani.

Quan el so s'estructura amb una finalitat artística, com en la música, a més de les propietats físiques o psicoacústiques, com intensitat i freqüència, percebem qualitats estètiques i conceptuals. Una interpretació visual en aquest cas hauria almenys de recollir part d'aquestes qualitats físiques, estètiques i conceptuals i aportar noves idees obtingudes de la seua fusió.

En aquest context, la investigació l'hem centrat en la generació en temps real d'imatge a partir del so; proposant un mètode de visualització de so basat en les mostres numèriques, obtingudes com a resultat de la seua digitalització. Aquest mètode està ideat per a un entorn en què la música i la generació d'imatges es desenvolupen en temps real, i es perceben de forma conjunta, sent adequat per a esdeveniments audiovisuals en directe.

Si tenim en compte que un so digitalitzat no és més que un conjunt de nombres ordenats en el temps, denominats mostres i una imatge no és més que un conjunt de nombres ordenats en l'espai, anomenats píxels, el mètode que proposem pren les mostres de so i les situa espacialment generant els píxels de la imatge. Amb el pas del temps s'anirà construint una seqüència d'imatges. En funció de l'estratègia d'ubicació espacial de mostres, el nombre de mostres (presents i passades) i el nombre de fonts sonores a incorporar, obtenim

diferents famílies d'algorismes de visualització amb característiques estètiques pròpies.

L'objectiu que perseguim amb el mètode de visualització és trobar interpretacions visuals, generades en temps real, que recullin part de les propietats físiques, estètiques i conceptuals del so, sota la hipòtesi que l'estructura interna del senyal sonor conté propietats perceptuals independents del canal sensorial . Per assolir aquest objectiu i corroborar la hipòtesi, hem desenvolupat una aplicació informàtica per a generar les imatges a partir de les mostres del so i hem analitzat els resultats dins el camp de l'art digital.

El mètode de visualització proposat es fonamenta en el fet que des del punt de vista informàtic el so i la imatge en moviment són seqüències de nombres i simplement el que plantegem és construir les imatges en funció dels números del so. A través del disseny d'uns casos d'estudi, hem experimentat amb diferents composicions sonores i amb diferents estratègies d'ubicació espacial de les mostres de so. De l'anàlisi efectuada hem conclòs que fins i tot amb algoritmes d'ubicació molt senzills s'aprecia l'estreta correlació que es produeix entre el so i la imatge. Quan es divideixen les línies de so en canals de color o en imatges, es veuen clarament els instruments a través de les imatges. Com més periòdic és el so més estructurada i ordenada és la imatge resultant, fins i tot amb algoritmes d'ubicació que mantenen estrictament l'ordre seqüencial de les mostres de so. En general el resultat estètic el podem emmarcar en el corrent d'art abstracte, corroborant la naturalesa essencialment abstracta del so.

Com a futurs treballs es planteja aprofundir més en els algoritmes d'ubicació espacial, dotant-los d'intel·ligència artificial, en la hibridació d'aquests algoritmes, en la possibilitat d'incorporar transformades

matemàtiques basades en el domini de la freqüència, i en l'extensió de la investigació a un àmbit més general de visualització de dades.

Paraules-clau: AUDIOVISUALITZACIÓ, SO ESTRUCTURAT, MÚSICA VISUAL, VISUALITZACIÓ DE DADES, PROGRAMARI ART, TEMPS REAL, MOSTRES DE SO, SO EXPERIMENTAL.

CONTENIDO

1 INTRODUCCIÓN.......... 29
1.1 Presentación del objeto de estudio 29
1.2 Motivaciones.......... 31
1.3 Hipótesis y objetivos.......... 33
1.4 Metodología.......... 35
1.5 Estructura de contenidos.......... 36
2 VISUALIZACIÓN DEL SONIDO EN EL CAMPO DEL ARTE.... 39
2.1 Instrumentos visuales.......... 43
2.2 Medios audiovisuales 54
2.2.1 Música Visual 54
2.2.2 VideoArte.......... 73
2.2.3 VideoClip 85
2.3 Sistemas digitales.......... 87
2.3.1 Lenguajes de Programación y Aplicaciones 89
2.3.2 Interactividad y tiempo real 97
2.4 Otros aspectos artísticos y tecnológicos de interés..... 106
3 CONCEPTOS TÉCNICOS DE SONIDO.......... 110
3.1 Digitalización del sonido.......... 111
3.1.1 Muestreo.......... 113
3.1.2 Cuantificación.......... 115
3.1.3 Codificación.......... 116
3.1.4 La señal digital de sonido.......... 117

3.2 Comunicación entre la aplicación y la tarjeta de sonido118

4 UN MÉTODO DE VISUALIZACIÓN DEL SONIDO BASADO EN LAS MUESTRAS OBTENIDAS DE SU DIGITALIZACIÓN. 121

4.1 De números a números 122

4.2 Un método de visualización de sonido basado en muestras 132

4.2.1 De intensidad sonora a intensidad luminosa 133

4.2.2 Correlación temporal 136

4.2.3 Todo cambia 137

4.2.4 Ejemplo 138

4.2.5 Formalización del método de visualización 140

4.3 Algoritmos de ubicación espacial de muestras 142

4.3.1 Estrategias simétricas básicas 145

4.3.2 Estrategias simétricas en espiral. 162

4.3.3 Otras estrategias simétricas. 163

4.3.4 Estrategias basadas en reglas de comportamiento. 168

4.3.5 Generalización de los algoritmos de ubicación cuando tenemos más de una línea de sonido. 176

4.4 El lenguaje del cambio. 178

5 CICLOPE: UNA APLICACIÓN PARA LA VISUALIZACIÓN DE SONIDO EN UN ENTORNO DE TIEMPO REAL 181

5.1 Análisis de la aplicación 186

5.1.1 Casos de uso 188

5.1.2 Requisitos funcionales y no funcionales 190

5.2 Diseño de la aplicación.. 198

5.2.1 Módulo de control 203

5.2.2 Interfaz de usuario.. 209

5.2.3 Módulo de sonido 234

5.2.4 Módulo de Imagen.. 239

5.2.5 Gestión de errores.. 244

5.2.6 Otras consideraciones de diseño 245

5.3 Implementación de la aplicación 246

5.3.1 Acceso en exclusión mutua..................................... 248

5.3.2 Eficiencia de la aplicación 249

6 CASOS PRACTICOS. ANÁLISIS DE ALGORITMOS DE UBICACIÓN DE MUESTRAS. 253

6.1 Descripción técnica del diseño de sonido 254

6.2 Descripción técnica de los parámetros de visualización258

6.3 Descripción del equipo de trabajo. El colectivo PDP11.259

6.4 Descripción del material técnico usado para las pruebas260

6.5 Esquema general de cada uno de los casos prácticos 261

6.6 Casos Prácticos.. 262

6.6.1 Caso 1. Señal periódica 262

6.6.2 Caso 2. Guitarra eléctrica.. 274

6.6.3 Caso 3. Sintetizador 284

6.6.4 Caso 4. Guitarra eléctrica y sintetizador 289

6.6.5 Caso 5. Improvisación PDP11. 293

6.6.6 Caso 6. Anteproyecto *Lux Interior* PDP11 297

6.7 Videos en Vimeo 304

7 CONCLUSIONES 305

8 BIBLIOGRAFÍA 311

Anexo1. Documentación de Ciclope. 323

A.1.1. Librería de sonido 325

A.1.2. Librería de interfaz de usuario 346

A.1.3. Librería de imagen 372

A.1.4. Librería de controlAplicacion 411

LISTA DE FIGURA

Figura 2.1.1. Caricatura del clavicordio ocular de Louis Bertrand Castel 44

Figura 2.1.2 Órgano de color de Bishop 45

Figura 2.1.3 Órgano de color de Alexander Wallace Rimington 46

Figura 2.1.4. Piano Optofónico y discos transparentes. 47

Figura 2.1.5. Clavilux de Thomas Wilfred. 48

Figura 2.1.6. Proyector de color de Adrian Klein. 49

Figura 2.1.7. Sonchromatoscope de Alexander László. 50

Figura 2.1.8. Órgano de color de Hallock-Greenewalt. 51

Figura 2.1.9. Tabla comparativa entre colores y sonidos. 52

Figura 2.1.10. Estudio para la película *Rythme Coloré* de Survage. . 56

Figura 2.1.11.Primera acuarela abstracta de Kandinsky. 58

Figura 2.1.12. *Diagonale, Sonido Apacible* y Contrasting Sound. 60

Figura 2.1.13. *Nocturno* de Kupka, 1911. 61

Figura 2.1.14. Fotograma de *Lichtspiel Opus I,* Ruttman, 1921. 63

Figura 2.1.15. Fotograma de *Rhythmus 21*, Ritcher, 1921. 64

Figura 2.1.16. Fotograma de *Synphonie Diagonale*, Eggeling, 1924. 65

Figura 2.1.17. Experimentos de *Ornament Sound*. Fischinger. 67

Figura 2.1.18. Fotograma de *Color Box*, Lye, 1935. 68

Figura 2.1.19. Fotograma de *Loops*, Norman McLaren, 1940. 69

Figura 2.1.20. Fotograma de *Early Abstractions*, Smith, 1946-58. 70

Figura 2.1.21. Oscilón nº4, Laposky, 1950. 71

Figura 2.1.22. Fotograma de *Abstronic*, Bute (1954). 72

Figura 2.1.23. Imagen publicitaria de la PortaPack de Sony. 75

Figura 2.1.24. Paik-Abe Video Synthesizer. 77

Figura 2.1.25. *Direct Video Zero.* Stephen Beck................................ 78

Figura 2.1.26. Sintetizador. *Image Processor,* Dan Sandin. 79

Figura 2.1.27. *Atari Video Music* y fotograma de una visualización. . 80

Figura 2.1.28. Fotograma de *NoiseFiels*, Vasulka (1974). 81

Figura 2.1.29. Fotograma de *Electronic Linguistic*, Hill (1977). 82

Figura 2.1.30. Fotograma de *Bits*, Hill (1977). 83

Figura 2.1.31. Fotograma de *Fourth Dimension*, Rybczynski (1988). 84

Figura 2.1.32. Cartel de “Corre Lola corre”, Tykwer (1998). 86

Figura 2.1.32a. Ranking de lenguajes en 2015. 91

Figura 2.1.33. Un patch de Pure Data. 93

Figura 2.1.34. El entorno de desarrollo de Gamuza. 94

Figura 2.1.35. El entorno de desarrollo de Modul8. 96

Figura 2.1.36. Fotograma de *Trioon I*, 2003. 98

Figura 2.1.37. Fotograma de *Rotationsstudien Sequenzen 20-5*, 2015. 99

Figura 2.1.38. Performance audiovisual *Test Pattern*, Ikeda, 2008. 100

Figura 2.1.39. Instalación *Test Pattern* en Time Square, Ikeda, 2013. 101

Figura 2.1.40. Instalación Modulador de Luz, laboluz, 2008. 102

Figura 2.1.41. Performance Root 1.1, PDP11, 2011. 103

Figura 2.1.42. Performance Deep Web, Monolake, 2015. 104

Figura 2.1.43. Performance *Messa di Voce*, Levin (2003). 105

Figura 3.2.1. Esquema de conversión A/D y D/A 113

Figura 3.2.2. Conversor A/D 113

Figura 3.2.3. Señal analógica y señal digital 114

Figura 4.1.1. Detalle de una forma de onda de sonido. 126

Figura 4.1.2. Imagen en la que se ven los pixeles. 127

Figura 4.1.3. Relaciones espaciales entre *pixeles*. 128

Figura 4.1.4. Relaciones temporales entre muestras. 129

Figura 4.1.5. Redistribución de muestras en imágenes. 130

Figura 4.2.1. Imagen generada a partir de un sonido sinusoidal. 140

Figura 4.3.1 Esquema del algoritmo de ubicación $idab$. 146

Figura 4.3.2 Imagen resultado del algoritmo de ubicación $idab$. 147

Figura 4.3.3 Esquema del algoritmo de ubicación $idba$. 148

Figura 4.3.4 Esquema del algoritmo de ubicación $diab$. 149

Figura 4.3.5 Esquema del algoritmo de ubicación $diba$. 150

Figura 4.3.6 Esquema del algoritmo de ubicación $abid$ 151

Figura 4.3.7 Esquema del algoritmo de ubicación $abdi$ 152

Figura 4.3.8 Esquema del algoritmo de ubicación $baid$ 153

Figura 4.3.9 Esquema del algoritmo de ubicación $badi$ 154

Figura 4.3.10 Esquema del algoritmo de ubicación $idzab$ 155

Figura 4.3.11 Esquema del algoritmo de ubicación $idzba$ 156

Figura 4.3.12 Esquema del algoritmo de ubicación $dizab$ 157

Figura 4.3.13 Esquema del algoritmo de ubicación $dizba$ 158

Figura 4.3.14. Esquema del algoritmo de ubicación $abzid$ 159

Figura 4.3.15. Esquema del algoritmo de ubicación $abzdi$ 160

Figura 4.3.16. Esquema del algoritmo de ubicación $bazid$ 161

Figura 4.3.17. Esquema del algoritmo de ubicación $bazdi$ 162

Figura 4.3.18. Estructura de una imagen por niveles. 163

Figura 4.3.19. Esquema del algoritmo de ubicación en espiral 163

Figura 4.3. 20 Esquema combinado $iadab$ y $diba$ 165

Figura 4.3.21. Esquema combinado $iadab$ y $diba$ 165

Figura 4.3.22. Esquema combinado $iadab$ y $diba$ 166

Figura 4.3.23. Estados consecutivos del juego de la vida 173

Figura 4.3.24. Transformación aritmética 178

Figura 5.1 Logotipo de la aplicación Cíclope. 182

Figura 5.2. Interfaz gráfica del prototipo inicial de Cíclope 183

Figura 5.3. Fases en el desarrollo de la aplicación con realimentación. 185

Figura 5.1.1. Plantilla para cartel de conciertos del grupo pdp11.... 187

Figura 5.1.2. Esquema de entradas y salidas de la aplicación 188

Figura 5.1.3. Diagrama de casos de uso de la aplicación 189

Figura 5.2.1. División lógica de la aplicación 201

Figura 5.2.2. Módulos principales de la aplicación 202

Figura 5.2.3 Ciclo de vida de la aplicación...................................... 204

Figura 5.2.4. Esquema de clases del módulo de control 208

Figura 5.2.5. Esquema de lienzo final con varias imágenes............ 215

Figura 5.2.6. Ejemplo de nodos de secuenciación. 218

Figura 5.2.7. Ejemplo completo de nodos de secuenciación. 218

Figura 5.2.8 Marco principal de la interfaz gráfica. 223

Figura 5.2.9. Reproductor de sonido Aiwa XR-EM400. 223

Figura 5.2.10. Modelo de botón clásico. 224

Figura 5.2.11. Primeras versiones de los botones. 225

Figura 5.2.12. Logotipo del lenguaje de programación Java. 225

Figura 5.2.13. Bocetos iniciales del logotipo de la aplicación. 226

Figura 5.2.14. Logotipo de la aplicación. 226

Figura 5.2.15. Botones de reproducción. 227

Figura 5.2.16. Imagen de botón pulsado. 228

Figura 5.2.17. Botones de aceptación y configuración. 228

Figura 5.2.18. Tipo de letra Seagoe UI 228

Figura 5.2.19. Marco con todos los elementos desordenados de la IGU. 229

Figura 5.2.20. Pantalla inicial de bienvenida. 231

Figura 5.2.21. Cambio de estado de INICIAL a INCIADO. 231

Figura 5.2.22. Cambio de estado de INICIADO a CONFIGURADO. 232

Figura 5.2.23. Pantalla para controlar la visualización..................... 232

Figura 5.2.24. Lienzo de visualización. 233

Figura 5.3.1. Interfaz gráfica principal de Eclipse. 246

Figura 5.3.2 Página principal de la documentación de Ciclope. 247

Figura 6.6.1 *Path* de Pure Data sinusoidal y ruido. 263

Figura 6.6.2. Patrón característico cada 700Hz. 266
Figura 6.6.3. Simetrías estática en 1174 Hz. 267
Figura 6.6.4. Patrón característico cada 700Hz. 268
Figura 6.6.5. Visualización de ruido. 269
Figura 6.6.6. Visualización de ruido modulado. 270
Figura 6.6.7. Sinusoidal y armónico. 271
Figura 6.6.7. Fotogramas sinusoidal y armónico. 272
Figura 6.6.8. Visualización sinusoidal más ruido blanco con color. . 273
Figura 6.6.8a. Visualización sinusoidal con hormiga de Langton. ... 274
Figura 6.6.9. Guitarra Fender Jazzmaster. 275
Figura 6.6.9. Fotogramas guitarra con algoritmo $idab$, escalado duro. 278
Figura 6.6.10. Fotogramas guitarra con algoritmo $idab$, escalado suave. 279
Figura 6.6.11. Fotogramas guitarra con algoritmo $abid$, escalado duro. 281
Figura 6.6.12. Fotogramas guitarra con algoritmo combinado. 281
Figura 6.6.13. Aparición de ruido. 282
Figura 6.6.14. Fotogramas guitarra formando ángulos. 283
Figura 6.6.15. Formas rectangulares estilizadas. 284
Figura 6.6.16. Visualización en forma de cruz. 286
Figura 6.6. 17. Secuenciación de fotogramas caso 3a. 288
Figura 6.6.18. Secuenciación de fotogramas caso 3b. 289
Figura 6.6.19. Secuenciación de fotogramas casos 4. 292

Figura 6.6.19a. Deco Nascimento con parte de su equipo. 293

Figura 6.6.20. Visualización con 4 zonas. 295

Figura 6.6.21. Textura guía de inprovisación. 296

Figura 6.6.22. *Lux Interior The Cramps*. 298

Figura 6.6.23. Bajo acústico Whashburn AB-10. 299

Figura 6.6.24. Visualización con 4 zonas en forma de cruz. 301

Figura 6.6.25. Fotogramas *Lux Interior*. 303

Figura A1.1. Página principal de la documentación *de Java.* 324

1 INTRODUCCIÓN

En este apartado vamos a realizar una introducción al proyecto de investigación. Inicialmente presentaremos el marco general donde definimos la investigación, concretando el objeto de estudio. A continuación exponemos las motivaciones personales que nos han conducido a trabajar en este proyecto. Seguidamente presentaremos la pregunta principal que cuestiona el trabajo, la hipótesis, y los objetivos acometidos para abordarla. Posteriormente comentaremos la metodología de investigación empleada y finalmente resumiremos los contenidos de esta memoria que recoge el proceso de investigación realizado.

1.1 Presentación del objeto de estudio

Este trabajo de investigación se enmarca dentro del campo de la visualización del sonido, que en el contexto del arte explora la síntesis entre imagen y sonido, las representaciones físicas y gráficas del sonido y su evolución en el arte contemporáneo. En este ámbito la investigación la centraremos en la síntesis de imagen a partir del sonido en un contexto de tiempo real.

Cuando el sonido se estructura con una finalidad artística, como en la música, además de las propiedades físicas o psicoacústicas, como intensidad y frecuencia, percibimos cualidades estéticas y conceptuales. Una interpretación visual en este caso debería al menos recoger parte de esas cualidades físicas, estéticas y conceptuales y aportar nuevas ideas obtenidas de su fusión.

A lo largo de la historia se han realizado muchas aportaciones en la búsqueda de relaciones entre el sonido y la imagen. Para centrar el objeto de nuestro estudio podemos establecer dos grandes aproximaciones de visualización del sonido. Por un lado las que

denominamos metafóricas puras, donde la interpretación visual del sonido se corresponde con una visión particular del artista, estableciendo principalmente relaciones subjetivas entre lo visual y lo sonoro. Un buen representante de esta aproximación es prácticamente todo el género del videoclip, donde si eliminamos su aplicación comercial, lo que tenemos es una interpretación, habitualmente figurativa de la música. La correlación entre el sonido y la imagen se realiza porque vemos al grupo musical tocar, los cambios de planos están sincronizados con el ritmo de la canción, se presentan bailes sincronizados con la música,… Por otro lado las que denominamos naturales, donde la interpretación visual está directamente relacionada con propiedades del sonido. En un caso extremo tendríamos las visualizaciones técnicas del sonido mediante forma de onda o espectrogramas, y en el otro toda la tradición de lo que se ha denominado música visual, donde se busca una interpretación artística centrada en las relaciones directas entre sonido e imagen. Además del ritmo, se establecen conexiones entre los tonos musicales, la intensidad, melodía, armonía,… El resultado estético es como la música, abstracto.

En este contexto, nuestra propuesta se centra en la aproximación natural, donde la interpretación visual se generará en tiempo real a partir del sonido y de forma automática. El método que proponemos está basado en las muestras numéricas del sonido, obtenidas como resultado de su digitalización. Este método está ideado para un entorno en el que la música y la generación de imágenes se desarrollen en tiempo real, y se perciban de forma conjunta, siendo adecuado para eventos audiovisuales en directo.

Si tenemos en cuenta que un sonido digitalizado no es más que un conjunto de números ordenados en el tiempo que representan la intensidad de la señal en cada instante, y una imagen en un instante

determinado, no es más que un conjunto de pixeles, cuyo valor es un número que representa la intensidad de luz en un punto. El método que proponemos toma los valores numéricos de las muestras de sonido y los ubica espacialmente formando pixeles de una imagen. Con el paso del tiempo se irán generando imágenes que formarán una secuencia. En función de la estrategia de ubicación espacial de muestras, el número de muestras (presentes y pasadas) y el número de fuentes sonoras a incorporar, finalmente obtenemos diferentes familias de algoritmos de visualización con características estéticas propias.

1.2 Motivaciones

La idea de conectar sonido e imagen a través del universo informático y con una visión artística, no me vino de forma repentina sino que ha sido un proceso paulatino e íntimamente relacionado con la evolución de mis intereses profesionales y personales.

En el inicio fue la música lo que más me cautivó, empecé a tocar la guitarra interpretando las canciones que estaban de moda, pero esa visión de intérprete, ya desde el principio no me convenció, y empecé a componer mis propias canciones. Me atraía la idea de dejarme llevar, los dedos fluían por el mástil de la guitarra, por su cuerpo y el sonido devenía. Siempre me ha gustado ese aspecto efímero del sonido, aparece e inmediatamente se vuelve a perder. Si queremos que perdure, hay que preparar su registro, ya que su naturaleza es fugaz. Es el medio más estrechamente relacionado con el tiempo, puesto que se rige por sus mismas reglas.

La imagen con menor intensidad, también ha estado presente en mi vida. Al principio fue la pintura y el dibujo. Como con el sonido siempre me ha interesado más el lado de la composición que el de la

interpretación, en este caso de la realidad. Es posible que una cosa lleve a la otra y como la estética natural del sonido es la abstracción, desde pequeño lo que más me ha interesado han sido justamente conceptos visuales simples: líneas, puntos, colores,… Mi interés por la fotografía como con la pintura ha sido discontinúo a lo largo de los años. Más que la recreación del mundo social, el interés fotográfico ha recaído siempre sobre detalles, texturas y sutilezas del entorno.

Por otro lado, a nivel profesional me he dedicado a la Informática en el ámbito docente. Mi visión de la informática siempre ha estado del lado del software, siendo en el campo de la programación donde he dedicado más esfuerzos docentes. De la programación lo que más me gusta es la sencillez con la que se construye algo complejo. La base es materialmente muy pequeña, los ladrillos de la programación son simples líneas de texto. Se trata de un lenguaje muy reducido, que sin embargo recoge la mayoría de estrategias lógicas que los seres humanos usamos, porque la complejidad siempre se construye desde la sencillez.

Cuando a principios de siglo comienza a popularizarse el tratamiento digital de sonido e imagen, empiezo a introducirme en el campo multimedia, primero a nivel personal y luego laboral. Primero a través de la imagen, con el tratamiento digital de video y los efectos visuales, y luego con el sonido. Este realmente fue todo un acontecimiento para mi desarrollo profesional, ya que se brindaba la oportunidad de trabajar con el sonido, con la imagen y todo dentro del ámbito de la informática. Es en esta época cuando empiezo a pensar en relacionar el sonido con la imagen dentro del universo informático. Podemos decir que es cuando se siembra la semilla de este trabajo de investigación. Pero fue a partir de que entro en contacto con mi actual grupo de investigación, el Laboratorio de Luz, cuando realmente

defino el ámbito de la investigación y la enmarco dentro del campo de las artes visuales.

A nivel personal este trabajo supone la fusión de tres facetas de mi vida que en los inicios estuvieron separadas y ahora los he abordado de forma conjunta: el sonido, la imagen y la programación.

1.3 Hipótesis y objetivos

Con este trabajo de investigación proponemos un método de visualización del sonido basado en las muestras obtenidas de su digitalización. La idea que subyace en la propuesta consiste en una reinterpretación del lenguaje numérico. El sonido al digitalizarse se aproxima mediante una secuencia de números ordenados temporalmente. Estos números contienen información sobre las variaciones de intensidad de la onda sonora producida en el aire o generada sintéticamente[1]. Cuando queremos volver a escuchar ese sonido, tendremos que realizar el proceso inverso y la señal digitalizada deberá reconvertirse en analógica. Lo que proponemos es presentar la secuencia numérica original de forma visual. Con esto reinterpretamos los números que pasarán a denotar intensidad visual en vez de sonora.

Lo que queremos comprobar es si al reinterpretar la señal sonora, a través de la secuencia numérica que la identifica, mediante una secuencia de imágenes que contienen exactamente los mismos números, se mantienen las propiedades perceptuales del sonido. De manera que podamos concluir que *la estructura interna de la señal sonora contiene propiedades perceptuales independientes del canal sensorial*.

[1] La señal sonora no necesariamente se produce de forma analógica, sino también se puede producir de forma digital mediante esquemas de síntesis de sonido.

Para comprobar esta hipótesis se pretende:

- Definir de forma precisa las características del método de visualización, teniendo en cuenta las propiedades del sonido y de la imagen, y estableciendo la máxima correlación entre ellas.
- Desarrollar un entorno informático adecuado y flexible para poner en práctica el método de visualización propuesto. El entorno informático debe tener en cuenta que el método propuesto es un marco de trabajo que establece la regla principal de visualización: reinterpretar las muestras de sonido de forma visual. Las diferentes estrategias de ubicación espacial de las muestras en la imagen constituyen en si las interpretaciones visuales. Esas estrategias pueden ser tan sencillas como ubicarlas de forma ordenada de arriba abajo o de izquierda a derecha hasta usar otras que contengan cierta inteligencia artificial, como un autómata celular.
- Proponer interpretaciones visuales adecuadas para diferentes propuestas sonoras, estableciendo de esta forma un variado conjunto de casos prácticos.
- Analizar estos procesos dentro del campo del arte digital.

El desarrollo de estos objetivos nos conducirá a tener una visión completa sobre la idoneidad del método propuesto, y comprobar si realmente en la reinterpretación de la secuencia de números se siguen manteniendo algunas de las propiedades perceptuales del sonido original.

1.4 Metodología

El marco metodológico que proponemos para el desarrollo de la investigación recoge tanto métodos del paradigma cualitativo como del cuantitativo, siguiendo un modelo híbrido característico del arte digital; en el que se combinan metodologías de investigación de ingeniería y arte.

Como punto de partida se ha realizado una investigación bibliográfica del campo de la visualización del sonido para conocer el estado actual de la cuestión. Además con esta revisión pretendemos: buscar, recopilar, organizar, valorar y criticar información bibliográfica sobre el sonido y sus propiedades, el tratamiento digital del sonido y la imagen, las relaciones históricas y actuales entre imagen y sonido en el contexto del arte digital. Para este proceso diseñaremos una base de datos con información de referencia de cada una de las fuentes bibliográficas consultadas. Aunque esta base de datos no la presentemos en la memoria explícitamente es uno de los pilares sobre el que se va a sentar el desarrollo de la investigación.

Con la finalidad de delimitar las composiciones sonoras que vamos a visualizar nos apoyaremos en los proyectos sonoros del colectivo PDP11 del que formamos parte activa, realizando una selección de trabajos e incluso planteando nuevas composiciones específicas para la investigación. Esta estrecha cooperación nos permitirá estudiar elementos sonoros de forma independiente y conjunta.

Para el desarrollo de un entorno informático de visualización proponemos una metodología clásica del campo de la ingeniería del software. Inicialmente analizaremos el problema, posteriormente diseñaremos la solución y por último realizaremos la implementación de la herramienta. En la parte de diseño estudiaremos y analizaremos

diferentes lenguajes de programación y librerías en base a su eficiencia y eficacia para analizar sonido y visualizar imagen en tiempo real.

Una vez desarrollado el entorno, centraremos la investigación casos de estudio. Para cada caso, inicialmente seleccionaremos la pieza sonora, en base a sus cualidades sonoras, determinando aspectos del sonido tales como periodicidad, complejidad, armonía, frecuencia, intensidad,... Luego se experimenta con diferentes algoritmos de visualización, desde estrategias geométricas hasta modelos mucho más complejos. Por último se evalúan parámetros como: respuestas visuales, idoneidad de la visualización o eficacia del sistema al evento sonoro visual, con la finalidad de establecer conclusiones generales.

1.5 Estructura de contenidos

Esta memoria la hemos dividido en ocho capítulos más un anexo.

En el segundo capítulo presentamos el marco referencial. Debido a que no es el objetivo de esta tesis realizar un estudio histórico exhaustivo de las aportaciones artísticas en el campo de visualización del sonido, hemos incluido solo aquellas propuestas que más directamente han influido en el desarrollo de la investigación.

En el tercer capítulo revisamos los conceptos teóricos en los que se fundamenta el método que proponemos. Aunque consideramos que se trata de conocimientos básicos en el campo de teoría de la señal digital, hemos querido incluirlos debido al carácter interdisciplinar de este proyecto. El objetivo de este apartado es introducir la notación formal de la señal digital sonora que usaremos para formalizar las características del método propuesto.

En el cuarto capítulo presentaremos el método que proponemos desde un marco formal y detallaremos un conjunto de estrategias de ubicación espacial de las muestras. El objetivo de este capítulo es presentar el método sin entrar en valoraciones de la efectividad visual o idoneidad de las estrategias.

En el quinto capítulo mostramos los detalles del desarrollo de la aplicación realizada para la prueba del método. La estructura del capítulo obedece a las diferentes fases de desarrollo de software. Primero realizamos el análisis, seguidamente presentamos el diseño de la aplicación, para terminar con las ideas generales de implementación y prueba de la aplicación.

En el sexto capítulo exponemos los diferentes casos prácticos que hemos realizado con la finalidad de probar las estrategias de visualización en varios contextos sonoros. En este apartado es donde abordaremos el análisis del método.

El séptimo capítulo está dedicado a presentar las conclusiones de la investigación y esbozar futuras líneas de investigación.

Por último en el capítulo ocho presentamos las referencias bibliográficas consultadas en el desarrollo de la investigación.

Adicionalmente hemos incluido un anexo, con la documentación detallada de la implementación de la aplicación.

“Al mismo tiempo. Dibujaba un pie o dos de imágenes y luego, inmediatamente después, dibujaba el sonido. Pero no tenía el equipo para oírlo. Me fascinaba el constatar que era capaz de dibujar el sonido sin poder escucharlo.”

Norman MacLaren[2]

2 VISUALIZACIÓN DEL SONIDO EN EL CAMPO DEL ARTE.

La idea de obtener imágenes a partir de la música ha sido tratada a lo largo de los siglos; no como un movimiento independiente sino siempre dentro del contexto cultural y tecnológico de cada época. Este esfuerzo histórico por buscar relaciones entre la música, el sonido y la imagen se puede agrupar en lo que se ha venido a denominar Música Visual, Visualización del Sonido o Sonido Visual[3] que fundamenta su naturaleza en la noción de sinestesia (Brougher 2005).

Con independencia de la disciplina en la que nos encontremos la sinestesia en su sentido más amplio, se puede entender como una mixtura de impresiones que se perciben mediante distintos sentidos. De ahí que una persona sinestésica sea capaz de ver el sonido o escuchar un color. En el contexto de visualización del sonido, lo que se persigue es ver el sonido. Se trata de realizar una interpretación visual del sonido pero tomando como base principal el propio sonido,

[2] En (Backedano, 1987,48).
[3] Realmente nunca ha habido un término que identifique unívocamente al campo.

de la misma forma que lo haría una persona con capacidades sinestésicas. Las relaciones entre el sonido y la imagen o la luz a lo largo de la historia siempre han estado fusionadas con la idea de sinestesia.

En esa búsqueda de interpretar visualmente el sonido se han realizado multitud de propuestas, estando marcadas fundamentalmente por los medios tecnológicos involucrados en la creación de las imágenes. A grandes rasgos podemos trazar dos etapas históricas que hacia finales del siglo XX convergerán en los sistemas digitales. Por un lado la tradición de órganos de color o instrumentos visuales, que se basa en realizar extensiones o modificaciones de instrumentos musicales con sistemas de proyección acoplados, capaces de establecer relaciones directas y en tiempo real entre el sonido y la imagen -el intérprete conforme toca la música va lanzando y generando las imágenes. Y por otro la tradición que surge alrededor del cine y el video analógico, estas tecnologías, con sus peculiaridades, permiten almacenar sonido e imagen sincronizada y procesar sus elementos.

Con la llegada del ordenador se produce la convergencia a un único dispositivo capaz de diseñar instrumentos de color, de fusionar el cine y el video en un único medio, de generar imagen sintética y sobre todo capaz de traducir la imagen y el sonido en un mismo lenguaje, el lenguaje de los números. En este contexto es cuando las posibilidades creativas de conexión entre el sonido y la imagen se manifiestan en su plenitud -sintetizadas en un lenguaje común.

Actualmente es un campo muy activo debido a que se están realizando convergencias tanto del campo del sonido hacia la visualización, como del campo de las artes visuales hacia el sonido. Una muestra de ello es la gran cantidad archivos y recursos activos

en la web, de los que podemos descatar: *Center of Visual Music*, *Iota Center*, *See this Sound* y *Visual Music Archive*, como verdaderos contenedores de información sobre el campo de visualización, donde podemos encontrar desde artistas precursores del género hasta las más novedosas tendencias actuales.

El Center of Visual Music[4] (CVM) es un archivo fílmico, sin ánimo de lucro dedicado a la música visual, animación experimental y cine abstracto, con base en Los Angeles. El centro gestiona obra, libros, revistas y catálogos de artistas como Richard Baily, Jordan Belson, John Bachanan, Scott Draves, Wiking Eggeling, Oskar Fischinger, Ken Jenkins, Len Iye, Neubauer, Semiconductor, Jeff Perkins, Man Ray, Jurgen Reble, Richard Reeves, Hans Richter, Elias Romero, Walter Ruttmann, Robert Seidel, Paul Sharits, George Stadnik, Stan VanDerBeek, Joshua White, Thomas Wilfred entre otros. Este centro presenta una línea clásica pero muy amplia del campo de visualización. Mantiene un sistema de socios o miembros del centro, de manera que pueden publicitar sus artículos, consultar la videoteca, descuentos en libros y una serie de beneficios adicionales.

En una línea similar al CVM, encontramos el *Iota Center*[5] especializado en cine abstracto, animación y películas de artistas de la costa oeste de Estados Unidos de América. Activo desde 1994 abarca desde conservación hasta investigación y exhibición de obras. Como ocurre en el CVM adolece de la promoción de obras con tecnología más actual.

[4] La página web del Centro es: http://www.centerforvisualmusic.org/. Consultado 4/12/2015

[5] La página de web del Centro es http://www.iotacenter.org. Consultado 4/12/2015.

El archivo web *See This Sound* [6], arranca en 2009 como un proyecto para Linz 2009, capital de cultura europea, realizado en estrecha colaboración con *Lentos Art Museum Linz*[7] y *Ludwig Boltzmann Institute Medien.Kunst.Forschung*[8]. Actualmente lo mantiene y actualiza la *Academy of Visual Arts Leipzig.* El proyecto aborda también exposiciones y congresos a cerca de las conexiones entre imagen y sonido en los ámbitos del arte, medios audiovisuales y percepción. El sitio web está organizado por una serie de temas: cine abstracto, visuales en tiempo real, sonidificación, sinestesia,…, para cada uno de los temas se ha desarrollado un artículo amplio y una relación de artistas y obras relacionados con el tema. Es bastante completo y abarca desde los orígenes hasta la actualidad.

Por último nos gustaría destacar el archivo web *Visual Music Archive*[9]*,* se trata de una colección no institucional, y como apuntan desde el propio archivo altamente subjetiva de trabajos inspirados en el amplio campo de la Música Visual. A nuestro juicio representa la mejor y mayor recopilación de material relacionado con las intersecciones entre el sonido y la imagen. Desde autores y proyectos clásicos hasta las últimas tendencias. Adicionalmente hay referencias a prácticamente toda la literatura del campo.

En lo que resta del capítulo realizaremos un breve repaso histórico, a través de las tres etapas fundamentales que nos sirven de marco referencial: instrumentos visuales, medios audiovisuales y sistemas digitales. Dentro de estas etapas vamos a resaltar tendencias y autores que estén o bien directamente relacionados con los objetivos

---

[6] La página web del archivo web es: http://www.see-this-sound.at/. Consultado 4/12/2015.

[7] La página web del museo es: http://www.lentos.at/html/de/index.aspx. Consultado 4/12/2015.

[8] La página web del instituto es: http://www.lbg.ac.at/de/themen/lbi-fuer-medienkunstforschung. Consultado 4/12/2015.

[9] La página del archivo web es: http://visualmusicarchive.org. Consultado 4/12/2015.

de este trabajo o que hayan influenciado de forma indirecta con el desarrollo del trabajo. Fundamentalmente vamos a manejar dos ejes principales:

- La abstracción como corriente global de arte no figurativo y a nuestro entender estética natural del sonido.
- Imagen en movimiento generada a partir de propiedades directas del sonido o altamente influenciadas por éstas. En este ámbito realizaremos un mayor hincapié en las propuestas donde la imagen es generada de forma sintética y en tiempo real.

Finalmente dedicaremos un pequeño subapartado para incluir otros aspectos artísticos y técnicos que hemos tenido en cuenta en el desarrollo de la investigación.

2.1 Instrumentos visuales

Desde la antigüedad el ser humano ha especulado sobre la relación de los sonidos y los colores. Los filósofos griegos Aristóteles y Pitágoras establecieron correlaciones entre las notas de escala musical y el espectro del arco iris. Pero fue durante el siglo XVIII y el siglo XIX donde se sentaron las bases para el diseño de los primeros instrumentos de color. Isaac Newton en su tratado sobre óptica de 1704 ideó una correspondencia entre el espectro de colores y las notas sucesivas de la escala musical, de forma que el Do era rojo, Re era naranja, Mi era amarillo, Fa era verde, Sol azul, La era índigo y Si violeta (Peacok 1988). En esa época todavía se pensaba que la luz y el sonido eran físicamente iguales. Estas teorías inspiraron la construcción de instrumentos de color, principalmente basados en clavicordios o pianos, de forma que presionando las teclas se pudiera generar imagen e incluso sonido de forma simultánea.

En 1730, el matemático francés Louis Bertrand Castel construyó un instrumento visual que denominó Clavicordio Ocular. Posiblemente es el primer instrumento musical con potencialidad visual, que se conoce. Este dispositivo era básicamente un clavicordio modificado de forma que cuando se pulsaba una tecla se corría una cortina desvelando así una ventana de cristal coloreado, a través de la cual pasaba un haz de luz procedente de unas velas y proyectaba el color asociado a la tecla. En la figura 2.1.1, podemos ver una caricatura del instrumento, ya que no se conserva ningún instrumento, ni ninguna imagen real. En (Franssen 1991) se realiza un estudio pormenorizado del instrumento y del impacto que tuvo. Castel soñaba que algún día en cada hogar habría un clavicordio ocular.

Figura 2.1.1. Caricatura del clavicordio ocular de Louis Bertrand Castel[10]

[10] Se trata de una caricatura de Charles Germain de Saint Aubin.

En el siglo XIX, el inventor estadunidense Bainbridge Bishop se impuso la tarea de construir un mecanismo efectivo y práctico para poder generar de forma simultánea colores y sonido. Su correspondencia de color a tono[11] musical se fundamentó en la correlación del color a la escala musical cromática. En 1877 y después de varios intentos Bishop construyó un órgano de color como podemos ver en figura 2.1.2. En (Bishop 1893) se puede consultar una descripción muy detallada de éste órgano de color.

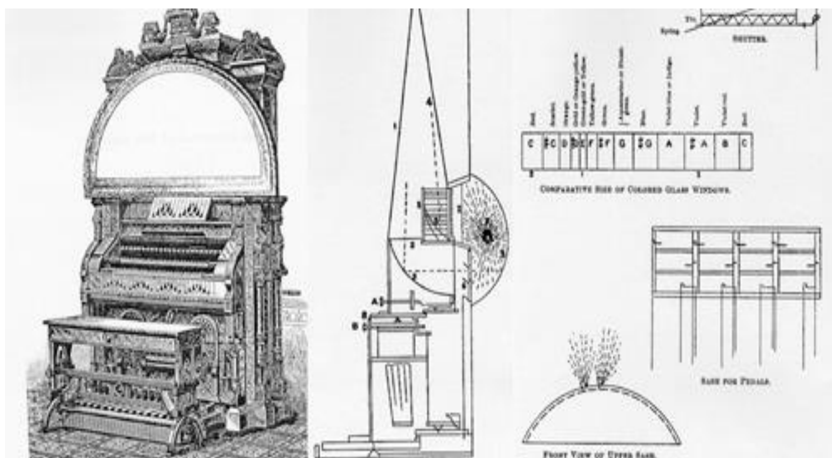


Figura 2.1.2 Órgano de color de Bishop

En 1893, el pintor británico Alexander Wallace Rimington diseñó un órgano de color (Wallace 1912), obteniendo una proyección mucho más luminosa a través de un sistema eléctrico de alimentación de los filtros de colores (ver figura 2.1.3). Es el primer investigador en denominar a su invento órgano de color. Rimington creía que existían analogías físicas entre el color y el sonido, de forma que su escala de color-música se basaba en dividir el espectro de la luz en intervalos de las mismas proporciones que las distancias entre las notas de la

[11] Al final del capítulo en la figura 2.1.9, se puede ver una tabla comparativa con diferentes equivalencias entre colores y notas musicales.

escala musical. Cada octava por lo tanto contenía los mismos tonos de colores, diferenciándose por la cantidad de luz blanca que contenían. Las octavas superiores estaban más iluminadas. El instrumento no podía generar color y sonido simultáneamente, así que habitualmente se tocaba a dúo con otro órgano tradicional.

Figura 2.1.3 Órgano de color de Alexander Wallace Rimington[12]

En 1916, el pintor futurista Vladimir Baranoff Rossiné, construye el Piano Optofónico, un piano eléctrico que generaba efectos ópticos sobre los que podía controlar la intensidad de la luz (ver figura 2.1.4). En 1924 en el teatro Bolshoi de Moscú presentó su instrumento, se trataba de un instrumento sinestésico capaz de crear sonidos y luces de colores, patrones y texturas de forma simultánea. El Piano Optofónico era un instrumento muy sofisticado y capaz de generar muchas texturas diferentes.

[12] Exterior del Órgano de Color, imagen obtenida del libro (Wallace, 1912)

Figura 2.1.4. Piano Optofónico y discos transparentes.

El músico y pintor danés Thomas Wilfred en 1919 construyó el Clavilux, inspirado en los primeros órganos de color. En lugar de centrarse en teorías de la luz y correlaciones con las notas musicales centro su trabajo en la propia luz. Denominó Lumia al nuevo arte de la luz y construyó el Clavilux (ver figura 2.1.5). En (Collopy 2000) se realiza un estudio pormenorizado del instrumento.

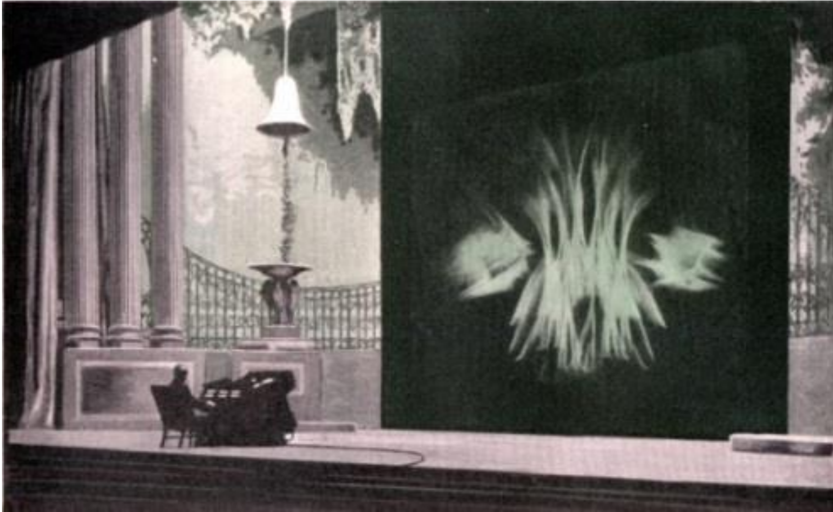


Figura 2.1.5. Clavilux de Thomas Wilfred.

En 1921 Adrian Klein[13] diseñó un proyector de color, que consistía en un gran espectroscopio que dispersaba la luz en una gran pantalla, la proyección era controlada mediante un teclado. Klein realizó muchos conciertos acompañados de música, improvisando a partir de un rango de 150 combinaciones de colores (ver figura 2.1.6).

[13] Es más conocido como Adrian Cornwell Clyne.

Figura 2.1.6. Proyector de color de Adrian Klein.

En Italia, Achille Ricciardo construyó un instrumento de color para el Teatro del Colore de Roma en 1920. En ese mismo periodo Richard Lovstrom patenta otro instrumento para generar color y música en Estados Unidos. En 1925 el compositor Alexander László, construye el Sonchromatoscope (ver figura 2.1.7).

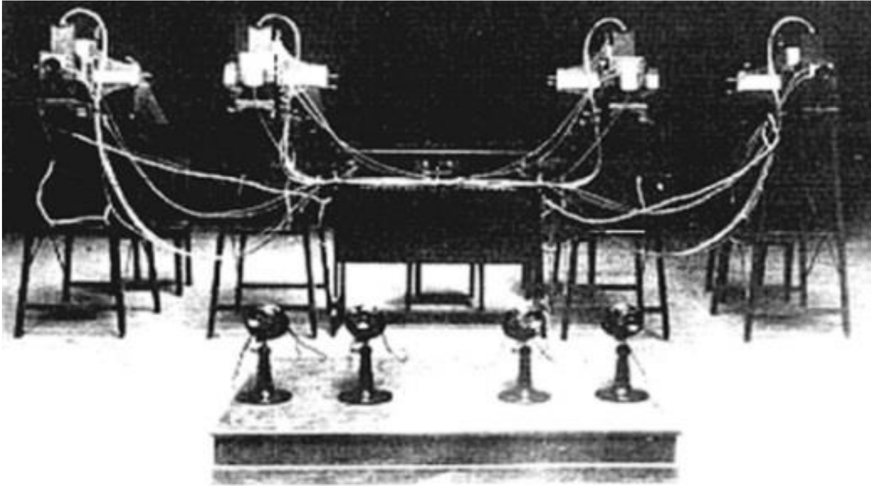


Figura 2.1.7. Sonchromatoscope de Alexander László.

Mary Elizabeth Hallock-Greenewalt entre 1916 y 1934 construyó el órgano de color que bautizaría como Sarabet (ver figura 2.1.8). Ella se distinguió por crear un arte que denominó Nourathar, que es una adaptación de las palabras árabes luz y esencia. A diferencia de otros autores no estableció una correspondencia estricta entre colores y notas musicales. Ella pensaba que estas relaciones eran variables y debían reflejar el temperamento y la habilidad del intérprete.

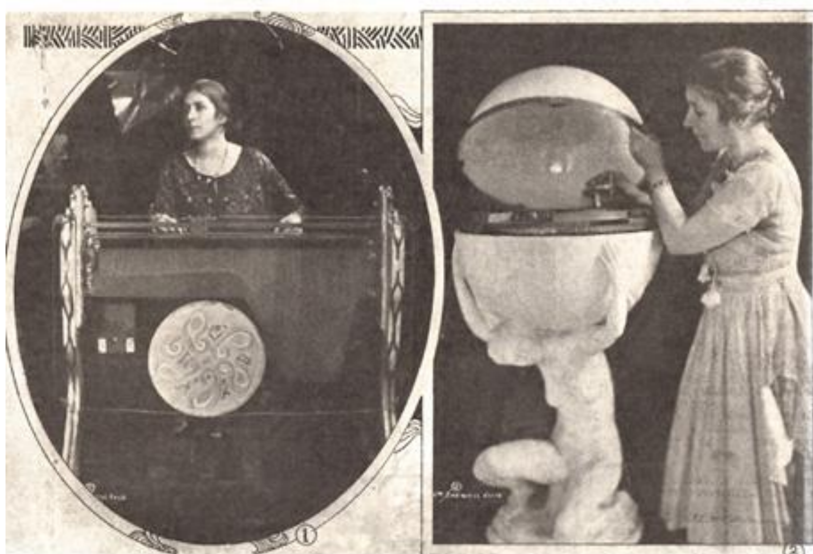


Figura 2.1.8. Órgano de color de Hallock-Greenewalt.

En (Levin 2000), (Peacock 1988) y en (McDonnell 2007) se realiza un estudio más detallado de las aportaciones históricas en este campo.

En la figura 2.1.9 podemos ver una tabla comparativa de correspondencias entre notas musicales y colores de diferentes autores a lo largo de los tiempos, de la que podemos concluir la falta de un criterio uniforme para establecer relaciones estables. Pese a todo en la actualidad podemos encontrar aplicaciones informáticas que realizan estadísticas sobre asociaciones entre color y sonido, como el experimento que se realiza en la página web Color of My Sound[14]. También en el trabajo (Palmer 2013) se realiza un experimento buscando conexiones entre el sonido, el color y las emociones.

[14] Color of My Sound es un experimento sinestésico, http://www.colorofmysound.com/ (consultado 1/12/2015)

	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
Newton 1704												
Castell 1734												
Field 1817[15]												
Bishop 1893												
Rimington 1893												
Scriabin 1911[16]												
Klein[17] 1930												

Figura 2.1.9. Tabla comparativa entre colores y sonidos[16][17][18].

[16] George Fields fue un químico inglés, de conocida reputación en el campo del color. En su ensayo titulado “Chromatics, or, an essay on analogy and harmony of colours” de 1817 propone esta correspondencia de tonos y colores.

[17] Aleksandr Nikoláyevich Scriabin fue un compositor y pianista ruso, con capacidades sinestésicas.

[18] Para más detalles sobre las propuestas de Klein consultar (Klein 1927).

De esta primera etapa, que se inicia con los filósofos griegos en su incipiente búsqueda de relaciones entre sonido e imagen y llega hasta principios del siglo XX, podemos concluir:

- Bajo la idea que el sonido y la luz físicamente son iguales, el esfuerzo en la mayoría de instrumentos se centra en obtener una correspondencia unívoca entre notas musicales y colores. Esta correspondencia en la mayoría de los casos es subjetiva y se sustenta en las capacidades sinestésicas de los autores.
- El tiempo real está implícito en todas las propuestas, puesto que no se utilizan medios de almacenamiento sonoro, ni visual. Se busca que el espectador tenga una experiencia multisensorial. Es importante resaltar este hecho puesto que con la aparición de los medios audiovisuales como el cine o el video, se pierde parte de este espíritu de *performance*.
- Al menos en las propuestas de Hallock-Greenewalt, Wilfred y Baranoff, el planteamiento de la visualización es diferente ya que proponen un nuevo arte de la luz. No se trata de realizar una correspondencia directa, sino de presentar un nuevo lenguaje basado en la luz y el sonido.

La tradición del desarrollo de instrumentos visuales se ha mantenido hasta nuestros días, donde en la mayoría de los casos el piano se ha sustituido por interfaces basadas en sensores y los mecanismos físicos de generación de imágenes por sistemas informáticos. Prácticamente todos los artistas revisados en el apartado de sistemas digitales abordan esta línea de trabajo.

2.2 Medios audiovisuales

Este apartado lo hemos divido en tres partes. Por un lado toda la corriente de cine abstracto y experimental que surge con la llegada del cine, por otro lado el nuevo panorama que se presenta con el desarrollo de la imagen electrónica y sus posibilidades de síntesis y sincronización con el sonido, y por último cómo estas corrientes estéticas y tecnológicas se popularizan a través del formato de videoclip.

2.2.1 Música Visual

Paralelamente al diseño de instrumentos visuales con la llegada del cine y las nuevas concepciones estéticas que impulsan las vanguardias se abren nuevas perspectivas en las relaciones entre la música y las artes visuales. El cine se presenta como el medio idóneo al posibilitar la coexistencia sincronizada del sonido y la imagen. El nuevo medio además de capturar la realidad sonora y visual proporciona en si un nuevo lenguaje, el lenguaje del tiempo a través del cual desarrollar nuevas relaciones entre el sonido y la imagen. Es por lo tanto el cine con su capacidad de almacenar imagen y sonido, el medio que propiciará las nuevas direcciones de la visualización del sonido.

El punto de inflexión lo encontramos en la primera década del siglo pasado, con los futuristas italianos, que tanto han influido en las vanguardias cinematográficas. En 1910, el artista Arnaldo Ginna[19] *ya* plantea la pintura no figurativa como una forma de expresar

[19] Arnaldo Ginna fue un pintor, cineasta y escultor italiano, participó activamente en el moviento futurista, las ideas tempranas de abstracción están plasmadas en su libro *Method and New Life.*

sentimientos y estados de ánimos. Junto con su hermano Bruno Corra es uno de los primeros cineastas en usar la técnica de pintar directamente sobre el celuloide. Esta técnica será la base fundamental para el desarrollo del cine abstracto. Con el manifiesto de la cinematografía futurista de 1916, se plantean los cimientos de las posibilidades del nuevo formato en el ámbito artístico:

"En la película futurista entrarán como medios de expresión los elementos más diferentes: desde el tema de vida real a la mancha de color, de la línea a las palabras en libertad, de la música cromática y plástica a la música de objetos. Será por lo tanto pintura, arquitectura, escultura, palabras en libertad, música de colores, líneas y formas, revoltijo de objetos y realidad caotizada"[20]

Junto con los futuristas en los primeros años del siglo XX, artistas como el alemán Hans Stoltenberg y el finlandés Leopoldo Survage comienzan a pintar directamente sobre el celuloide. Survage pensaba que cuando el color se mueve de la misma forma que el sonido, rítmicamente, su resultado visual era mucho más impactante que el color estático de la pintura. A través de su película de animación *Rythme Coloré*[21], trató de exponer estas ideas, pero tuvo que paralizar el proyecto debido a la llegada de la Segunda Guerra Mundial. Pese a todo nos han llegado las pinturas que realizó en la preparación del film (ver figura 2.1.10).

[20] Manifiesto de cine futuristas. Está firmado por Filippo Tommaso Marinetti, Bruno Corra, Emilio Settimelli, Arnaldo Ginna, Giacomo Balla, Remo Chiti. En http://el-cuadernode-alp.blogspot.com.es/2011/05/el-manifiesto-del-cine-futurista-milan, se puede consultar una traducción del manifiesto al castellano. Consultado 8/12/2015.

[21] Se trata de una serie de cuadros abstractos pensados para formar parte de una animación, que pudiera ser presentada como sinfonías de color.

Figura 2.1.10. Estudio para la película *Rythme Coloré* de Survage.

El movimiento de Abstracción tanto en su vertiente plástica como cinematográfica formaliza en gran medida su estética en los principios de composición musical (Kandinski 1996). Es justamente cuando el arte pierde la figuración cuando se dan las condiciones idóneas para plantear las relaciones históricamente buscadas entre la luz y el sonido. Si el sonido se puede ver sin duda su estética será abstracta, carente de una relación directa con el mundo tangible y real. Es en este momento histórico donde podemos decir que realmente se gesta la Música Visual, como movimiento independiente y bien definido. Más tarde con la llegada del video y los ordenadores se volverá a diluir en el entramado de las nuevas corrientes, pero su formulación se plantea clara en los inicios del siglo XX.

La pintura abstracta basada en la composición musical se ha considerado como se propone en (Ox 2006) una vertiente de la Música Visual. A partir de unas fuentes visuales no figurativas se estructura la composición, usando conceptos y estructuras musicales. De esta forma la pintura puede reflejar el ritmo, la armonía y el movimiento del sonido, de la misma manera que una partitura refleja de forma estática la ordenación temporal de las notas. En el lienzo los elementos geométricos se distribuyen rítmicamente por el espacio, como los sonidos en el aire.

Entre 1910 y 1913, Wassily Kandinsky realizó su primera acuarela abstracta (ver figura 2.1.11), y en su libro De lo espiritual en el Arte (1996) plantea las posibles conexiones entre la música y el color, aunque realmente se centra mucho más en los aspectos místicos de la creación artística. En (Martinez 2011) se realiza una nueva revisión[22] de cuales fueron los elementos claves de Kandinsky para la gestación de la abstracción, en base a su capacidad sinestésica y las teorías de neuroestética[23].

---

[22] La revisión se realiza en base a una nueva lectura de su biografía y una nueva interpretación de los escritos de Kandinsky.

[23] La neuroestética es una rama de la filosofía, que busca las bases biológicas y neuronales de la creatividad y la belleza.

Figura 2.1.11.Primera acuarela abstracta de Kandinsky.

Muchos de los trabajos de Kandinsky tienen títulos directamente relacionados con el sonido y la música, como *Sonido Apacible*, *Diagonale* o *Constrasting Sounds.* Estos trabajos fueron realizados entre 1923 y 1924 (ver figura 2.1.12).

Figura 2.1.12. *Diagonale, Sonido Apacible* y Contrasting Sound.

En 1911 Frantisek Kupka con su pintura *Nocturno* (ver figura 2.1.13), alude también de forma explícita a las piezas musicales para piano de expresión nostálgica.

Figura 2.1.13. *Nocturno* de Kupka, 1911.

Son muchos los artistas que trabajaron la composición visual usando estructuras musicales. Las pinturas de Paul Klee son un ejemplo de cómo la música es el marco para el desarrollo de las ideas artísticas. Klee estaba interesado en la armonía y las reglas del contrapunto[24]. Se interesó en la definición de un vocabulario de arte abstracto comparable al musical. Otros artistas que trabajaron en líneas similares son El Lissitzki, Alexej Jawlenski, Hans Arp o Sophie Taeuber.

Junto a la pintura abstracta se desarrolla la vertiente abstracta del cine. En los primeros años de la década de 1920 el cine abstracto alemán o cine absoluto representa un cine basado en formas abstractas obtenidas del mundo real o directamente dibujadas sobre

[24] El contrapunto es una técnica de composición musical que se basa en la escritura de líneas musicales muy diferentes y normalmente para instrumentos también diferentes, de forma que al sonar de forma simultánea se obtiene un equilibrio armónico.

la película. Las formas evolucionan y se transforman a lo largo del tiempo, según estructuras importadas de la música. En la mayoría de los trabajos se prescinde del sonido en el resultado final, y es en la estructura de la composición visual donde se aplican los conceptos musicales, como el ritmo y las teorías del contrapunto. Walter Ruttmann, Hans Richter, Viking Eggeling y Oskar Fischinger son los principales representantes de esta corriente.

De Walter Ruttmann podemos destacar la serie de películas, *Lichtspiel Opus I, II, III, IV* realizadas entre 1921 y 1925 (ver figura 2.1.14). La técnica principal usada en estas películas era ir fotografiando fotograma a fotograma unas placas de vidrio sobre las que iba pintando formas geométricas con óleo. Tras cada pincelada sobre el vidrio realizaba una fotografía, en muchas ocasiones además incluía recortes geométricos en una capa separada. Para *Lichtspiel Opus I*, Ruttman encargó al compositor Max Butting la composición de un cuarteto de cuerdas, estableciendo unas indicaciones claras a los músicos para sincronizar los elementos visuales con los sonoros. En (Moritz 1997) se presenta una descripción más detallada e interesante sobre las técnicas que uso en el desarrollo de estas películas.

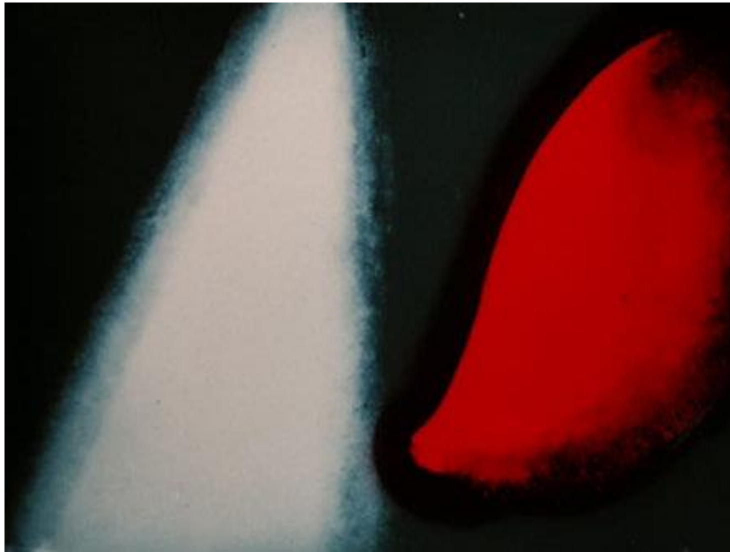


Figura 2.1.14. Fotograma de *Lichtspiel Opus I,* Ruttman, 1921.

El artista alemán Hans Richter integrante del movimiento dadaísta y activista político, en 1921 realiza una de las primeras películas abstractas[25], *Rhythmus 21* (ver figura 2.1.15). Se trata de un excelente ejemplo de orquestación basada en el tiempo. Las formas rectangulares muy sencillas evolucionan con el tiempo, modificando su tamaño, la propia pantalla es en sí un elemento de la composición.

[25] Ritcher afirmaba que su película *Rhythmus 21* era la primera película abstracta.

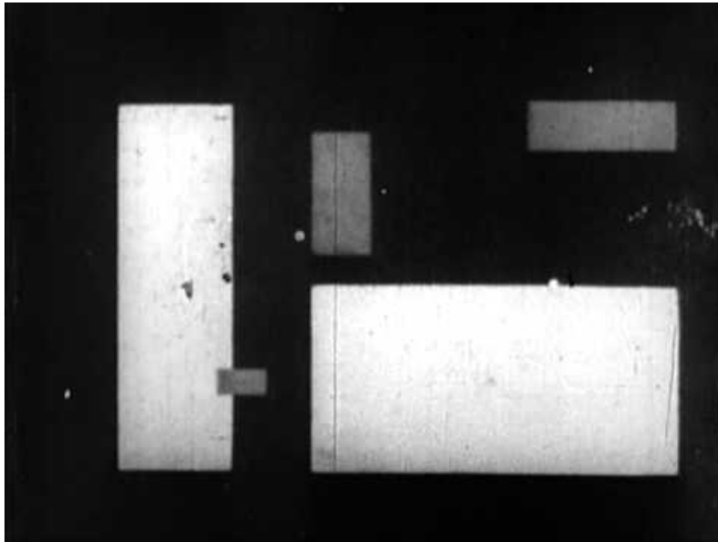


Figura 2.1.15. Fotograma de *Rhythmus 21*, Ritcher, 1921.

Viking Eggeling vinculado también al dadaísmo es uno de los fundadores del cine absoluto y según palabras de César Ustarroz:

"*Eggeling, que anteriormente había trabajado con Hans Richter, crea una composición en la que el ritmo, armonía y tempo musical son analizados exhaustivamente para encontrar una equivalencia sonora en los movimientos y las proporciones de líneas y figuras. Synphonie Diagonale*[26] *se erige como una de las obras cumbres del cine abstracto de los años 20"* (Ustarroz 2013)

[26] Ver figura 2.1.16.

Figura 2.1.16. Fotograma de *Synphonie Diagonale*, Eggeling, 1924.

La técnica usada en *Synphonie Diagonale,* consistió en fotografiar fotograma a fotograma figuras que habían sido creadas recortando papel de estaño y cartón. Las figuras van evolucionando en el tiempo y se componen como si de una orquesta se tratara. El trabajo no tenía sonido.

Oskar Fischinger fue otra de las figuras clave en el desarrollo del cine abstracto. Una de las características es que además de artista plástico tuvo formación musical, e incluso tenía conocimientos sobre la construcción de órganos. Esto le llevó a colaborar con el compositor húngaro Alexander Laszlo en el desarrollo del *Sonchomatoscope,* una especie de órgano de color. Finalmente terminó diseñando su propio órgano de color, el *Lumigraph*. Debido a su formación musical centro gran parte de su trabajo en la

visualización de obras de música clásica, como el poema óptico de Listz, *Rapsodia húngara nº2*, para los estudios MGM[27]. Posiblemente es el artista que más contribuyó a la sincronización de la imagen y el sonido. No sólo usando conceptos musicales en la composición visual sino estableciendo puntos de sincronización entre lo visual y lo sonoro. Hay que tener en cuenta que esta sincronización que actualmente con los ordenadores es una tarea bastante sencilla, a principios del siglo XX consistía en un proceso artesanal. Primero se debía grabar el sonido en la película de cine y luego mirando la forma de la onda y calculando tiempos se dibujaban las animaciones o se superponían los recortes.

En 1931 realizó una serie de experimentos, buscando la relación inversa entre el sonido y la imagen, esto es a partir de imágenes generar sonido. Estos experimentos se conocen como *Ornament Sound,* la idea era dibujar o más bien fotografiar la banda de sonido de la película de cine con una serie de patrones abstractos (ver figura 2.1.17).

[27] Estudios Metro Golwyn Mayer.

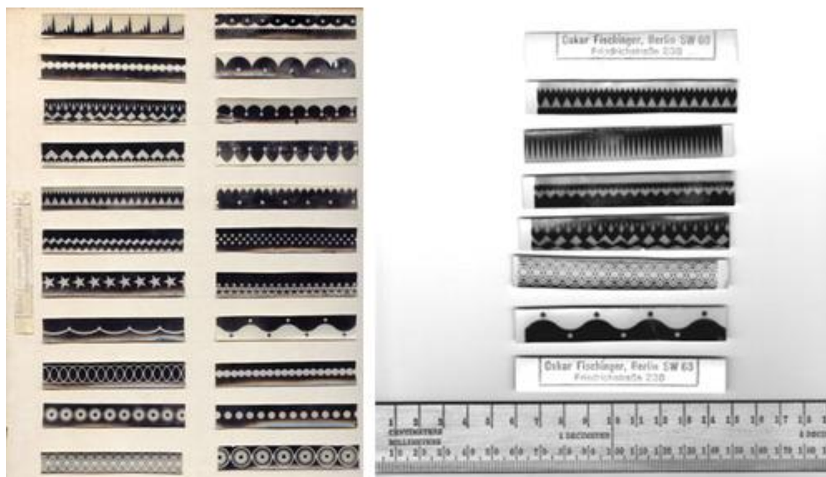


Figura 2.1.17. Experimentos de *Ornament Sound*. Fischinger.

El cine abstracto alemán definió las bases técnicas de animación cinematográfica e introdujo los conceptos musicales en la composición visual de las imágenes, ejerciendo una gran influencia en la siguiente generación de cineastas experimentales, donde encontramos a Len Lye, Harry Smith, los hermanos Whitney, Jordan Belson, Chris Larkee, Bärbel Neubauer, Larry Cuba y Norman McLaren entre otros (Moritz 1997, Wheeler 2002).

El neozelandés Len lye fue un cineasta experimental y escultor cinético, que fundamentó todo su trabajo en la composición visual del movimiento. Él pensaba que:

"si estaba la composición musical, podría estar también la composición del movimiento. Después de todo, hay figuras melódicas, ¿por qué no puede haber figuras del movimiento?" (Lye 1984).

Color Box de 1935 (ver Figura 2.1.18) es su primera película en la que usa la técnica de pintar y rascar directamente la película. No obstante la utilización del sonido o conceptos del sonido en la

composición visual son muy escasos en sus trabajos o los deja a un segundo plano, el principal esfuerzo lo realizó en la parte del movimiento de las figuras y las formas. Influyó decisivamente en los trabajos de Norman McLaren.

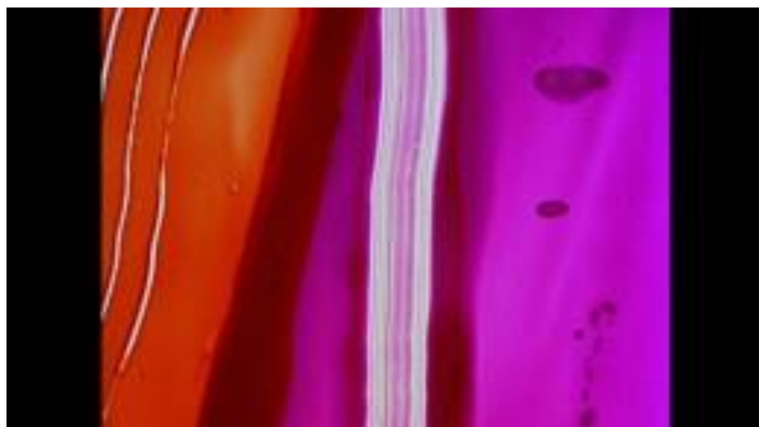


Figura 2.1.18. Fotograma de *Color Box*, Lye, 1935.

Norman McLaren fue un artista clave de la primera mitad del siglo XX. Desarrolló muchas técnicas manuales de animación tanto en el sonido como en la imagen, que se usan actualmente. En (Baquedano 1987) se realiza un estudio pormenorizado de toda su obra completa, incluidas entrevistas al propio autor. De entre sus trabajos podemos destacar *Dots y Loops* de 1940 (ver figura 2.1.19), en donde el sonido y la imagen son construidos de forma sintética, logrando de esta forma una conexión completa entre la imagen y el sonido. En el video de su autobiografía[28], McLaren cuenta que cuando era joven y escuchaba música cerraba los ojos y veía formas geométricas que danzaban con la música y al entrar en contacto con los trabajos de Fischinger y Lye, redescubrió esas impresiones de la juventud.

[28] que filma en National Film Board of Canada

Figura 2.1.19. Fotograma de *Loops*, Norman McLaren, 1940.

El estadunidense Harry Everett Smith, artista, bohemio y místico desarrolló entre 1946 y 1958 la serie de películas denominadas *Early Abstractions* (ver figura 2.1.20). Mediante técnicas de rayado y dibujo a mano, *collage*[29] y grabación fotograma a fotograma explora los principios rítmicos de la imagen y su correspondencia con el lenguaje de la música. Smith hablaba de sinestesia y de la búsqueda de correspondencias entre el color, el sonido y el movimiento.

[29] El *collage* es una técnica artística que consiste en ensamblar en un lienzo diferentes elementos.

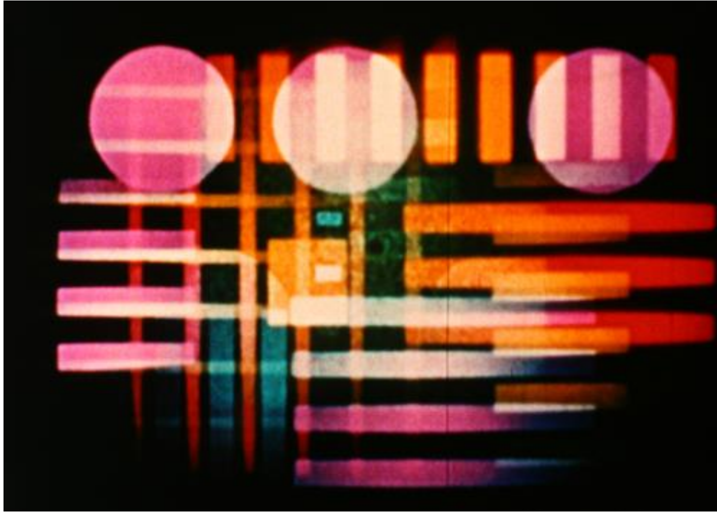


Figura 2.1.20. Fotograma de *Early Abstractions*, Smith, 1946-58.

Por otro lado durante la primera mitad del siglo XX, los científicos desarrollan dispositivos de análisis y visualización de sonido, como el osciloscopio y el espectograma (Koening 1946, Potter 1947). Estos descubrimientos científicos que permitían ver la onda sonora no pasaron inadvertidos en el campo del arte y artistas como Ben Laposky y Marie Ellen Bute experimentaron con estos nuevos instrumentos.

El osciloscopio fue desarrollado en 1874 por el alemán Carl Ferdinand Braun, sentando las bases para el desarrollo del tubo de imagen del televisor y el radar. La idea del osciloscopio era representar gráficamente señales periódicas eléctricas, como la señal sonora. Ben Laposky, matemático y artista en 1950 empezó a usar el osciloscopio analógico para crear patrones de luz electrónicamente, que posteriormente registraba mediante una película de alta velocidad. A estas fotografías las llamaba “oscilones” o “abstracciones electrónicas” (ver figura 2.1.21).

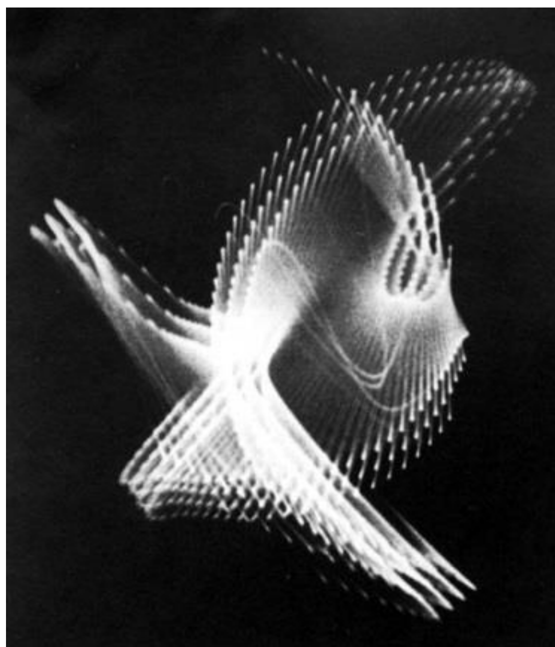


Figura 2.1.21. Oscilón nº4, Laposky, 1950.

La cineasta Mary Ellen Bute es sin duda uno de los referentes principales de la visualización creativa del sonido, estableciendo los cimientos para ver el sonido. Trabajó con Leon Theremin[30], Thomas Wilfred y Norman McLaren además de estar muy influenciada por Oskar Fischinger (Moritz 1996).

En 1950 empezó a usar patrones de un osciloscopio como elementos principales para sus películas. *Abstronic* (1954) (ver figura 2.1.22) y *Mood Contracts* (1956), son dos de los trabajos en los que usaría imágenes de osciloscopio, incluso en su publicidad aseguraba que era la primera persona en combinar arte y ciencia. Aunque Norman McLaren ya había usado anteriormente dicha técnica en su película

[30] Leon Theremin fue un inventor ruso, conocido por haber desarrollado el Theremin, uno de los primeros instrumentos musicales electrónicos.

Around is Around de 1950. Por otro lado también Hy Hirst había experimentado con esa técnica (Moritz 1996).

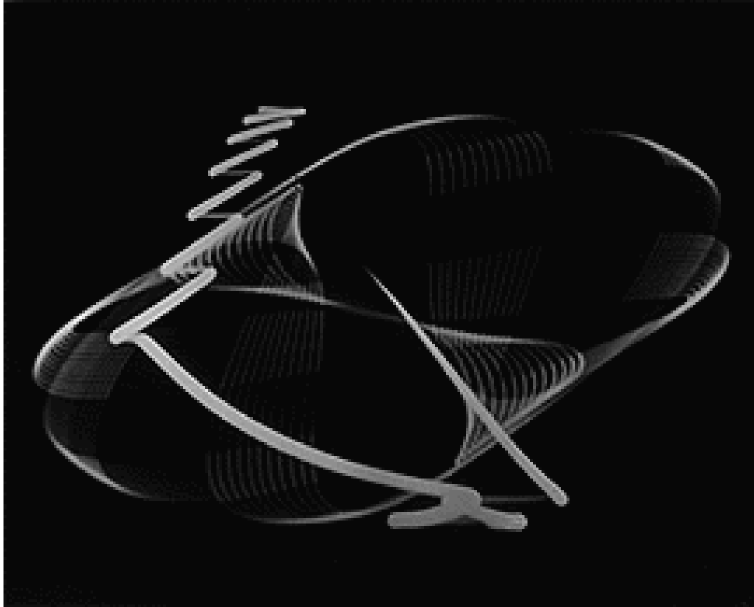


Figura 2.1.22. Fotograma de *Abstronic*, Bute (1954).

El cine abstracto y experimental de la primera mitad del siglo XX marca la nueva dirección de las relaciones entre la imagen y el sonido. Como con los instrumentos visuales su desarrollo ha continuado hasta nuestros días. De esta segunda gran etapa podemos destacar:

- El sonido y la imagen se almacenan en el mismo medio y por lo tanto se pueden sincronizar.
- La estética que se plantea para visualizar el sonido es la ausencia de figuración. El sonido se ve mediante figuras geométricas y texturas que evolucionan en el tiempo, donde sus movimientos se formalizan a través de conceptos musicales de orquestación, contrapunto, ritmo o armonía.

- La forma con la que se crean imágenes es pintando o colocando máscaras directamente sobre el celuloide, o fotografiando cuadro por cuadro, la sincronización con el sonido se establece mirando físicamente la pista de sonido o realizando cálculos. Incluso el sonido se puede pintar.
- Con el cine se pierde la idea de tiempo real, que tenían los instrumentos visuales, aunque con la llegada del video y posteriormente los sistemas informáticos se volverá a recuperar.

Para una profundización en el campo de la música visual se pueden consultar los trabajos de (Brougher 2005), Daniels 2009) o (Kock 1971).

2.2.2 VideoArte

Si con los trabajos de Laposky y Bute se empezaron a entre ver las relaciones directas entre tecnología y el arte, con la llegada del video estos lazos se terminan de fusionar y se dan las condiciones óptimas para que artistas y técnicos establezcan equipos de trabajo multidisciplinares. Esta estrecha relación entre técnicos y artistas se mantendrá hasta nuestros días y aunque ya en esa época se vislumbraba la convergencia a un artista/técnico, todavía hoy en día supone un reto a alcanzar, aunque se han realizado muchos avances. Gene Younblood apuntaba hacia esa nueva concepción del artista:

"La imagen del artista ha cambiado radicalmente. En el nuevo arte conceptual, es la idea del artista y no su habilidad técnica manipulando los medios lo que prima. Aunque actualmente, un gran énfasis se emplaza en la colaboración entre artista y técnicos, la

verdadera tendencia apunta hacia un individuo que conversa al mismo tiempo artística y técnicamente” (Younblood 1970)[31]

En los años sesenta y como reacción a los *mass media*, surge la estética del videoarte y la videoinstalación que en cierta medida implican una continuación del cine experimental. El abaratamiento de los costes de producción que supone el formato de video analógico y su manipulación electrónica proporciona el entorno adecuado para que los artistas investiguen nuevas formas de presentar la imagen y el sonido.

Artistas como Nam June Paik, el grupo Fluxus, Steina y Woody Vasulka, Gary Hill, Dan Sandin y Zbigniew Rybczinski entre otros son considerados como pioneros de estas nuevas estéticas que surgen con la imagen electrónica. Woody Vasulka recoge el espíritu de esta nueva etapa:

“Yo empecé con la luz, luz y sombra, una práctica fílmica típica; empecé trabajando con luces estroboscópicas. Entonces encontré el vídeo cuyos principios niegan esencialmente al cine. Dejé la luz al instante. El vídeo era un territorio indefinido, libre, no competitivo, un medio muy libre. La comunidad era ingenua, joven, fuerte, cooperativa, una tribu acogedora.” (Hill 1995)[32]

Pese a que el primer sistema de video casero lo comercializó la empresa Ampex en 1963, el equipo que supuso la revolución audiovisual fue el PortaPack[33] de Sony comercializado en 1967 (ver figura 2.1.23). El mundo exterior podía ser capturado y de forma inmediata ser reproducido, visual y sonoramente. El sistema

[31] También citada literalmente en (Ustarroz 2013)
[32] Entrevista realizada a Woody Vasulka.
[33] Se trataba del model DV-2400 Video Rover, formado por cámara y magnetoscopio.

PortaPack ofrecía inmediatez, facilidad de manejo, comodidad de transporte, captura simultánea de sonido e imagen y un precio razonable. De manera que su popularización a nivel doméstico y profesional cambió definitivamente el rumbo de la imagen en movimiento. Los artistas recogieron el nuevo invento para dar un paso más en sus investigaciones y la industria televisiva pudo al fin liberarse de la lacra del cine y del directo, modificando desde los cimientos toda su estructura.

Figura 2.1.23. Imagen publicitaria de la PortaPack de Sony.

Durante esta época la terminología cambia y los nuevos conceptos que se empiezan a manejar son los de procesamiento, postproducción, efectos, síntesis. El artista recoge las imágenes y los sonidos y los procesa, los transforma. La postproducción no consiste en ensamblar imágenes y sonidos sino que se convierte en el centro del proceso creativo donde las herramientas son las imágenes

electrónicas, que se fusionan, se transforman y sintetizan. La imagen interacciona con el lenguaje y con la música, en una comunicación de reciprocidad.

Con el cine, la imagen es fotográfica y completa, la unidad mínima de información es el cuadro, el sonido se puede ver, se almacena junto al cuadro, pero los dos medios permanecen separados, su interacción es artesanal, medida y dibujada. El tratamiento de la imagen electrónica viene precedido y fundamentado por los avances en el tratamiento del sonido electrónico y los esquemas de síntesis. En el modelo de síntesis por modulación, mediante dos señales eléctricas simples se podía generar una gama muy amplia de timbres, simplemente relacionando las dos señales. Esa idea de procesamiento y transformación heredada del campo de la acústica electrónica fue la que realmente estableció el punto de arranque de las nuevas estéticas.

Woody y Steina Vasulka comenzaron a trabajar con ondas generadas por sintetizadores de sonido tratando que influyeran en la pantalla de un televisor. Nan June Paik empezó manipulando las imágenes electrónicas con imanes. Esta necesidad de procesamiento de la imagen electrónica hace que los artistas e ingenieros se propongan la elaboración de dispositivos capaces de generar y manipular imagen electrónica.

A finales de los 60 y principios de los 70 del siglo pasado, tomando como referencia el sintetizador de audio analógico Moog[34] (1963), hubo una eclosión de dispositivos técnicos capaces de generar imagen electrónica sintética. Entre 1969 y 1971 Nam June Paik y el ingeniero Shuya Abe construyen el primer sintetizador de video, denominado *Paik-Abe Video Synthesizer* (ver figura 2.1.24). Este

[34] El sintetizador analógico Moog fue desarrollado por Robert Moog en 1964.

dispositivo era capaz de realizar modificaciones a siete fuentes de video de forma simultáneamente y en tiempo real. *Video Tape Study No.3* y *Electronic Moon No.2* realizado junto a Jud Yalkut, son algunos de los trabajos característicos de Paik.

Figura 2.1.24. Paik-Abe Video Synthesizer.

En 1968, el artista Stephen Beck construye el primer visualizador electrónico, el *Direct Video Zero* (ver figura 2.1.25), este dispositivo a partir del sonido generado por un sintetizador producía imagen sintética. Desde el punto de vista tecnológico es un paso importante en la búsqueda de relaciones entre la imagen y el sonido, puesto que partía de un sonido electrónico y este era visualizado electrónicamente. Debido a los pobres resultados obtenidos pronto

fue desechado y Beck centra sus esfuerzos en sintetizar imagen sin influencia del sonido, o al menos de forma tan directa. Este trabajo representa para nuestra investigación uno de los referentes más importantes ya que supone un marco similar que el que hemos desarrollado pero a nivel electrónico en vez de digital, puesto que genera imagen a partir del sonido, no como metáfora o como parámetro de manipulación de propiedades visuales sino directamente muestra el sonido.

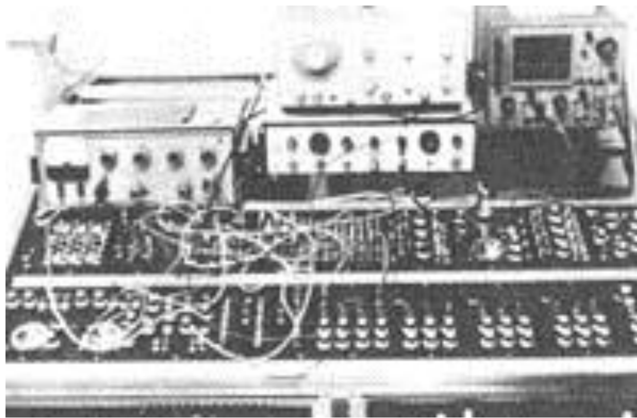


Figura 2.1.25. *Direct Video Zero.* Stephen Beck

En 1972 el profesor de la Universidad de Illinois Dan Sandin desarrolla el sintetizador *Image Processor* (ver figura 2.1.26). Además de sus capacidades de manipulación de la imagen, el dispositivo añadía funcionalidades para ser utilizado en actuaciones en directo. Es interesante remarcar que se trata de uno de los primeros dispositivos de lo que hoy se denomina hardware libre, puesto que nunca quiso comercializarlo y sus especificaciones las entregaba a quién se lo pidiera, en palabras de Sandin:

“Credo de la distribución: El Procesador de Imagen puede ser copiado sin cargo alguno por las personas, pero no por las

instituciones comerciales. Las instituciones comerciales tendrán que negociar los permisos para copiarlo. Yo pienso que la cultura tiene que aprender a usar las máquinas para su crecimiento personal, estético, religioso, intuitivo, comprensivo, exploratorio. El desarrollo de máquinas como el Procesador de Imagen es parte de esta revolución. A mí me paga el estado, por lo menos en parte, para producir y comunicar esta información; así que lo hago" (Furlong 1983)

Figura 2.1.26. Sintetizador. *Image Processor,* Dan Sandin[35].

En 1973 Steve Rutt y Bill Etra, basándose en el sintetizador de Paik terminan el prototipo del *Rutt/Etra Video Synthesizer*. Aunque solo trabajaba con imágenes en blanco y negro proporcionaba una gran cantidad de prestaciones. El Rutt/Etra fue el sintetizador habitualmente usado por Woody y Steina Vasulka.

En 1976 el diseñador de videojuegos Robert Brown construye el primer visualizador electrónico comercial, el *Atari Video Music*[36] (ver

[35] Fotografía de Alain Depocas.

[36] Atari, Inc fue una empresa estadunidense de video juegos y ordenadores personales, fundada en 1972 por Nolan Bushnell y Ted Dabney. Sus ordenadores

figura 2.1.27). Era un equipo que se conectaba con el amplificador del equipo de sonido del cual tomaba la señal de sonido y su salida se conectaba con el televisor. Las imágenes resultantes eran figuras geométricas sincronizadas con la música (ver figura 2.1.27). Realmente fue un dispositivo extraño y muy adelantado a su época. Prácticamente al año de su comercialización Atari decidió retirarlo del mercado.

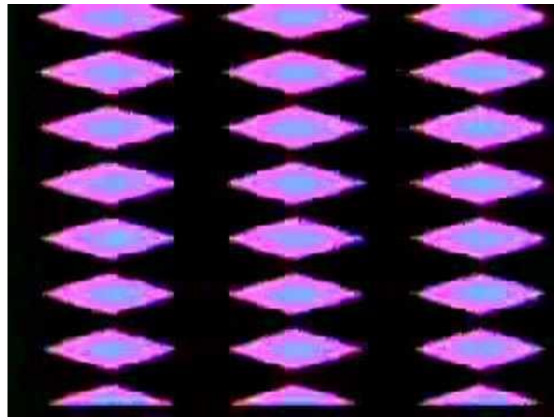


Figura 2.1.27. *Atari Video Music* y fotograma de una visualización.

Woody y Steina Vasulka comenzaron a investigar con la acústica electrónica, estudiando su repercusión en el televisor. Su trabajo es

---

personales fueron un referente en el tratamiento de sonido digital a principios de los 80s del siglo XX.

uno de los más distinguidos en tecnología electrónica y se basa en la utilización de la trama, la manipulación del tiempo de barrido o la alteración de las variaciones de amplitud y frecuencia de las señales. El video *NoiseFields* de 1974 representa un claro ejemplo de sus experimentaciones, realizado con el videosintetizador Rutt/Etra (ver figura 2.1.28).

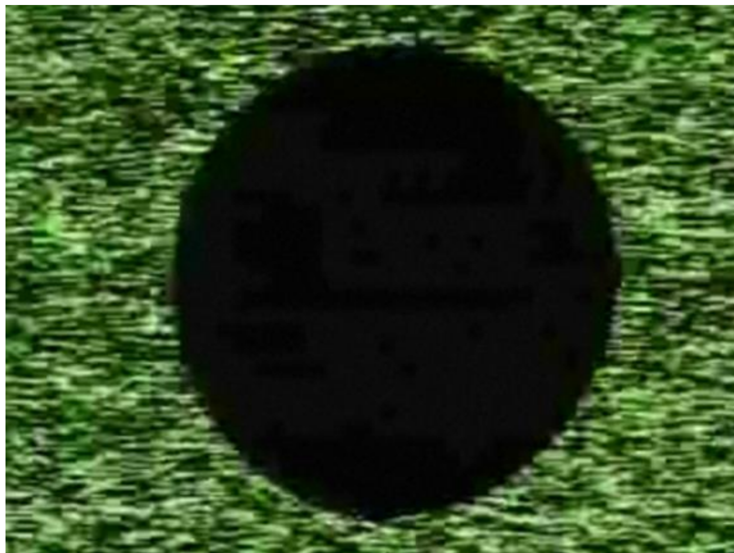


Figura 2.1.28. Fotograma de *NoiseFiels*, Vasulka (1974).

Gary Hill dedica prácticamente todo su trabajo a investigar en las relaciones del lenguaje y la imagen. Recogiendo la herencia dejada por cineastas como Einsenstein[37], Fritz Lang o MacLaren que ya introdujeron la palabra en sus películas, Hill establece las bases de los poemas audiovisuales. En los trabajos de Hill, las imágenes y las palabras se acercan y se estructuran con la misma esencia que la poesía dicta los entresijos de los sentimientos. Claramente podemos distinguir dos líneas diferenciadas en sus obras. La primera basada

[37] Los trabajo teóricos de Eisenstein (1942, 1947) han influyeron de una forma definitiva en toda la corriente de video artistas.

en la lingüística electrónica, en la que el sonido y la palabra se fusionan con la imagen. En *Electronic Linguistic* de 1977 Hill plantea las relaciones orgánicas y estructurales entre el sonido y la imagen (ver figura 2.1.29). Las imágenes aparecen como visualizaciones de sonidos generados electrónicamente, donde inicialmente se presentan unas pulsaciones de pixels y poco a poco las formas van inundando toda la imagen. La segunda utiliza un discurso más cotidiano procesando imágenes del mundo real. En *Bits* (1977), Hill plantea una pintura electrónica creada transformando la grabación de un paisaje por medio de un sintetizador de video (ver Figura 2.1.30).

Figura 2.1.29. Fotograma de *Electronic Linguistic*, Hill (1977).

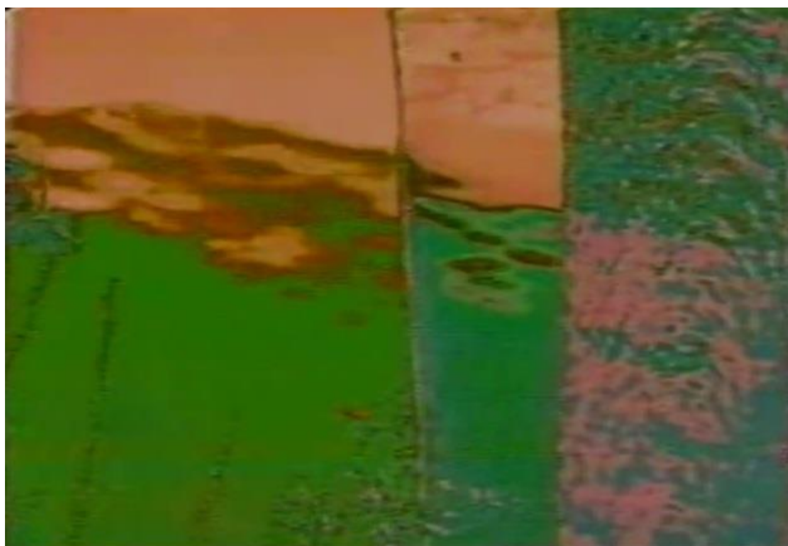


Figura 2.1.30. Fotograma de *Bits*, Hill (1977).

Por último de esta etapa nos gustaría resaltar las investigaciones del artista polaco Zbigniew Rybczynski cuyo trabajo ha estado siempre en íntima relación son el sonido, siendo además un claro ejemplo de cómo las estéticas del videoarte se llevaron al campo del videoclip. Uno de los trabajos más interesantes de Rybczynski en lo que concierne a procesamiento de imagen es *Fourth Dimension* realizado en 1988 (ver figura 2.1.31).

Figura 2.1.31. Fotograma de *Fourth Dimension*, Rybczynski (1988).

De este periodo podemos destacar la importancia que su supuso la generación automática de imágenes a través de los videosintetizadores, incluso se realizan los primeros experimentos de ver el sonido. Todas las técnicas de manipulación de sonido electrónico son llevadas también a la imagen. Se retoma el concepto de directo y de tiempo real, el videosintetizador puede crear imagen y manipularse en tiempo real. La imagen y el sonido electrónico hablan el mismo lenguaje, nunca los dos medios habían estado tan cerca. El sonido se ve abstracto y la imagen se sincroniza de forma automática con el sonido. De nuevo todas estas tendencias llegan hasta nuestros días.

Para una profundización en el campo de la videocreación se puede consultar el trabajo que realizó Televisión Española con la serie “El Arte del Video” (Pérez 1991), así como (Rosseti 2011) y (Benavides 2010). Para un revisión del videoarte en España se puede consultar (Sedeño 2011).

2.2.3 VideoClip

Hacia finales de los setenta y herederos de las estéticas del cine experimental y videoarte, surge dentro de la cultura popular el género del Videoclip Musical, como un producto comercial orientado principalmente a la promoción de grupos musicales (Sánchez 2009).

El video musical se popularizará en los años 80 a través de la cadena de televisión MTV[38]. El video de la canción "*Video killed the radio star*" del grupo inglés The Buggles realizado por Russell Mulcahy fue el primero que emitió MTV el 1 de agosto de 1981.

Un aspecto interesante que aporta este género en el campo de la visualización es que habitualmente las imágenes se crean después de la música y en función de esta. Además con el videoclip se popularizan los nuevos lenguajes surgidos del cine experimental y el video arte, hasta el momento patrimonio exclusivo del campo del arte.

En (Martin 2011) se propone una clasificación del género que abraca desde estilos que simulan coreografías, al estilo de los musicales cinematográficos, estilos basados en la representación televisiva o estilos experimentales, por citar algunas de sus tendencias. Además cada género musical establecerá con el tiempo unos cánones característicos. Una característica que marca la estética del videoclip es la ausencia de reglas fijas y la apropiación de todos los formatos, lenguajes y modelos audiovisuales, importados del cine, televisión, arte o animación. La libertad de interpretación, junto con la voluntad de experimentar y fusionar estéticas y estilos hace que el género sea uno de los más creativos e innovadores, aunque siempre bajo del concepto promocional de la música que visualiza.

[38] MTV es un acrónico de *Music Television*.

Desde su aparición ha atraído a cineastas, artistas y diseñadores de reconocido prestigio que han realizado propuestas muy interesantes, Laurie Anderson, John Sanborn, Andy Warhol, Martin Scorsese, Spike Jonze, Michel Gondry, son algunos ejemplos. También su estética ha influenciado en el cine como con el trabajo de Tom Tykwer en su película “Corre Lola corre” de 1998 (ver figura 2.1.32).

Figura 2.1.32. Cartel de “Corre Lola corre”, Tykwer (1998).

La idea de visualizar la música se gesta como hemos visto en las vanguardias de principios de siglo y es con el videoclip cuando se produce su popularización, pero mientras en los trabajos de Música Visual y Videoarte se buscan las relaciones directas entre el sonido y la imagen, en el videoclip se busca principalmente realizar un promoción del grupo o interprete musical. Si tomamos como marco de referencia el videoclip experimental en el que no aparece el grupo y la estética es abstracta, nos aproximamos a la Música Visual y al Videoarte. La música electrónica y experimental, habitualmente carente de letra, desde sus orígenes ha adoptado una estética cercana al video experimental.

Del videoclip podemos destacar que toma su estética de todas las corrientes audiovisuales, estableciendo una hibridación y un mestizaje entre ellas. Otro aspecto muy importante es la popularización que supone la visualización de la música, con o sin músicos, con el consiguiente enriquecimiento cultural que verá sus frutos a finales del siglo XX y principios del XXI con una gran cantidad de trabajos sin ánimos comerciales. Por último resaltar que el objetivo artístico principal es acompañar visualmente a la música, estableciendo una clara primacía del sonido sobre la imagen.

Para una profundización en el campo de la videoclip se pueden consultar los trabajos de (Sanchez 2009) y (Selva 2012).

2.3 Sistemas digitales

Con la expansión de la tecnología digital se abre un nuevo marco para reformular la herencia analógica y plantear nuevos retos en el campo de visualización. Los sistemas digitales se establecen como nexo y punto de conexión entre las diferentes corrientes surgidas a raíz de las tecnologías analógicas, siendo los conceptos de multimedia, interactividad y tiempo real los encargados de liar y expandir definitivamente al sonido y a la imagen.

Recapitulando las aportaciones históricas revisadas vemos que los instrumentos visuales e instalaciones audiovisuales aportan la interactividad y el tiempo real, la música visual define la estética abstracta como visualización propia del sonido e incorpora las estructuras musicales en la composición visual, el videoarte incorpora el concepto de procesamiento como elemento fundamental en desarrollo creativo y el videoclip aporta fundamentalmente la popularización de la estrecha relación entre el sonido y la imagen. En

este panorama, los sistemas informáticos ejercerán de crisol integrando todos estos elementos, y además propiciarán la aparición de nuevas propuestas como las instalaciones y performances multimedia y el *vjing* o *live cinema*.

El ensayista Paul Virilio estable como quinta máquina de visión al ordenador en su relación con la imagen videográfica, en sus propias palabras:

"*La primera máquina de visión es la cámara obscura. Numerosos filósofos han hablado de ella y siguen hablando. La segunda máquina es la linterna mágica*[39]*, innovada por Athanasius Kircher, el conocido jesuita. La tercera cámara es la cámara fotográfica y cinematográfica pues creo que es obligatorio vincular ambas; si no se hubiera innovado la fotografía instantánea, no se hubiera podido crear el fotograma que permite llegar a la secuencia cinematográfica, fílmica. Así pues, la tercera máquina es la cámara fotográfica y la cámara de registro cinematográfico. La cuarta máquina es efectivamente, el video, la televisión si se prefiere. Y la quinta máquina es el acoplamiento del ordenador y el video."* (Virilio 1989)[40]

Esta nueva etapa plantea un nuevo paradigma tecnológico, la era digital, no se trata de un nuevo formato sino de una nueva visión de la tecnología tan importante como lo fue la revolución industrial a principios del siglo pasado. No solo revoluciona el campo de imagen y sonido, sino prácticamente en mayor o menor medida todas las áreas de conocimiento humanista y sobre todo científico.

La manipulación de la imagen y del sonido digital se realiza a través del lenguaje. El lenguaje de los ordenadores, va mucho más allá de

[39] La linterna mágica es un aparato precursor de cinematógrafo, diseñado entre 1646 y 1671.
[40] Texto citado explícitamente también en (Pérez 1991)

las reglas de uso de las consolas de las mesas de mezcla, de los sintetizadores o de los videosintetizadores, mediante una sintaxis y unas reglas gramaticales muy sencillas proporciona una capacidad de manipulación tal, que todavía hoy en día no se vislumbran los límites de sus posibilidades.

Sin embargo, la estrategia que se ha seguido para introducir los sistemas informáticos en la sociedad ha sido mediante interfaces gráficas sencillas y amigables. De la misma forma que las palancas de las interfaces físicas de los videosintetizadores ocultaban la complejidad electrónica de los circuitos, las actuales interfaces gráficas de las aplicaciones esconden los detalles del lenguaje digital, proporcionando una funcionalidad reglada por el diseñador de la aplicación. Estas aplicaciones en la mayoría de los casos son suficientes, salvo que se persiga un objetivo más innovador y experimental en el que es necesario introducirse en las entrañas de la máquina a través de su propio lenguaje.

A continuación repasamos algunas de las aplicaciones y lenguajes de programación para el tratamiento de sonido e imagen digital.

2.3.1 Lenguajes de Programación y Aplicaciones

El verdadero artífice de las conexiones entre sonido e imagen, como hoy las conocemos son los lenguajes de programación. Si con la imagen y el sonido electrónicos ya establecíamos un entorno común en el que los dos medios se encontraban, produciéndose en artistas como Nam Juke Paik y los Vasulka una comunicación creativa con el medio a través de sus posibilidades de manipulación, con la llegada de los ordenadores y sus lenguajes de manipulación nos encontramos con un gran abanico de nuevas posibilidades. Posibilidades que hoy en día todavía estamos experimentando.

Con la imagen y el sonido electrónico se establece el modelo a seguir; los medios se capturan del mundo real o se generan de forma artificial, para posteriormente ser procesadores y obtener el resultado final. Los videosintetizadores, mezcladores y sintetizadores de sonido se manipulan a través de un lenguaje relativamente sencillo. El planteamiento de una pieza depende por lo tanto del lenguaje natural en su relación con el mundo exterior, del lenguaje de síntesis y del lenguaje de manipulación y procesamiento.

La adaptación de este modelo al campo informático amplía las posibilidades por un lado y por el otro simplifica el uso de lenguajes, lo que tenemos es un lenguaje único a nivel de síntesis y manipulación que es el lenguaje de programación. Por encima del lenguaje de programación de bajo nivel lo que nos encontramos habitualmente son capas de software, aplicaciones que tratan de ocultar la complejidad del lenguaje y ofrecen un entorno interactivo y amigable.

Para establecer conexiones entre el sonido y la imagen en el campo de la informática tenemos gran cantidad de lenguajes de alto nivel, así como gran cantidad de aplicaciones. Como decíamos, la aplicación nos proporciona un pequeño subconjunto de todas las funcionalidades que podríamos desarrollar con la programación, pero para un usuario no informático en la mayoría de ocasiones es suficiente.

A grandes rasgos podemos decir que entre finales de los años 80 y principios de los 90, todos los grandes lenguajes de programación van introduciendo paulatinamente funcionalidades para manipular imagen y sonido. En la figura 2.1.32a se presenta un gráfico que establece un ranking de popularidad de los lenguajes de programación realizado en 2015. Llama la atención la cantidad de

lenguajes diferentes que existen. Desde hace unos cuantos años los lenguajes más utilizados son: Java, C++, C#, JavaScript, PHP, Ruby y Python.

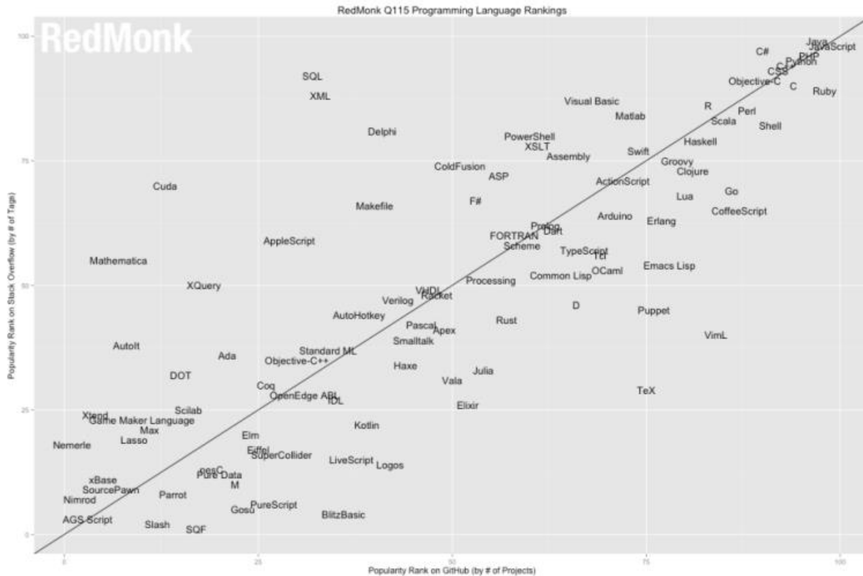


Figura 2.1.32a. Ranking de lenguajes en 2015.[41]

Por un lado tenemos los lenguajes clásicos orientados a objeto como Java, C++ o C# y luego los lenguajes scripts, pensados para una ejecución más rápida incluso con capacidades de edición en tiempo real, como JavaScript, Python o Ruby. Todos estos lenguajes son de uso generalista.

Si nos centramos en los lenguajes más específicos para el tratamiento del sonido y la imagen lo que tenemos es por un lado una familia de lenguajes gráficos basados en la manipulación gráfica de

[41] Ranking publicado en el sitio web de redMonk y realizado por Stephen O'Grady. http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/ (consultado 12/12/15).

flujos de datos, como Pure Data, Max/Msp[42] o OpenMusic[43]. Y por otro, entornos de desarrollo[44] orientados al tratamiento de la imagen y el sonido, que persiguen simplificar la tarea de programación, proporcionando un lenguaje adaptado y herramientas específicas para su manipulación, como son Processing[45], GAmuza, Fluxus[46], Zajal[47] o SuperCollider[48]. En ambas aproximaciones se pretende proporcionar un entorno para trabajar en tiempo real.

Pure Data[49] es un lenguaje de programación desarrollado por Miller Puckette durante los años 90 para la creación de música por ordenador interactiva y obras multimedia (ver figura 2.1.33). Se trata de un proyecto de código abierto de manera que la comunidad internacional de programadores es responsable de sus extensiones. Pd es muy similar al lenguaje que originalmente realizó Puckette, el Max/MSP.

---

[42] La página principal del lenguaje es: https://cycling74.com/ (consultado 15/12/15).
[43] La página principal del lenguaje es: http://repmus.ircam.fr/openmusic/home (consultado 15/12/15).
[44] En el campo de la programación a los entornos para el desarrollo de programas se les denomina IDE, que son las siglas de *Integrated Development Environment*.
[45] La página principal del lenguaje es: https://processing.org/ (consultado 15/12/15).
[46] La página principal del lenguaje es: http://www.pawfal.org/fluxus/ (consultado 15/12/15).
[47] La página principal del lenguaje es: http://eyebeam.org/projects/zajal (consultado 15/12/15).
[48] La página principal es: http://supercollider.github.io/ (consultado 15/12/15).
[49] La página principal del lenguaje es: https://puredata.info/ (consultado 15/12/15)

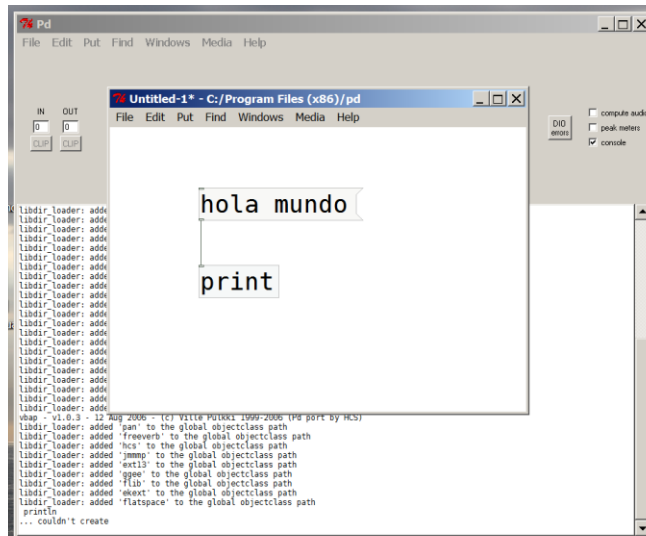


Figura 2.1.33. Un patch de Pure Data.[50]

Gamuza[51] es un entorno desarrollado por Emanuelle Maza que recoge y coordina de forma particular los lenguajes Lua y C++, y las librerías openFrameworks[52] y openCV[53], además de incorporar otras librerías propias y en general todas las librerías de Lua y openFrameworks. Su objetivo principal es facilitar el desarrollo de programas orientados a sonido e imagen para artistas en el campo de la programación creativa (ver figura 2.1.34).

[50] Ranking publicado en el sitio web de redMonk y realizado por Stephen O'Grady. http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/ (consultado 12/12/15).

[51] La página principal del lenguaje es: http://gamuza.d3cod3.org/ (consultado 15/12/15).

[52] OpenFrameworks es una herramienta de código abierto diseñada por Zachary Lieberman para el desarrollo de código de programación creativo.

[53] OpenCV es una biblioteca libre de visión artificial originariamente desarrollada por la empresa Intel.

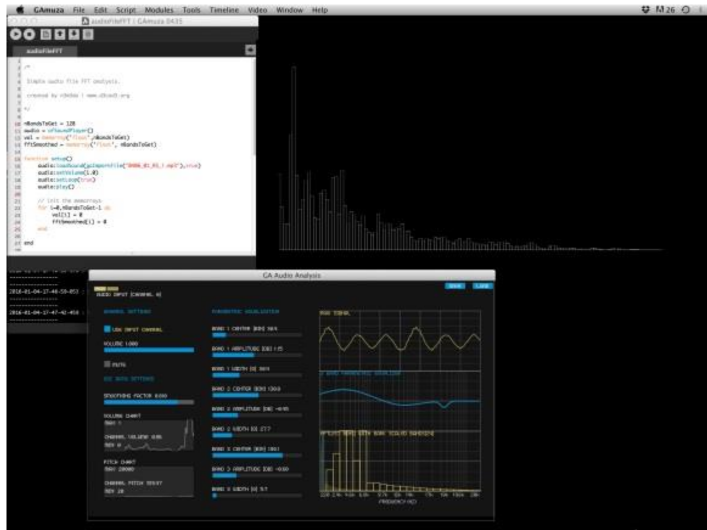


Figura 2.1.34. El entorno de desarrollo de Gamuza.

Para terminar este breve recurrido por los lenguajes de programación tenemos los lenguajes generalistas como Java. Java es un lenguaje inicialmente desarrollado por James Gosling de Sun Microsystem en 1995, gestado en el boom de la programación orientada a objetos. La posibilidad de manipular imagen está incluida desde el inicio, y el tratamiento del sonido se introduce en la versión 1.3 de 2000.

De forma simultánea a los lenguajes de programación se desarrollaron y se siguen desarrollando aplicaciones para la manipulación de imagen y sonido. Debido a que el número de aplicaciones es muy elevado, nos centraremos únicamente en algunas especializadas en visualización del sonido.

Como heredero directo del Atari Video Music, en 1985 la empresa Infinite Software lanza al mercado *Sound to Light Generator* para el ZX Spectrum. Este software representa el inicio de una serie de programas orientados a visualizar música, generando imágenes

geométricas cuya forma y disposición varía a lo largo del tiempo en función de la intensidad del sonido y su frecuencia. A finales del siglo XX se produce el auge de este tipo de herramientas informáticas cuyos modelos de visualización se incorporaban como plugins dentro de los reproductores de sonido más extendidos. Por ejemplo el popular *G-Force* de Andy O'Meara, que fue diseñado para incorporarse con el reproductor *iTunes* en el año 2000. En la actualidad existe un gran número de programas de visualización con estas características.

De la misma forma que el disc jockey (DJ) selecciona y mezcla piezas musicales, crea repeticiones a través de pequeñas muestras de sonido y todo lo procesa mediante efectos en tiempo real, el video jockey o vjing realiza acciones similares pero con la imagen. Paralelamente al auge que ha tenido esta nueva figura en los clubs de baile, han aparecido muchas aplicaciones orientadas a la realización de estas sesiones. Las características principales de este tipo de software de visualización[54] son: edición en tiempo real, posibilidad de establecer sincronización directa con el sonido, facilidad de inserción de efectos, posibilidad de usar imagen sintética y facilidad para lanzar imágenes en bucle. En esta línea podemos resaltar programas como Resolume[55], Modul8[56] o Arkaos[57], siendo el Modul8 uno de los más utilizados por los vjing, puesto que se adapta continuamente a los nuevos protocolos y tendencias informáticas,

[54] Una recopilación de la mayoría de aplicaciones que existen actualmente sobre software de visualización la podemos encontrar en la página web: http://www.audiovisualizers.com/toolshak/vjprgpix/softmain.htm (visitado 22/12/15).

[55] La página principal de la aplicación es: https://resolume.com/download/ (visitado 22/12/15)

[56] La página principal de la aplicación es: http://www.modul8.ch/ (visitado 22/12/15).

[57] La página principal de la aplicación es: http://www.arkaos.net/ (visitado 22/12/15).

pese a ser de pago (ver figura 2.1.35). Quizá el único inconveniente que presenta es que solamente hay versión para el sistema operativo de los ordenadores de la marca Apple.

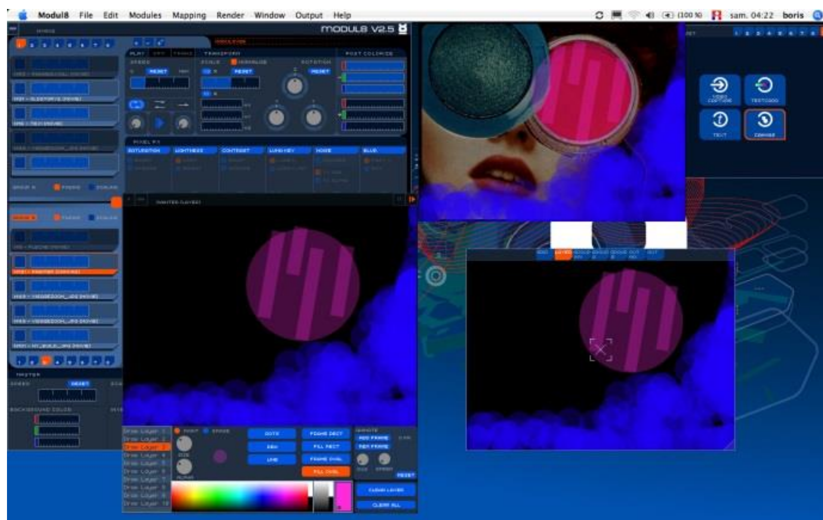


Figura 2.1.35. El entorno de desarrollo de Modul8.

Una variante del vjing o más bien otro punto de vista es el live cinema, se trata de desarrollar visualización en tiempo real pero con una mayor vinculación en la campo artístico. No obstante son muchos los autores que no establecen una clara distinción entre el vjing y el live cinema, salvo por el hecho de estar el vjing más arraigado a la cultura del club.

De este apartado podemos resaltar como conclusiones que la era digital es sin duda el mejor momento histórico para comenzar a estudiar en profundidad las intrincadas relaciones entre el sonido y la imagen, a través de sus herramientas los lenguajes de programación y las aplicaciones especializadas. Mediante los lenguajes de programación y la mayoría de las aplicaciones especializadas se pueden realizar generación de imagen sintética, sincronización entre

propiedades del sonido y la imagen, análisis de propiedades del sonido, procesamiento del sonido y de la imagen, y comunicación con sensores, actuadores e internet, y todo ello en tiempo real.

2.3.2 Interactividad y tiempo real

En este apartado vamos a recoger algunos trabajos que más directamente han influido en el desarrollo de la investigación, sin aplicar más criterio que el personal. Una revisión muy completa del campo se puede consultar en el archivo web *visualmusicarchive*, que ya hemos comentado al inicio del capítulo, donde se recoge una espléndida recopilación de los trabajos más actuales del campo. También se pueden encontrar en (Tez 2009) y (Phoenix 2002) una recopilación de trabajos experimentales de visualización muy interesantes.

Sin duda una de las mayores aportaciones que se ha realizado en el campo de la visualización del sonido con la llegada de los sistemas informáticos ha sido la posibilidad de introducir interactividad y tiempo real en el proceso de visualización. Interactividad entendida como la capacidad de que un agente externo influya en el desarrollo de la visualización en tiempo real. Siguiendo esta línea de investigación desde finales de los 80 hasta nuestros días encontramos gran cantidad de propuestas de artistas como: Golan Levi, Zachary Lieberman, Robert Henke, Alva Noto, Ryoji Ikeda, Karl Kliem, Kalus Obermaier, Craig Allan, Robert Hodgin, Robin Fox, Laboratorio de Luz, colectivo PDP11, Markus Heckman, Daniel Palacios, Fredik Olofsson entre otros.

Carsten Nicolai, conocido en su actividad musical como Alva Noto, es un artista sonoro y visual que también ha trabajado en grupos como

Signal (junto a Frank Bretschneider, cuyo alias es Komet y Olaf Bender, también conocido como Byetone) y Cyclo (con Ryoji Ikeda), así como ha realizado muchas colaboraciones con otros músicos y artistas visuales. En los trabajos en los que se produce una gran convergencia de medios y tecnologías es habitual que se desarrolle en grupos o colectivos, debido a la dificultad que conlleva la resolución de la pieza. *Trioon I* de 2003 es una pieza de visualización del sonido donde se establece una relación directa entre los sonidos y las imágenes, la música es de Alva Noto y Ryuichi Sakamoto y los visuales de Kalrl Kliem (ver figura 2.1.36). Los tres artistas en 2006 realizaron un tour a nivel internacional llamado *Insen Tout Part*. Otro trabajo interesante es el que realiza junto con Ryuichi Sakamoto y la orquesta de cámara Esembre Modern, se trata de una serie de conciertos en los que de nuevo las relaciones entre el sonido y las imágenes interactúan de forma programada.

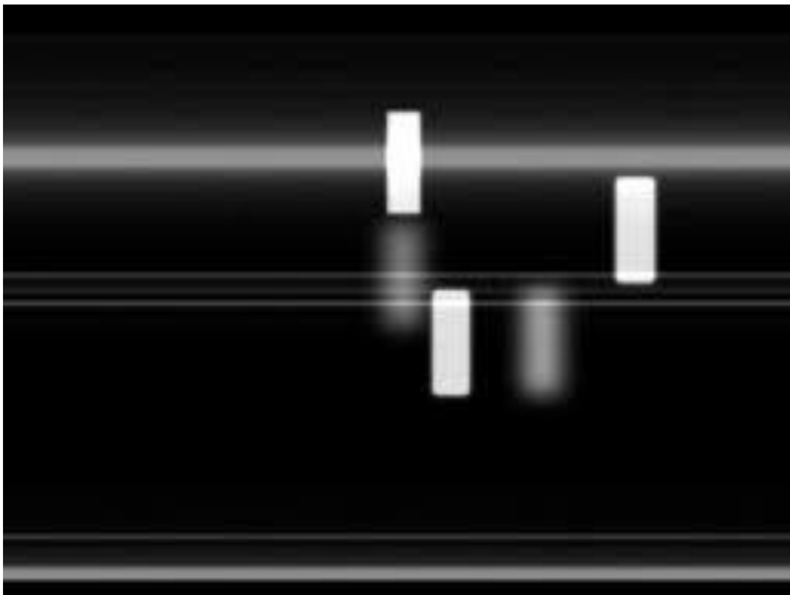


Figura 2.1.36. Fotograma de *Trioon I*, 2003.

Karl Kliem, también conocido como Dienststelle es un artista multimedia que ha desarrollado conciertos visuales junto a muchos músicos como Alva Noto, Ryuichi Sakamoto, Mouse on Mars[58] o Thomas Brinkmann. Junto con estos músicos ha realizado innumerables conciertos, así como instalaciones multimedia. Prácticamente todos los trabajos de Kliem son generados algorítmicamente y por lo tanto aptos para su presentación en directo y en tiempo real. *Rotationsstudien Sequenzen 20-5* es una instalación que realiza junto a Jan St. Werner, formada por una serie de composiciones audiovisuales, donde cada uno de los visitantes escucha y ve una pieza diferente generada mediante la ejecución de un programa (ver figura 2.1.37).

Figura 2.1.37. Fotograma de *Rotationsstudien Sequenzen 20-5*, 2015.

Ryoji Ikeda es otro de los artistas actuales que desarrolla su trabajo entre lo visual y lo sonoro, ha realizado innumerables instalaciones multimedia y ha colaborado con artistas como Carsten Nicolai, William

[58] Mouse on Marses un dúo alemán compuesto por Jan St. Werner y Andi Toma, desarrollando su música dentro del género de la electrónica.

Forsythe, Horoshi Sugimoto, Toyo Ito o el colectivo Dumb Type[59]. *Test pattern* de 2008 es un sistema que convierte datos (sonido, texto, imágenes o películas) en patrones visuales en blanco y negro, formando códigos de barras. Este proyecto lo realiza junto al programador Tomonaga Tokuyama, y se ha presentado en diferentes formatos como instalación o concierto (ver figura 2.1.38 y 39).

Figura 2.1.38. Performance audiovisual *Test Pattern*, Ikeda, 2008[60].

[59] Dump Type es un colectivo formado por miembros de diversas disciplinas como son artes visuales, sonido, teatro, arquitectura y informática. El colectivo se fundó en 1984.

[60] Fotografía de Liz Hingley.

Figura 2.1.39. Instalación *Test Pattern* en Time Square, Ikeda[61], 2013.

Laboratorio de luz[62], es un grupo de investigación que desde 1990 viene desarrollando proyectos multidisciplinares dentro del marco institucional de la Universidad Politécnica de Valencia. *Modulador de luz* es una instalación multimedia que explora las relaciones y comportamientos entre luz y sonido en función de las acciones sonoras que los usuarios realicen. La instalación se presenta como un espacio escénico vacío, a la espera del acto del habla; hablar por los micrófonos para atraer la luz teatral. Pero, por su programación, el comportamiento de las luces en relación a lo parlante, lo sonoro y lo musical propicia juegos imprevistos; distintos roles y actitudes de la luz: a veces de forma rápida e intensa, otras dubitativa o tímidamente se dirigen hacia un micrófono u otro, por lo que no siempre se alcanza la iluminación deseada por todos los espectadores-actores (ver figura 2.1.40)

[61] Fotografía de Ryoji Ikeda.

[62] Actualmente el Laboratorio de Luz está formado por: Amparo Carbonell, Salomé Cuesta, Maribel Doménech, Dolores Furió , Carlos García Miragall, Trinidad Gracia, Josefa López, Moisés Mañas, Emilio Martínez, María José Martinez de Pisón, Emanuele Mazza, Lorena Rodriguez, Francisco Sanmartín y Laura Silvestre.

Figura 2.1.40. Instalación Modulador de Luz, laboluz, 2008.

El colectivo PDP11[63], es un grupo multidisciplinar cuyo objetivo principal es establecer una plataforma abierta a la investigación del sonido, la imagen y el arte culinario, en su lado más cercano a la tecnología digital. El proyecto *Root 1.1* se presenta en formato de *performance*, donde los ordenadores, los músicos, lo visual y los espectadores se mezclan en una transformación continua sonora y visual. Por un lado se plantean sonidos ejecutados directamente por los músicos con guitarra eléctrica y acústica, teclados, piezo eléctricos y máquina de escribir, sonidos generados por la ejecución de los programas del entorno Pure Data y sonidos generados por los asistentes que son grabados y manipulados, y por otro imágenes de video recogidas en la sala e imágenes generadas en función de los diferentes sonidos que se están generando. *Root 1.1* plantea una reflexión sobre el origen del sonido y la imagen, estableciendo una similitud con el proceso de madurez personal ante el abanico de

[63] El colectivo PDP11 actualmente está formado por Deco Nascimento, David Jengibre, Francisco Sanmartin y Carlos García Miragall.

posibilidades, elecciones y dudas que genera el entorno actual, caracterizado por su complejidad tecnológica (ver figura 2.1.41).

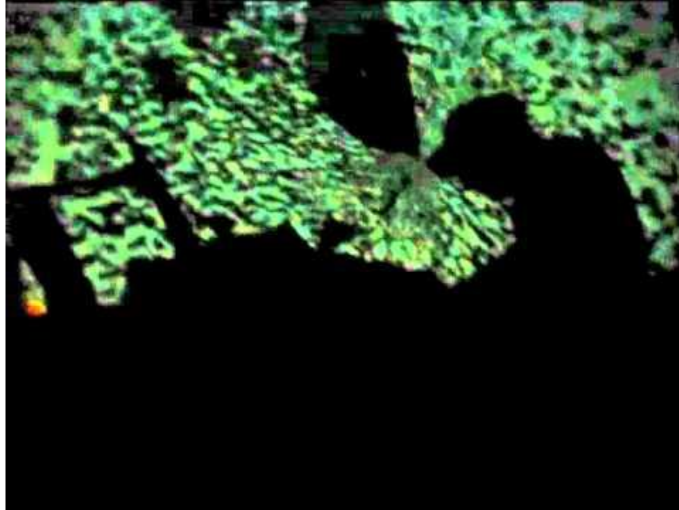


Figura 2.1.41. Performance Root 1.1, PDP11, 2011.

Monolake es Robert Henke un artista multidisciplinar que ha desarrollado su trabajo alrededor de la visualización del sonido, mediante el uso programado del láser. Una de sus últimas *performance* multimedia es *Deep Web*, desarrollada junto a Christopher Bauder y Michel Sollinger (ver figura 2.1.42).

Figura 2.1.42. Performance Deep Web, Monolake, 2015.

Golan Levin es un artista multidisciplinar, compositor e ingeniero interesado en el diseño de sistemas para la manipulación y creación de imagen a partir del sonido. Desde 2002 ha colaborado estrechamente con Zachary Lieberman en diversos proyectos. A través de instalaciones interactivas y performances Levin establece conexiones personales con la tecnología digital y pone de relieve nuestra relación con las máquinas. El proyecto *Messa di Voice* de 2003 es una colaboración con Zachary Lieberman, Jaap Blonk y Joan La Barbara, donde a través del sonido se van generando unos gráficos. El proyecto aborda temas de comunicación abstracta y de relaciones sinestésicas, usando el lenguaje del comic junto con esquemas de escritura en un contexto de narración audiovisual sofisticada, humorística y virtuosa. El programa informático transforma cada matiz vocal en gráficos complejos y muy expresivos. *Messa di Voce* está diseñado para provocar preguntas sobre el significado y los efectos de los sonidos del habla, los actos del habla, y el medio ambiente de inmersión de la lengua (ver figura 2.1.43).

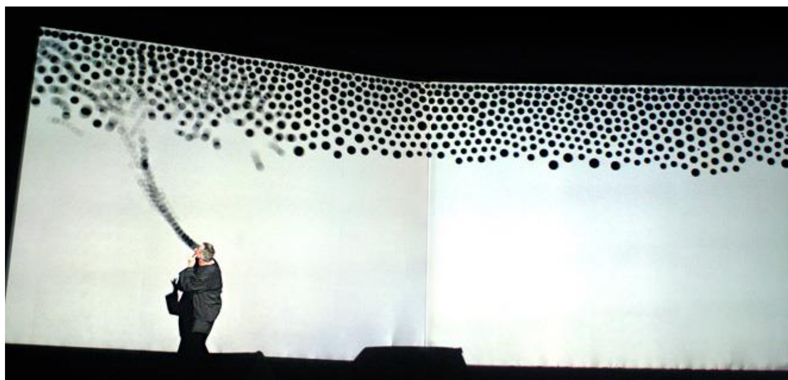


Figura 2.1.43. Performance *Messa di Voce*, Levin (2003).

Una de las características principales que encontramos en las tendencias actuales es justamente el mestizaje que se está produciendo a raíz de las colaboraciones que propician que el arte en general este tomando una nueva dirección. La figura del artista ensimismado e inserto en su proceso de creación está desapareciendo para dar paso al desarrollo de ideas mediante equipos de trabajo interdisciplinares, donde el proyecto va tomando forma a través de la interacción creativa de todos sus componentes. Como dato anecdótico nos gustaría comentar que este apartado lo hemos comenzado hablando de artistas, a lo sumo ligados temporalmente a alguno de los movimientos vanguardistas, hemos continuado con las primeras colaboraciones entre técnicos y artistas, en los inicios del videoarte, y hemos terminado en la era digital hablando de artistas y sus colaboraciones, de colectivos y de grupos. Estamos por lo tanto inmersos en una época de fusión no solo del sonido y de la imagen, sino de fusión entre el sonido, la imagen, la arquitectura, el teatro, el arte culinario, la psicología, la matemática ,… Este sin lugar a dudas, es el signo de nuestro tiempo.

2.4 Otros aspectos artísticos y tecnológicos de interés

Por último antes de finalizar este repaso histórico sobre las relaciones entre el sonido y la imagen y siendo conscientes que en aras a presentar un resumen compacto y breve de la materia hemos dejado muchísimos autores y propuestas, queremos recoger algunos aspectos adicionales artísticos y técnicos relacionados con la imagen, el sonido y el análisis de sus propiedades.

Inicialmente nos gustaría comentar que existe una rama muy importante de visualización del sonido, que es la visualización del sonido a través de su vibración sobre cuerpos sólidos. Dentro de esta línea tenemos los trabajos basados en cimática, donde una placa, diafragma o membrana se hace vibrar, colocando sobre ella partículas, pastas o líquidos. La característica principal es que las partículas se distribuyen formando patrones geométricos. La placa de Chladni[64] o el CymaScope[65] son algunos ejemplos. En (Silleras 2015) se realiza un estudio muy detallado sobre este tipo de técnicas aplicadas al campo del arte.

Otro campo muy importante que en este capítulo hemos tratado de forma lateral, es toda la evolución histórica de los gráficos por ordenador, en su vertiente de animación abstracta. En (Russet 1988) se realiza una revisión histórica de la animación experimental desde sus inicios hasta finales de los años 80 del siglo pasado.

El estudio de las propiedades del sonido, tanto físicas como psicoacústicas y perceptuales ha sido y sigue siendo un campo muy

---

[64] Estas figuras sonoras fueron estudiadas por el investigador alemán Ernst Florenz Chladni y publicadas en 1787 en su libro *Entdeckungen über die Theorie des Klanges.*

[65] El CymaScope es un instrumento para hacer visible el sonido desarrollado en 1997 por el físico John Stuart Reid.

activo. Un trabajo clásico en esta línea es (Terhardt 1978) o más recientemente (Zwicker 1999). En trabajos como el de (Slawson 1985) se proponen nuevas propiedades del sonido, denominas color del sonido. La idea de Slawson se basa en que una de las cualidades principales del sonido, que casi nunca se toma en cuenta es el envolvente del espectro de armónicos. De manera que reproduciendo la misma envolvente en sonidos con diferentes armónicos, el resultado tiende a percibirse como similar. Los trabajos de (Allen 1984) y (Masataka 2001) se centran en analizar y recuperar el tempo de una canción. Por otro lado si nos acercamos a la psicología de la percepción del sonido encontramos trabajos clásicos como (Seashore 1938) y (Lage 1972) y más recientes como el de (Scheiver 2000) que estudian las emociones generadas por la música y analizan las técnicas humanas empleadas para detectar estilos musicales, entre otros estudios. Todas estas ideas técnicas son muy interesantes en su aplicación al campo artístico de visualización del sonido.

Otro campo muy activo es aquel que visualiza el sonido buscando analizar su estructura, determinar propiedades físicas, estudiar estilos musicales, en definitiva ver el sonido no con un fin artístico sino con un fin de análisis. En esta línea nos encontramos con trabajos muy interesantes como es el de (Sapp 2001) que centra su investigación en el estudio de la visualización de la armonía. Otro trabajo igualmente interesante es el de (Smith 1997), en este caso siguiendo las ideas de Sapp pero con un planteamiento de la visualización en 3D. El dotar a la visualización de la tercera dimensión o como sería más correcto de la cuarta dimensión nos produce más información y por lo tanto más posibilidades de análisis. En el campo de las visualizaciones artísticas este es sin duda el nuevo paradigma sobre el que se está trabajando más intensamente. Otros trabajos de este campo de investigación son (Wattenberg 2002), (Spiegel 1998),

(Cleveland 1993) y (Yedo 2004) por poner algunos ejemplos. De estos trabajos podemos concluir que cuantos más datos del sonido se puedan extraer de forma automática, muchas más posibilidades de establecer conexiones entre el sonido y la imagen tendremos. La mayoría del software de visualización que actualmente se está desarrollando recoge los resultados de estos trabajos en su implementación.

Otro campo interesante que afecta a las relaciones entre la imagen y el sonido es el desarrollo de lenguajes gráficos orientados a la escritura musical. En trabajos como (Sedes 2004) o (Lemi 2007) encontramos revisiones sobre este tema.

El área que se conoce como *Musical Information Retrieval* (MIR), trata de recuperar información musical de colecciones de ficheros donde fundamentalmente lo que tenemos es música, con el objetivo de realizar catalogaciones, procesamiento, recuperar propiedades, establecer estadísticas etc. En este campo se fusionan disciplinas como musicología, psicología y procesamiento de la señal entre otras. Uno de sus objetivos es definir funciones de similitud con el fin de poder realizar clasificaciones como en los trabajos de (Church 1993) y (Foote 1999). En el desarrollo de este trabajo de investigación se han consultado muchas fuentes en esta línea de investigación buscando conexiones con la visualización del sonido creativa. En trabajos como (Stober 2013) y (Cooper 2002) se realiza un compendio de la mayoría de avances realizados sobre este tema.

También nos gustaría comentar que existen muchos trabajos que realizan el esfuerzo en obtener sonidos a partir de imágenes, a este campo se le suele denominar sonificación de la imagen (Kaper 1999).

El campo de visualización del sonido quedaría dentro de un campo mucho mayor que sería el de Visualización de datos. Actualmente es un campo muy activo, en (Lima 2013) y (Fry 2007) se realizan revisiones más amplias del tema.

“Piezo es una obra diseñada para un chip de sonido de 4 bits/ 12Khz/ 60 segundos con un altavoz piezo […] 00:04-00:05 8”

Robert Henke[66]

3 CONCEPTOS TÉCNICOS DE SONIDO

En este capítulo vamos a presentar los conceptos teóricos del sonido que pensamos son importantes para comprender el método que hemos propuesto. La mayoría de estos conceptos son elementales y se pueden encontrar en textos básicos de sonido. Para una revisión más profunda sobre el sonido digital se puede consultar (Roads 1996), para aspectos del sonido relacionados con su manipulación y síntesis un texto clásico es (Mathews 1969)[67].

La idea del capítulo es exponer un recorrido con un punto de vista técnico, desde que se genera el sonido hasta que éste llega a una aplicación informática. Se trata de realizar un sencillo repaso por las diferentes fases de digitalización del sonido. Este capítulo lo hemos

[66] En (Henke, 2012).

[67] Max Mathews es considerado como uno de los pioneros el campo de la música por ordenador.

dividido en dos partes: digitalización del sonido y comunicación con la tarjeta de sonido.

3.1 Digitalización del sonido

El sonido es un fenómeno perceptual que se produce de forma natural cuando un objeto entra en vibración, y se traduce en una variación de la presión en el aire que se propaga.

Cuando esta vibración es repetitiva decimos que el sonido es periódico. Las vibraciones periódicas generan una sensación de altura o tono. Los instrumentos musicales históricamente se han diseñado buscando justamente la generación de ondas periódicas. Por el contrario los sonidos naturales son casi todos semiperiódicos, es decir, corresponden a una combinación de vibraciones periódicas y no periódicas. Un sonido no periódico es lo que suele denominar ruido.

Estas variaciones se denominan comúnmente formas de onda. La duración de un motivo recurrente de la forma de onda es el período T. El número de veces que dicho período se repite en un segundo nos da su frecuencia fundamental T. La frecuencia fundamental de una onda determina su altura. Si el sonido sólo contiene una frecuencia, decimos que el sonido es sinusoidal puro. Los sonidos naturales contienen diversas frecuencias, además de la fundamental, que forman un sonido complejo y da lugar a la percepción del timbre. A las frecuencias adicionales se les denomina armónicos o parciales, según sea su frecuencia múltiplo o no de la frecuencia fundamental.

Como resumen vemos que el sonido tiene una serie de cualidades principales, muchas de ellas relacionadas entre sí. Tendremos intensidad, frecuencia fundamental o altura, armónicos y parciales que representan el timbre, duración y reverberación que nos informa

del espacio donde se produce. Todas estas cualidades además varían a lo largo del tiempo.

El primer paso para realizar algún tratamiento sobre el sonido es transformar esa variación de la presión del aire en una señal de voltaje variable en el tiempo. Esta señal analógica continua puede grabarse en una cinta magnética mediante tecnología electromagnética o se puede volver a transformar y convertirse en una señal digital para almacenarse en un sistema informático. A este último proceso es al que se denomina digitalización de la señal analógica.

Los ordenadores son máquinas digitales y sus operaciones se basan en matemáticas discretas, por lo que los cálculos deben trabajar con números finitos y exactos. El sonido en este contexto debe ser representado con números y puesto que el ordenador internamente trabaja con números binarios, esos números tendrán que almacenarse finalmente en forma binaria, es decir con combinaciones de 0 y 1. En el sentido contrario, las señales digitales deben ser convertidas a un formato analógico para ser escuchadas. De manera que el ordenador tiene que tener dos tipos de conversores de datos: convertidor analógico a digital (A/D) y digital a analógico (D/A) (ver figura 3.2.1). Estas transformaciones o conversiones las realiza la tarjeta de sonido[68].

[68] Cuando nos referimos a los conversores, estamos haciendo referencia a la tarjeta de sonido.

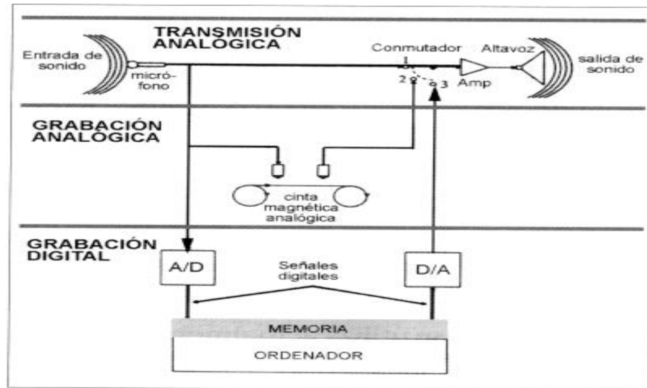


Figura 3.2.1. Esquema de conversión A/D y D/A

Para generar una señal digital a partir de una señal analógica, tendremos que seguir los siguientes pasos: muestreo, cuanficación y codificación (ver figura 3.2.2).

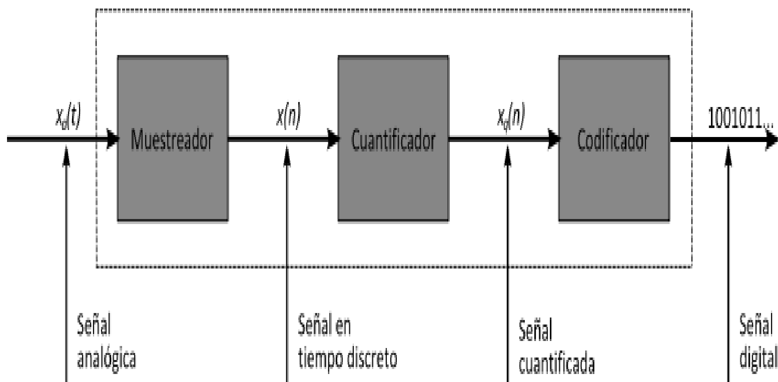


Figura 3.2.2. Conversor A/D

3.1.1 Muestreo

La primera fase para realizar la conversión es la de muestrear, que consiste en tomar muestras del sonido a intervalos regulares de tiempo. Esto matemáticamente lo podemos expresar de la siguiente

forma: muestrear una señal es tomar valores de una señal continúa $x(t)$ a determinados instantes de tiempo t_n:

$$x(t = t_n) = x[n], \text{donde } t_n = n \cdot T_m \qquad (3.1)$$

El intervalo de tiempo que hay entre dos muestras consecutivas se denomina período de muestreo, y se mide en segundos. Su inversa $f_m = 1/T_m$ se denomina frecuencia de muestreo y se mide en ciclos por segundo o Hz[69]. En el proceso de muestreo pasamos de una señal continua a un conjunto de muestras (es decir, puntos discretos en el tiempo, ver figura 3.2.3).

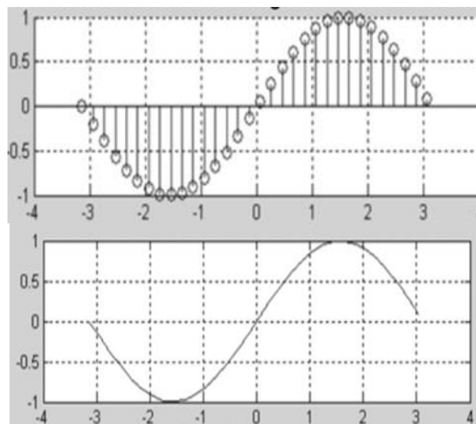


Figura 3.2.3. Señal analógica y señal digital

El teorema de Nyquist demuestra que para representar adecuadamente una sinusoide es necesario al menos tener dos muestras por cada ciclo de la sinusoide. Por lo tanto, para representar adecuadamente un sonido, la frecuencia de muestreo f_m debe ser

69 Herzios es una medida de la frecuencia, variaciones de un acontecimiento por unidad de tiempo.

mayor como mínimo, del doble de la frecuencia más alta f_{max} contenida en la señal:

$$f_m \geq 2 \cdot f_{max} \qquad (3.2)$$

Se denomina frecuencia de Nyquist a la frecuencia más alta que se puede capturar con una determinada frecuencia de muestreo f_m:

$$f_{Nyquist} \geq f_m/2 \qquad (3.3)$$

El oído humano es capaz de detectar frecuencias de hasta 20.000 Hz (en las mejores condiciones). Por lo que para muestrear correctamente cualquier sonido se necesitará una frecuencia de muestreo superior o igual a 40.000 Hz. Las frecuencias más habituales con las que trabajan las tarjetas de sonido son 44.100 Hz, 48.000 Hz, 96.000 Hz y 192.000 Hz.

Cuando el sonido tiene componentes de frecuencia que son mayores que la mitad de la frecuencia de Nyquist se produce el fenómeno de *aliasing*, que inventa frecuencias de aproximadamente la diferencia entre la frecuencia original y la de muestreo. Las soluciones posibles a este problema son: aumentar la frecuencia de muestreo para que esta sea mayor o igual al doble de la frecuencia máxima de la señal, o realizar un filtrado de las frecuencias por encima de la frecuencia de Nyquist, a estos filtros se les denominan antialiasing y son filtros de tipo paso bajo[70].

3.1.2 Cuantificación

Una vez la señal ha sido muestreada nos encontramos con un conjunto de muestras o de valores continuos de la amplitud de la señal. La cuantificación se realiza al limitar los posibles valores de

[70] Un filtro paso bajo, permite el paso de las frecuencias más bajas y atenúa las altas.

amplitud de una señal, definiendo una serie discreta (no continua) de valores posibles.

El número de posibles valores de amplitud viene determinado por la resolución del convertidor. La resolución de los convertidores depende del tamaño que se utiliza para representar cada una de las muestras de la señal. La resolución de un convertidor se mide en número de bits que utiliza, y un convertidor de n bits de resolución cuantificará a 2^n valores de la señal. Un sistema con una resolución de 4 bits tendría sólo 16 valores diferentes de señal (2^4), y un sistema de 16 bits tendría $2^{16} = 65536$ valores diferentes. Cuanto mayor sea la resolución del convertidor, mayor precisión tendremos en la representación de la señal.

El ruido de cuantificación aparece en el proceso de cuantificación, cuando sustituimos la amplitud de la muestra por la amplitud más cercana del conjunto de valores admitidos. Se define como la diferencia entre la señal muestreada antes de cuantificar y la señal muestreada y cuantificada:

$$r[n] = x[n] - x_c[n] \quad (3.4)$$

$x[n]$ sería el valor de la muestra sin cuantificar, $x_c[n]$ sería el valor de la muestra cuantificada, y $r[n]$ sería el valor del ruido de cuantificación para la muestra número n. El ruido de cuantificación representa la pérdida de calidad de sonido al cuantificar.

3.1.3 Codificación

El proceso de codificación consiste en asignar un código binario o conjunto de bits a cada uno de los valores posibles de las muestras de la señal. Hay muchas posibilidades de realizar este proceso de

codificación. El códec[71] es el código específico que se utiliza para codificar y decodificar datos. El códec normalmente incluye parámetros referentes a todo el proceso de digitalización, para saber cómo se tiene que realizar el proceso de conversión:

- Número de canales: monoaural, binaural o multicanal.
- Frecuencia de muestreo.
- Resolución: número de bits. Como hemos visto en el punto anterior, cuanto mayor sea el número de bits que utilicemos, mayor resolución tendremos y menor ruido de cuantificación.
- Velocidad o tasa de transferencia (en bits por segundo).

3.1.4 La señal digital de sonido

Como resultado del proceso de digitalización lo que tendremos es una señal digital de sonido, representada por una secuencia de números:

$$\ldots, x[n-1], x[n], x[n+1], \ldots \qquad (3.5)$$

Donde el índice $n \epsilon \mathbb{N}$, denota el número de muestra y $x[n]$ denota el valor de la muestra.

Como ya hemos visto el rango de valores que puede tomar una muestra dependerá de la resolución de la tarjeta de sonido. La mayoría de convertidores trabajan con resoluciones de 16 o 24 bits, con lo que el rango de valores estará entre 2^{16} y 2^{24} respectivamente. No obstante cuando se trata de una señal sonora esos valores oscilarán entre $-2^8 \ldots 2^8$ y $-2^{12} \ldots 2^{12}$. Las muestras se almacenan usando el concepto de palabra (*byte*) que equivale a 8 bits, con lo que la resolución de 16 bits es de 2 bytes y la de 24 es de 3 bytes. En

[71] Codec es la abreviatura de codificador y decodificador

ocasiones también nos encontramos que estos números enteros se normalizan para que tomen valores en el intervalo real $[-1 \dots 1]$.

3.2 Comunicación entre la aplicación y la tarjeta de sonido

Si desde una aplicación se quiere procesar la señal digital sonora, tendremos que llevar las muestras de la tarjeta de sonido a la aplicación. Hasta el momento se ha revisado cómo se realiza el proceso de digitalización del sonido y los conceptos que están involucrados, ahora veremos cómo esas muestras llegan hasta la aplicación.

Para una aplicación la captura de muestras de sonido es un proceso más o menos transparente, puesto que hay una componente de software, que es el controlador del dispositivo físico, llamado habitualmente *driver*[72], que se encarga de hacer transparente este proceso.

Un *driver* proporciona una interfaz para poder utilizar el dispositivo físico, de forma que el sistema operativo y las aplicaciones puedan interactuar con el dispositivo. Lo que se intenta es que esa interfaz sea una componente estandarizada de manera que no tengamos que especializar las aplicaciones para cada uno de los modelos de dispositivos. El objetivo de la interfaz es realizar una abstracción de los detalles del periférico.

Los lenguajes de programación a su vez utilizan otra capa por encima del *driver* para poder acceder a los dispositivos. Son las librerías que

[72] Habitualmente en terminología informática el término *driver* no se traduce, de forma que a partir de ahora lo mantendremos en su idioma de origen.

en este contexto viene a ser lo mismo que el *driver* pero a nivel de lenguaje de programación. De forma que un programador para acceder al dispositivo lo hace a través de la librería que a su vez se comunica con la interfaz del *driver,* que finalmente accede al dispositivo. De manera que las librerías de sonido hacen transparente el acceso a la tarjeta de sonido. Incluso este esquema, dependiendo del lenguaje de programación usado, puede ser independiente del sistema operativo subyacente.

Lamentablemente cuando se quiere trabajar con tarjetas de sonido que poseen más de dos entradas, al estar poco estandarizados las interfaces de los *drivers*, varían mucho de un sistema operativo a otro. Esto conlleva que se deban usar librerías de sonido especializadas para cada uno de los sistemas operativos.

Las muestras de sonido digitalizado se transfieren a la aplicación en bloques de muestras, que se denominan *buffers*. El número de muestras que puede contener un *buffer*, normalmente múltiplo de 2, es un dato que se debe configurar, bien a través de la tarjeta de sonio o a través de la interfaz del *driver*. Los valores habituales son 512, 1024, 2048, 4096, por poner algunos ejemplos. El dispositivo trabaja con el mismo tamaño de *buffer* para todas las entradas.

Conforme llegan las muestras se van llenando los *buffers*, y cuando están llenos se transfieren a las aplicaciones a través de la interfaz del *driver*. El tiempo que tarda en llenarse un *buffer* depende de la frecuencia de muestreo con la que esté trabajando el conversor. Si el tamaño del buffer es muy grande se genera un retraso entre el instante en que se produce la señal, se digitaliza, se almacena en el *buffer* y luego se pasa a la aplicación. Si el tamaño del *buffer* es muy pequeño, la aplicación invertirá mucho tiempo en la comunicación y el procesamiento propio del sonido se verá afectado.

Si se tiene en cuenta el modelo de buffer para realizar la transferencia de muestras entre la tarjeta de sonido y el programa, podemos ver formalmente que una secuencia de sonido es un conjunto de buffers, y ello nos permite también ver la secuencia de muestras de un canal de sonido, cuyos valores están normalizados entre -1 y 1, como un conjunto de *buffers* de $(r-1) \geq 2$ elementos:

$$\ldots \text{buffer}_{t-1}, \text{buffer}_{t}, \text{buffer}_{t+1} \ldots \quad (3.6)$$

$$\text{buffer}_{t} = (s_0, s_1, \ldots, s_r) \quad (3.7)$$

Donde cada muestra $s_i \in [-1 \,..\, 1],\ 0 \leq i \leq r$ y $t = (t-1) \cdot (r-1)/f_m$, siendo f_m la frecuencia de muestreo. Esta es la notación que usaremos en el apartado siguiente para referirnos a las muestras que nos vienen de los *buffers*. Cuando se tiene más de un canal de sonido se etiqueta el *buffer* con el número del canal.

Por último comentar que según se deduce de la notación anterior, la frecuencia con la que se llenan los *buffers* es $\cdot\,(r-1)/f_m$, por lo que a mayor frecuencia de muestreo menor será la frecuencia con la que llenamos los *buffers*. Y cuanto mayor sea el tamaño de los *buffers* mayor será su frecuencia.

“Unos recuerdos están enlazados con otros, «cogidos de las manos» a través de conexiones endebles o fuertes en la sustancia gris cortical. Los hay que están dentro de otros, a modo de matrioskas. Unos son neutros desde el punto de vista emocional; otros incluyen connotaciones emocionales. Todos transportan literalmente nuestro yo existencial en el momento de su experiencia o adquisición, o incluso en un futuro imaginado“

Joaquín M. Fuster[73]

4 UN MÉTODO DE VISUALIZACIÓN DEL SONIDO BASADO EN LAS MUESTRAS OBTENIDAS DE SU DIGITALIZACIÓN.

El método de visualización que proponemos está pensado para un entorno en el que la música y la generación de imágenes se desarrollan en tiempo real, siendo idóneo para entornos de directo, como performances, conciertos, instalaciones interactivas o proyectos escénicos. No obstante se puede adaptar a entornos en diferido, aunque en estos casos los diferentes instrumentos o sonidos deben estar separados en canales, si queremos obtener unos resultados óptimos.

En este capítulo primero presentamos el fundamento numérico en el que se basa el método. Luego mostraremos a través de un ejemplo sencillo sus características principales, resaltando un esquema muy simplificado. Posteriormente realizaremos una definición más formal

[73] *(Fuster 2014, 103).*

del método, estableciendo un marco teórico adecuado para presentar las estrategias de ubicación de muestras. A continuación comentaremos las estrategias para abordar la ubicación espacial de las muestras de sonido que conducirán a los algoritmos de visualización. Y por último perfilaremos algunas ideas para la definición de un lenguaje del cambio.

4.1 De números a números

El lenguaje de los números o de las matemáticas[74] es un conjunto de signos y reglas concebido para comunicar, expresar y explicar fenómenos naturales. Para Pitágoras los números eran la esencia de las cosas, Galileo Galilei pensaba que el lenguaje de la Naturaleza estaba escrito en lenguaje matemático y quién pretendiera leerlo debería aprender dicho lenguaje[75]. Al matematizar la realidad, podemos hablar y razonar con un lenguaje sencillo y preciso, brindándonos la posibilidad de relacionar conceptos que de otro modo sería muy difícil de realizar.

Cuando capturamos a través de un sistema digital sonido o imagen en movimiento, estas señales se traducen o aproximan a un conjunto de números estructurados, denominados para el sonido muestras de sonido, y para las imágenes pixeles. En el universo digital toda la información se representa mediante números; no se trata de meros dígitos aislados sino que forman una estructura de relaciones entre ellos. El sonido son estructuras de números y las imágenes también, de manera que a través de un lenguaje común, representamos dos

[74] En (Devlin 2002) podemos encontrar un introducción muy extensa al lenguaje de las matemáticas.
[75] La cita clásica que se atribuye al científico renacentista es: “Las matemáticas son el lenguaje con el que Dios ha escrito el universo”. La cita proviene de una carta que le escribe a la duquesa Cristina.

conceptos que físicamente son diferentes[76]. Este nexo a través del lenguaje proporciona una oportunidad para estudiar sus semejanzas, poder compararlos e incluso poder fusionarlos.

La era digital en la que estamos inmersos, es un tiempo de lenguajes, y los lenguajes formales, numéricos o matemáticos son la esencia de los ordenadores, donde casi todo tiene que ver con protocolos, interpretaciones, compilaciones o traducciones de unos lenguajes numéricos en otros. Son los sistemas informáticos el mejor ejemplo de fusión de conceptos propiciados por un lenguaje común. Pero esto a nuestro entender no es más que una traslación simplificada de los mecanismos con los que los humanos realizamos nuestras deducciones y almacenamos nuestros conocimientos en el cerebro.

Si en un ordenador los sonidos y las imágenes se traducen en números, en nuestro cerebro se traducen en redes de neuronas repartidas por la corteza cerebral. El neurocientífico Joaquín Fuster lleva ya unos cuantos años defendiendo que los elementos básicos de conocimiento, que denomina cógnitos[77], forman redes de neuronas interconectadas entre sí. Realmente no hay una parte especial del cerebro donde reside un recuerdo, una experiencia o un conocimiento, sino que tenemos estructuras de neuronas conectadas en estructuras más o menos complejas. Esta forma de plantear cómo funciona nuestro cerebro nos lleva de nuevo al lenguaje común, un lenguaje sencillo donde tenemos neuronas formando estructuras, de la misma forma que en el ordenador todo se traduce a números que forman una determinada estructura. Todavía estamos muy lejos de

[76] Hasta finales del siglo XIX se pensaba que las ondas sonoras y las luminosas desde el punto de la física eran similares y se realizaron esfuerzos buscando correspondencias de colores con tonos musicales (ver capítulo 3).

[77] En (Fuster 2014) presenta el concepto de cógnito dentro del contexto que desarrolla en el libro, los cimientos dela libertad.

conocer cómo funciona nuestro cerebro pero actualmente estas son las ideas que se están barajando desde la neurociencia.

Desde un punto de vista fisiológico cuando se produce un estímulo sonoro, como la vibración de una cuerda de una guitarra, esta energía, que se propaga a través del aire, llega hasta nuestro oído que la transforma en impulsos nerviosos para finalmente llegar al cerebro quién se encargará de interpretarla (Roederer, 1997, 30-34). El cerebro para interpretar la sensación sonora, activará una red compleja de neuronas, donde obtendremos información de las características físicas del sonido, como grave, agudo, incluso si es una nota definida, el tipo de instrumento que ha generado el sonido en función de sus armónicos... Además es posible que parte de esa red rememore recuerdos de la juventud, nos recuerde el color del fuego en las noches de invierno, nos recuerde a una persona muy querida que nos dejó… Un estímulo sonoro no solo desencadena información digamos física o técnica sobre el sonido, sino que además se producen recuerdos, interpretaciones emotivas que en sí no tienen que ver directamente con el sonido, pero si forman parte de su percepción.

Mediante un proceso similar al del sonido, los estímulos visuales llegan a nuestros ojos y son transformados en impulsos nerviosos que llegan finalmente al cerebro (Pajares, 2001). El cerebro para interpretar la sensación visual, activará una red compleja de neuronas, donde obtendremos información de las características físicas de lo que estamos mirando, como si es grande, pequeño, que forma tiene… Además es posible que parte de esa red nos genere recuerdos de la juventud, nos recuerde el color del fuego en las noches de invierno, nos recuerde… Es muy posible que un estímulo sonoro active parte de la misma red que activa un estímulo visual y viceversa, de forma que en el lenguaje interno de nuestro cerebro no

se realiza una clara distinción entre lo sonoro y lo visual, salvo quizás en los aspectos básicos de sensaciones.

Las personas que según la neurofisiología son sinestésicos, pueden oír colores o ver sonidos. En estas personas la compartición de redes ante estímulos diferentes es habitual y no sólo desde el punto de vista emotivo sino también desde el punto de vista sensitivo.

El cerebro del ser humano traduce los estímulos exteriores a través de sus sentidos en impulsos nerviosos que activan y conexionan partes de la inmensa red de neuronas de nuestra corteza cerebral. En este único sistema lo que tenemos es un entramado de neuronas y conexiones entre ellas. Las conexiones están en continuo cambio, se crean, se activan, se refuerzan, se debilitan o desaparecen. La neuroplasticidad es la capacidad que tiene el cerebro para renovar el cableado y este se va moldeando con la actividad y el esfuerzo mental diario. Pese a que músicos, filósofos, pintores… desarrollen más unas determinadas áreas de la corteza cerebral que otras en función de su actividad, la plasticidad del cerebro es tal que incluso redes deterioradas por lesiones traumáticas pueden volver a formarse en otras partes del cerebro.

El universo digital de los ordenadores no es más que una réplica muy simplificada del comportamiento de nuestro cerebro, donde los estímulos exteriores se traducen en conjuntos estructurados de números y se interpretan a través de la ejecución de las instrucciones de los programas. En ambos casos el mundo exterior se modeliza con un lenguaje unificado y sencillo, a partir de ahí es cuando podemos empezar a realizar conexiones entre sonidos e imágenes. Si en un sistema informático los sonidos son números y las imágenes son números, podremos encontrar un mecanismo para generar imágenes en función de los números del sonido, imágenes que quizá un

sinestésico pudiera ver ante un estímulo sonoro. La construcción sintética de las imágenes además de los números tendrá que tener en cuenta la estructura y las relaciones que se establecen entre esos números.

Cuando representamos un sonido a través de un conjunto de muestras numéricas, cada una cuantifica intensidad sonora en un instante determinado (ver figura 4.1.1). Por lo tanto cada valor numérico viene etiquetado temporalmente y representa la unidad mínima de sonido. La relación del sonido con el tiempo es tal que una muestra aislada no se puede percibir, solo la percibimos a través del conjunto, a través del paso del tiempo.

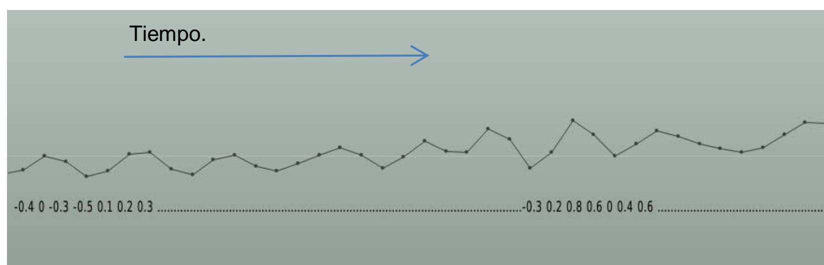


Figura 4.1.1. Detalle de una forma de onda de sonido.

Por otro lado el *pixel*[78] representa la unidad mínima de la imagen en movimiento[79], está formado por un valor numérico o conjunto de valores, una ubicación espacial y una etiqueta temporal. Si es una imagen monocromática el *pixel* estará formado por un único valor numérico que corresponde con la intensidad y si la imagen es en color por 3 o 4 valores dependiendo del modelo de color usado, y se

---

[78] Un pixel es el acrónico del inglés *picture element*, elemento de imagen.

[79] En general en todo el apartado se trabajará con imagen en movimiento, esto es una secuencia de imágenes estáticas etiquetadas temporalmente. Cuando hablamos de una imagen hacemos referencia a una imagen de una secuencia.

corresponden con la intensidad de cada uno de los canales de color. En cualquiera de los casos un *pixel*, representa intensidad lumínica. Si ampliamos una imagen mucho podremos ver los pixeles que componen la imagen (ver figura 4.1.2).

Figura 4.1.2. Imagen en la que se ven los pixeles.

El sonido y la secuencia de imágenes se representan internamente por números y estos están organizados temporalmente. La estructura numérica de la imagen es más compleja que la del sonido, puesto que si trabajamos en color cada *pixel* está formado por 3 números y además está ubicado espacialmente. Esto supone que los *píxeles* entre sí tienen dos tipos de relaciones una espacial y otra temporal.

Dada una imagen de una secuencia, cada uno de los *píxeles* está relacionado con el resto por su posición espacial. Como vemos en la figura 4.1.3, el *pixel* numérico 1 ocupa la posición de la imagen (i, j), donde i es la columna y j la fila, y está relacionado por su posición

con los pixeles que tiene a su alrededor. De esta forma los pixeles que ocupan las posiciones
$(i-1,j-1),(i,j-1),(i+1,j-1),(i-1,j),(i+1,j),(i-1,j+1)$,
$(i,j+1),(i+1,j+1)$ son los 8 vecinos directos de (i,j).

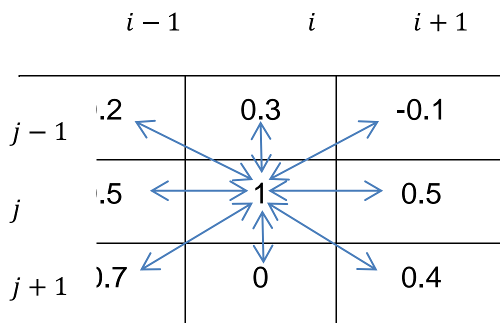


Figura 4.1.3. Relaciones espaciales entre *pixeles*.

Por otro lado tenemos que todos los píxeles de una imagen de la secuencia están relacionados temporalmente entre sí, todos ocurren en el mismo intervalo de tiempo o simplificando también podemos decir en el mismo instante. Dos *píxeles* de dos imágenes diferentes de la secuencia pueden ocupar la misma posición (i,j), pero entre ellos tendremos una relación temporal, puesto que uno es anterior o posterior al otro.

En el caso del sonido, su estructura interna es mucho más sencilla. En vez de una secuencia de imágenes, lo que tenemos es una secuencia de números. Entre las muestras de sonido hay una relación temporal, de manera que dadas dos muestras de la secuencia una será anterior o posterior a la otra (ver figura 4.1.4). Dada la muestra etiquetada temporalmente con r , cuyo valor es 1, tendrá la muestra $r-1$ como anterior y la muestra $r+1$ como siguiente.

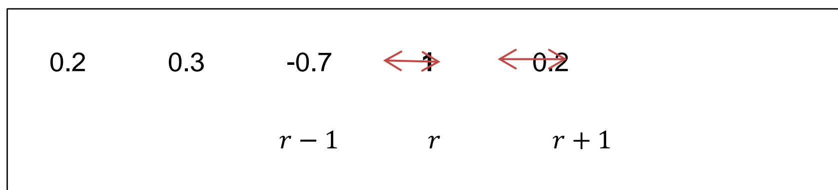


Figura 4.1.4. Relaciones temporales entre muestras.

Como conclusiones iniciales, de estas sencillas reflexiones, tenemos que pese a que el sonido y la secuencia de imágenes se representan a través del lenguaje numérico se observan algunas diferencias:

- los números de las muestras de sonido no se relacionan de forma espacial[80] como lo hacen los *píxeles*, y
- cada muestra se corresponde con un instante de tiempo diferente del resto, mientras que todos los *píxeles* de una misma imagen están etiquetados con la misma marca temporal.

El método que presentamos pretende generar una secuencia de imágenes a partir de un sonido. Si lo interpretamos desde el punto de vista numérico lo que queremos hacer es dada una secuencia de números etiquetados temporalmente redistribuirlos temporal y espacialmente para que formen una secuencia de imágenes (ver figura 4.1.5). Para redistribuir los números vamos a plantear una serie de premisas que entendemos son necesarias para que el sonido mantenga ciertas propiedades perceptuales en su visualización.

[80] En esta presentación simplificada, no hemos tenido en cuenta la posible ubicación física del sonido. Posteriormente se considerará como dato adicional la fuente del sonido o instrumento, como elemento diferenciador, entendiendo que cada instrumento puede estar ubicado en un lugar diferente.

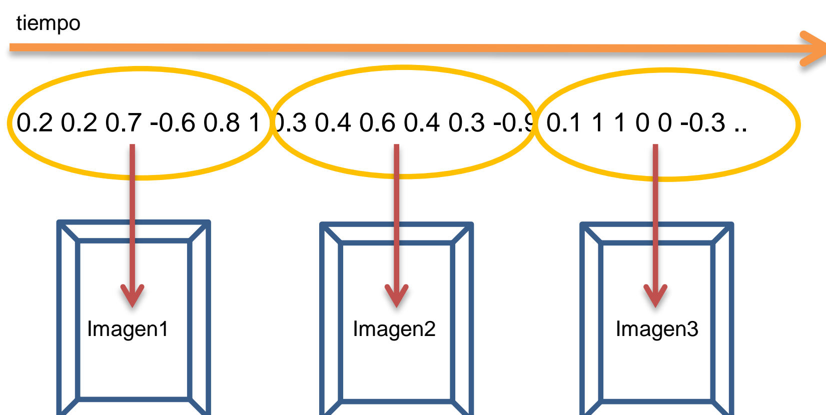


Figura 4.1.5. Redistribución de muestras en imágenes.

A cada una de las muestras de sonido le tendremos que asignar un *pixel* de una secuencia de imágenes, de forma que se tenga una estrecha correlación temporal entre ambas; la etiqueta temporal de la muestra debería estar dentro del intervalo temporal de la imagen o debería mantener una estrecha correlación temporal con ésta.

Además, también es importante tener en cuenta que en la percepción sonora, al igual que en la percepción visual, el efecto del estímulo permanece durante un tiempo[81]. Esto amplía el rango de correlación temporal de forma que muestras de sonido puedan aparecer visualmente en imágenes con etiquetas visuales posteriores a la de las muestras de sonido. Sería algo así como ver sonidos del pasado.

El sonido es un medio del tiempo y por consiguiente del cambio, percibir el sonido es percibir cambios. Como comentábamos antes,

[81] Históricamente este fenómeno en el campo visual ha sido conocido como persistencia visual o retiniana y se le han dado diferentes interpretaciones, a finales del siglo XIX Peter Mark Roget le dio una interpretación mecanicista, posteriormente se planteó su explicación dentro de la psicología de Gestalt, denominándose fenómeno *phi*, y actualmente se interpreta dentro de la neurociencia.

una muestra de sonido solo tiene sentido al percibirla en relación con sus muestras anteriores y futuras. Esta idea de conjunto o de grupo que aparece en los medios dependientes del tiempo debería mantenerse en las propuestas de visualización.

La dependencia de la imagen con el tiempo no es tan estrecha como con el sonido. Puesto que en un instante una imagen tiene una entidad propia, representa un cuadro, una fotografía. Un instante visual es mucho más complejo que un instante sonoro, porque tiene una identidad conceptual. La psicología de Gestalt nos dice que una cosa se ve, tiene sentido y significado por las relaciones entre las partes. El todo es mucho más que las partes. Una muestra de sonido desde un punto de vista conceptual no es nada, es como una letra en una palabra, la frase o la palabra es justamente la secuencia de muestras que genera un todo, que es el sonido. Desde un punto de vista similar un pixel no es nada, pero un conjunto de pixeles representa una imagen y un conjunto de imágenes representa el movimiento visual.

Todo esto nos lleva a pensar que una visualización del sonido necesariamente debe mantener ese concepto de secuencia, no se puede mostrar una muestra aislada de su entorno, de sus muestras vecinas. Para visualizar el sonido deberemos tomar un intervalo de tiempo y mostrar la secuencia de muestras. Si queremos además mantener parte de sus propiedades deberíamos tener en cuenta de alguna forma el orden que ocupan.

Cuando percibimos un sonido, no solo escuchamos los cambios de energía o intensidad, sino que también percibimos si el sonido es periódico o es un ruido, si tenemos sensación de altura definida, si su textura se corresponde con algún instrumento conocido, si es rítmico, si hay armonías o disonancias, si es rápido o lento... Estas

propiedades sin embargo vienen implícitas en la secuencia de muestras, en la secuencia de intensidades. Desde cierto punto de vista podemos decir que todos los procesos del sonido son procesos de la intensidad. Por ello para representar totalmente un sonido únicamente con la intensidad tenemos suficiente. Estudiando el orden de la secuencia podemos encontrar los ritmos, los armónicos o las alturas. Por eso una visualización que muestre la secuencia manteniendo parte de su orden estará muy cerca de la naturaleza del sonido.

Como vemos en la Figura 4.1.5, lo que proponemos es generar una imagen digital usando como pixeles los números del sonido correspondiente. El esquema se basa en que la representación digital interna de un sonido y de una imagen o conjunto de imágenes es en ambos casos un conjunto de números que representan intensidad sonora en el caso del sonido e intensidad lumínica en el caso de la imagen. De esta forma establecemos una relación directa entre ambos medios, a través del concepto de intensidad.

Desde otro punto de vista, lo que pretendemos es pintar una imagen con la secuencia de números de las muestras de sonido, donde cada pixel estará formado por muestras de sonido. Lo que proponemos es un método de visualización que tome las muestras de sonido de varios instrumentos y las presente como parte de los pixeles de una o varias imágenes.

4.2 Un método de visualización de sonido basado en muestras

El método que proponemos recoge a intervalos de tiempo muestras de sonido de la tarjeta de sonido, y con esos números forma una

imagen. Las tres premisas fundamentales en las que se apoya son las siguientes:

- Uso exclusivo de los números en valor absoluto de las muestras de sonido para construir las imágenes.
- Correlación temporal entre las muestras de sonido y la secuencia de imágenes, de forma que para construir una imagen en un instante determinado de tiempo t, sólo usaremos muestras de sonido obtenidas en dicho instante e instantes anteriores a t.
- En el devenir del tiempo todo cambia.

4.2.1 De intensidad sonora a intensidad luminosa

La primera de las premisas supone que la relación principal que establecemos entre el medio sonoro y el medio visual es la intensidad. En un sentido más amplio, la intensidad es el grado de fuerza con que se manifiesta un agente natural, una magnitud física, una cualidad una expresión, etc[82]. En sonido es la amplitud, la energía de las ondas sonoras y en luz es la intensidad el flujo luminoso. Cuando aproximamos una señal sonora mediante una secuencia de muestras, lo que tenemos es una estimación de su intensidad, de la misma forma que cuando aproximamos una imagen como una secuencia de números, lo que tenemos es una estimación de su intensidad. Es en base a esa estrecha relación, donde encontramos el punto adecuado para buscar conexiones directas entre los dos medios.

Es interesante como esta relación ya se establece en los inicios del cine sonoro, con la presentación visual del sonido, en lo que se conoce como sonido óptico analógico. Las variaciones de la señal

[82] Esta es la definición general de intensidad que aparece en DREAE.

sonora, previamente convertidas en variaciones de voltaje, eran reconvertidas en variaciones de luz, a través de una célula fotoeléctrica, y definitivamente registradas en el negativo de la película cinematográfica. Durante mucho tiempo esta fue la mejor forma de establecer la sincronización entre la imagen y el sonido. De alguna forma con el método que proponemos recuperamos esa conexión estrecha que ocurre entre las intensidades sonoras e lumínicas.

La simplicidad con la que diseccionamos un sonido para almacenarlo en un sistema informático, manteniendo todas sus propiedades perceptuales, ha sido el fundamento para idear el método de visualización. Si la secuencia numérica recoge, encapsula todas las propiedades del sonido por qué no va a seguir manteniéndolas si simplemente las presentamos de forma visual. Además no solo las mantiene sino que además proporciona una nueva conceptualización estética de su fusión. Tendremos que adaptar los números a las reglas de las imágenes, pero la base de la construcción es únicamente los números que representan la intensidad del sonido, esa propiedad casi mágica que engloba a un sonido y a una imagen.

Una de las adaptaciones más evidentes que vamos a manejar es el uso de números positivos. La esencia del sonido es la vibración que de forma numérica se representa con números variando entre valores negativos y positivos (por ejemplo -1 y 1), donde el 0 lo consideramos como la zona de reposo. Desde el punto de vista de la intensidad es igual el -1 que el 1. Por lo tanto en base a realizar una visualización y teniendo en cuenta que en imagen no hay un equivalente a esa variación, hemos decidido usar los números del sonido en valor absoluto. En este proceso somos conscientes que perdemos información valiosa del sonido, ya que el signo indica si la señal está en fase ascendente o descendente. No obstante recuperaremos esa

información en alguno de los algoritmos propuestos, aunque en la mayoría vamos a trabajar con el valor absoluto de los números.

Otra de las adaptaciones que tenemos que contemplar son las relacionadas con las diferencias numéricas entre la imagen y el sonido. Si nos centramos en imágenes en blanco y negro, un pixel viene identificado mediante un único número. En este caso la relación es directa una muestra de sonido un pixel. En este contexto se presenta un problema cuando varias muestras de sonido tienen que compartir un pixel. De forma general este problema nos lo vamos a encontrar también aunque manejemos imágenes en color.

Cuando la imagen es en color, tendremos que un pixel viene definido por 3 o 4 números, que representan cada uno de los canales de color. En función del modelo de color subyacente la interpretación de los canales puede variar. Los más habituales son RGB (*Red Green Blue*) y HSL (*Hue Saturation Luminance*). Para el desarrollo del método hemos elegido el modelo RGB, puesto que es la que más se parece a nuestro sistema óptico. La idea del modelo es representar un color mediante la mezcla por adición de los colores primarios de la luz, verde, rojo y azul. En sí el modelo no define de manera absoluta los colores, ya que posteriormente es el dispositivo último de visualización quién deberá interpretarlos, a través del espacio de color. No obstante pese a las diferencias finales de los dispositivos si tendremos una aproximación directa a los colores.

Cuando tenemos más de un sonido, que será lo habitual, nos referimos a cada uno de los instrumentos que suenan simultáneamente, que tendrán una representación a través de su secuencia de números. Las muestras de sonido tendrán que competir en su colocación espacial con el resto de muestras de los otros instrumentos. El posibilitar un modelo de color nos permite direccionar

cada instrumento a un determinado canal de color, evitando de esta forma las posibles colisiones. En cualquier caso como ya anunciábamos en el modelo monocromático, tendremos que establecer unas reglas para regular las colisiones.

4.2.2 Correlación temporal

Una secuencia de imágenes transcurre en el tiempo, lo mismo que un sonido. El sonido en sí vive en el tiempo, y es el medio más íntimamente relacionado con él, el sonido es tiempo y su existencia está ligada a él. Como hemos visto en el apartado de contextualización, la corriente de abstracción iniciada a principios del siglo XX, estableció una línea de trabajo en las relaciones entre música e imagen, la música como ente abstracto debe representarse sin figuraciones, pero la pintura limitada temporal y espacialmente solo podía recoger parte de la esencia del sonido, la abstracción. Por el contrario es el movimiento del cine abstracto o música visual, el que recoge esa esencialidad del tiempo fusionándola con la representación abstracta. Las imágenes evolucionan en el devenir del tiempo.

La segunda premisa establece la translación de la naturaleza temporal del sonido a las imágenes. Para formar una adecuada correlación, tendremos que determinar un intervalo temporal durante el cual recoger las muestras numéricas y generar con esos números una imagen. El intervalo temporal marcará el periodo con el que generamos una nueva imagen. También lo podemos entender como una única imagen, cuyo contenido evoluciona con el tiempo. En este sentido, la imagen es como una ventana al micro mundo de las muestras del sonido y estas conforme se van agrupando en tandas pasan por la ventana. Esta premisa no elimina la posibilidad de mostrar, como ecos del pasado, tandas anteriores. Percibimos el

sonido y las imágenes más cercanos en el tiempo, pero el pasado deja un rastro que se mantiene en nuestra percepción sonora y visual. Es por esto que también consideraremos como estrategia de visualización general las muestras pasadas.

En el momento que generamos una imagen podemos usar todas las muestras de sonido actuales, las inmediatas y las muestras de sonido pasadas.

4.2.3 Todo cambia

La tercera y última premisa establece un principio que rige la propia naturaleza del sonido. De manera que si queremos representar visualmente el sonido deberemos tener presente y tratar de acomodar esta idea de cambio. De alguna forma se trata de una consecuencia de las premisas anteriores. El sonido es dependiente del tiempo y los números que representan en el sonido cambian.

En la propia naturaleza del sonido, incluso en sonidos muy periódicos, podemos observar como la intensidad de su frecuencia fundamental varía, la intensidad de los armónicos varía, los propios armónicos desaparecen y aparecen nuevos. Salvo los sonidos más sintéticos y puros, todo en el sonido es cambia. Es por esto que pensamos que el propio método de visualización debe contemplar mecanismos para que se propicien los cambios.

Pese a la sencillez de la idea de tomar las muestras de sonido y ubicarlas como parte de los pixeles, se presentan un gran número de opciones en el proceso de adecuación de un medio al otro. Ese conjunto de opciones pueden determinarse inicialmente y permanecer fijas o variar a lo largo del tiempo. Como premisa establecemos que el método favorece que los parámetros cambien, manteniendo de esta forma la propia naturaleza del sonido.

4.2.4 Ejemplo

Con la finalidad de mostrar las principales características que presenta este método vamos a desarrollar un ejemplo muy sencillo de forma que se vea cómo podemos visualizar los números de las muestras del sonido.

Supongamos que tenemos conectada a la tarjeta de sonido de un sistema informático una fuente sonora, por ejemplo un teclado que emite un sonido muy simple, una onda sinusoidal. Como hemos visto en el apartado 3, las tarjetas de sonido traducen variaciones eléctricas en variaciones numéricas, de forma que tras un periodo de tiempo tendremos un conjunto de muestras de sonido, números. Estos números se almacenan en un vector o almacén de datos que denominamos $buffer$. Al cabo del tiempo volverá a llegar otro conjunto de muestras y este proceso se repetirá continuamente, obteniendo un conjunto de $buffers$ a lo largo del tiempo[83]:

$$\ldots buffer_{t-1}, buffer_t, buffer_{t+1} \ldots \quad (4.1)$$

En este modelo simplificado vamos a construir una imagen a partir de cada $buffer$, teniendo en cuenta las siguientes condiciones:

- Las imágenes van a ser monocromáticas. Con lo que cada muestra de sonido será un pixel.
- El número de muestras de sonido (tamaño del $buffer$) y el número de pixeles de cada imagen coinciden.

[83] Este proceso lo hemos detallado en el apartado 3.2 Comunicación entre la aplicación y la tarjeta de sonido, mostrando las fórmulas 3.6 y 3.7, las repetimos aquí para mayor claridad.

- El método para ubicar en la imagen las muestras de es por orden de posición en el $buffer$ colocándolas de izquierda a derecha y de arriba abajo ($idab$) en la imagen.

- Las muestras de sonido se interpretan en valor absoluto

- Las muestras de sonido toman valores en el intevalo $[-1..1]$ y los pixeles toman valores en el intervalo $[0..1]$, donde 0 es negro, 1 es blanco y cualquier valor intermedio es un nivel de gris.

Teniendo en cuenta esto, lo que propone esta versión simplificada de ejemple es:

- Dado un $buffer$ de 20 muestras, que representa parte de un patrón sinusoidal:

$$buffer = \begin{pmatrix} 0,0.2,0.4,0.6,0.8,1,0.8,0.6,0.4,0.2,0,-0.2,-0.4, \\ -0.6,-0.8,-1,-0.8,-0.6,-0.4,-0.2,0 \end{pmatrix}$$

- Obtenemos una imagen de 5×4, con los siguientes valores:

$$imagen = \begin{pmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 \\ 1 & 0.8 & 0.6 & 0.4 & 0.2 \\ 0 & 0.2 & 0.4 & 0.6 & 0.8 \\ 1 & 0.8 & 0.6 & 0.4 & 0.2 \end{pmatrix}$$

- Que tendrá el aspecto visual de la figura 4.2.1.

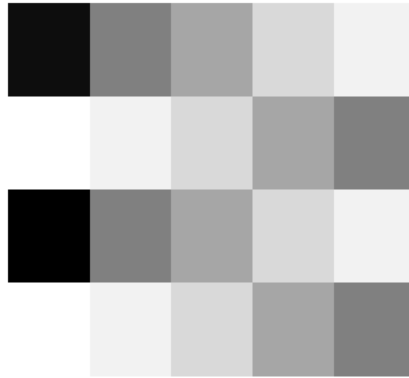


Figura 4.2.1. Imagen generada a partir de un sonido sinusoidal.

Con este modelo simplificado se pueden apreciar algunas de las relaciones directas que se producen entre la imagen y el sonido. Puesto que se trata de un fragmento de sonido periódico simple, el más simple que existe, este mantiene un patrón repetitivo que se aprecia en la imagen, a través de simetrías. En el apartado de casos prácticos, veremos como de esta fusión entre medios aparecerán nuevas percepciones y estéticas visuales.

4.2.5 Formalización del método de visualización

En este apartado vamos a formalizar el método con independencia de la estrategia usada para ubicar las muestras de sonido, y con independencia de los parámetros que potencialmente pueden variar a lo largo del tiempo.

En un escenario real tendremos un sistema, donde el $buffer$ varía a lo largo del tiempo y no solo tenemos un $buffer$ sino un conjunto de $buffers$, uno por cada una de las líneas de entrada.

Formalmente el problema lo podemos plantear de la siguiente forma. Dado un instante de tiempo t, y dado un conjunto de q $buffers$ con

$r+1$ muestras cada uno, siendo $r, s \in \mathbb{N}$, correspondientes a q líneas de entrada:

$$buffer_t^1 = \{s_0^1, \dots, s_r^1\},$$

$$buffer_t^2 = \{s_0^2, \dots, s_r^2\}$$

$$\dots \qquad (4.2)$$

$$buffer_t^q = \{s_0^q, \dots, s_r^q\}$$

Donde $\forall i \epsilon [0..r]\ y\ \forall j \epsilon [0..q]\ \ s_i^j$ es la $i-ésima$ muestra de sonido de la línea de sonido j que toma valores en el intervalo $[-1..1]$.

El método obtiene como resultado la matriz de salida $imagen$, en el instante *t:*

$$imagen_t = \begin{pmatrix} p_{0,0} & \cdots & p_{0,m-1} \\ \vdots & \ddots & \vdots \\ p_{n-1,0} & \cdots & p_{n-1,m-1} \end{pmatrix} \quad (4.3)$$

Donde n y m, son el número de filas y columnas respectivamente de la matriz $imagen$, siendo $n, m > 0\ y\ n \times m = r+1$, y $n, m \in \mathbb{N}$ y $p_{x,y}$ es el elemento de la matriz que ocupa la posición (x, y), siendo $0 \leq x \leq (n-1)$ y $0 \leq y \leq (m-1)$ y cada elemento de la matriz $p_{x,y}$, está formado por tres valores uno para cada uno de los canales de color. Estos valores serán muestras de sonido tomadas de alguno de los $s\ buffers$. Dependiendo de qué estrategias sigamos para asignar estos valores a los canales de color obtendremos diferentes algoritmos de visualización.

El método lo podemos generalizar para que obtenga en vez de una imagen un conjunto de imágenes de manera que lo que obtendremos es un conjunto l imágenes:

$$imagen_t^1 = \begin{pmatrix} p^{1}_{0,0} & \cdots & p^{1}_{0,m-1} \\ \vdots & \ddots & \vdots \\ p^{1}_{n-1,0} & \cdots & p^{1}_{n-1,m-1} \end{pmatrix}$$

$$imagen_t^2 = \begin{pmatrix} p^{2}_{0,0} & \cdots & p^{2}_{0,m-1} \\ \vdots & \ddots & \vdots \\ p^{2}_{n-1,0} & \cdots & p^{2}_{n-1,m-1} \end{pmatrix}$$

$$\ldots \qquad (4.4)$$

$$imagen_t^l = \begin{pmatrix} p^{l}_{0,0} & \cdots & p^{l}_{0,m-1} \\ \vdots & \ddots & \vdots \\ p^{l}_{n-1,0} & \cdots & p^{l}_{n-1,m-1} \end{pmatrix}$$

Cada uno de sus elementos estará formado por los canales de color, cuyos valores se obtienen de alguna combinación de las muestras de los $s\ buffers$.

Debido a que el espacio de presentación final, no necesariamente tiene que ser único, estas imágenes se podrían presentar por separado, se pueden aplicar operaciones de composición entre ellas e incluso se pueden realizar transformadas básicas para adecuarlas al espacio de presentación final.

4.3 Algoritmos de ubicación espacial de muestras

En este apartado vamos a presentar una serie de estrategias de ubicación espacial de muestras. Por establecer una clasificación los hemos dividido en tres grandes categorías:

- estrategias simétricas,
- estrategias basadas en reglas de comportamiento.

- estrategias mixtas con elementos aleatorios.

Las estrategias simétricas se fundamentan en que la estrategia de colocación es simple y obedece a reglas simples de posicionamiento en la imagen, por ejemplo de izquierda a derecha y de arriba abajo. Estas estrategias se pueden combinar entre sí usando en función de que la muestra cumpla una determinada propiedad o no, por ejemplo ocupe una posición par o impar.

Las estrategias basadas en reglas de comportamiento, tienen unas reglas más sofisticadas que dependen de los valores de las muestras ya ubicadas y otros factores. Otra de su característica es que puede que tengamos que renunciar a dibujar alguna muestra.

El último caso se corresponde con estrategias que son variaciones y combinaciones de las categorías anteriores pero introduciendo algún parámetro aleatorio. La idea de introducir algún valor aleatorio en la estrategia de ubicación proporciona una naturalidad adicional y como veremos conseguiremos resultados estéticos más ricos y variados. Este tipo de estrategias no las presentamos de forma independiente debido a que simplemente son variaciones introduciendo datos generados de forma aleatoria

Para simplificar la presentación formal de las estrategias asumimos un único $buffer$ y una única $imagen$ monocromática. Esta simplificación hace que se vea con mayor claridad el esquema formal de la estrategia.

Por lo tanto dado un $buffer$ de $r+1$ elementos y una $imagen$ de $n \times m = r+1$:

$$buffer = \{s_0, \dots, s_r\} \tag{4.5}$$

$$imagen = \begin{pmatrix} p_{0,0} & \cdots & p_{0,m-1} \\ \vdots & \ddots & \vdots \\ p_{n-1,0} & \cdots & p_{n-1,m-1} \end{pmatrix} \quad (4.6)$$

Tenemos que $\forall i \in \mathbb{Z}$ y $0 \leq i < r$, la muestra s_i, se ubicará en la posición (x, y) de la imagen. Donde x e y se definen:

$$x = f(s_i, i, n, m) \quad (4.7)$$

$$y = g(s_i, i, n, m) \quad (4.8)$$

Como vemos x e y dependen potencialmente del valor de la muestra s_i, de la posición de la muestra i en el $buffer$, del ancho y el alto de la imagen n y m respectivamente, y g y f son las funciones concretas que calculan la posición de ubicación de s_i. Por regla general el valor de muestra s_i se tomará siempre en valor absoluto, como hemos argumentado anteriormente. Por lo tanto para presentar las estrategias asumiremos que las muestras están en valor absoluto[84].

En el caso que la posición (x, y) tenga ya un valor asignado, debido a que previamente ha sido seleccionada para ubicar la muestra de otra línea, se aplicará el criterio definido en el método para resolver la colisión.

En el caso de las estrategias basadas en reglas de comportamientos f y g también tendrán en cuenta cualquier valor de las muestras ya ubicadas. Aunque para la mayoría de los casos este esquema sencillo nos bastará para formalizar la estrategia.

En los casos donde introducimos aleatoriedad alguno de los parámetros de f y g pueden ser obtenidos mediante generadores de números aleatorios.

---

[84] En el caso de no considerar el tema del valor absoluto, simplemente los valores negativos se toman como 0, esto es oscuridad.

4.3.1 Estrategias simétricas básicas

Para cada una de las estrategias que vamos a presentar:

- formularemos las funciones f y g que calculan la posición donde ubicamos cada muestra, salvo en aquellos casos que su presentación matemática sea muy compleja,
- formularemos la distribución de las muestras en la imagen resultante, salvo que la formulación sea muy compleja,
- mostraremos un esquema gráfico de cómo se van distribuyendo las muestras en la imagen. Este esquema visual lo aplicaremos a todos los métodos simétricos.

En el esquema gráfico tendremos una representación de las diferentes posiciones de la imagen, en forma de matriz, de manera que:

- la esquina superior izquierda se corresponde con la posición $(0,0)$ y la esquina inferior derecha con el último elemento de la matriz $(n-1, m-1)$,
- el punto azul con la palabra *inicio*, indica la posición inicial a partir de la cual se van ubicando las muestras en la dirección que indican la fecha naranja,
- las fechas azules indican cual es la siguiente posición después de la posición de la punta naranja, y
- el punto azul con la palabra *fin*, indica la posición final.

En la Figura 4.3.1 podemos ver un ejemplo, del esquema gráfico.

4.3.1.1 Izquierda a derecha y de arriba abajo (idab)

El primer algoritmo que vamos a ver se corresponde con la estrategia en la que las muestras de sonido se ubican de izquierda a derecha y de arriba abajo ($idab$). En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,m) = div(i,m) \qquad (4.9)$$

$$y = g(i,m) = mod(i,m) \qquad (4.10)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_0 & \cdots & s_{m-1} \\ \vdots & \ddots & \vdots \\ s_{m*(n-1)} & \cdots & s_{(m*n)-1} \end{pmatrix} \qquad (4.11)$$

En la Figura 4.3.1 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

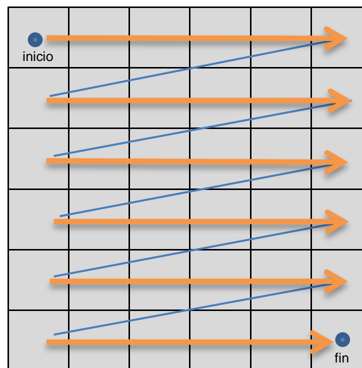


Figura 4.3.1 Esquema del algoritmo de ubicación $idab$.

En la Figura 4.3.2 podemos observar un ejemplo de una imagen correspondiente al método de izquierda a derecha y de arriba abajo ($idab$), para una señal sinusoidal básica.

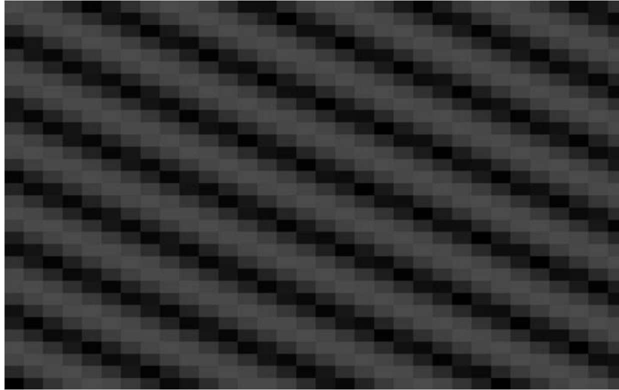


Figura 4.3.2 Imagen resultado del algoritmo de ubicación $idab$.

4.3.1.2 Izquierda a derecha y de abajo arriba (idba)

Con esta estrategia las muestras de sonido se ubican de izquierda a derecha y de abajo a arriba ($idba$).

En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,m) = (n-1) - div(i,m) \qquad (4.12)$$

$$y = g(i,m) = mod(i,m) \qquad (4.13)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_{m*(n-1)} & \cdots & s_{(m*n)-1} \\ \vdots & \ddots & \vdots \\ s_0 & \cdots & s_{m-1} \end{pmatrix} \quad (4.14)$$

En la Figura 4.3.3 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

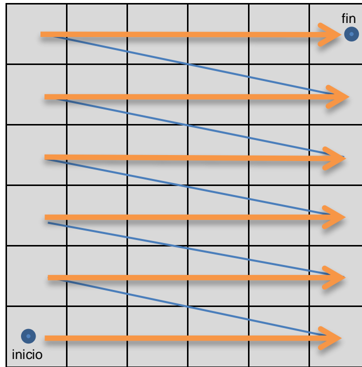


Figura 4.3.3 Esquema del algoritmo de ubicación $idba$.

4.3.1.3 Derecha a izquierda y de arriba a abajo (diab)

Con esta estrategia las muestras de sonido se ubican de derecha a izquierda y de arriba abajo ($diab$).

En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i, m) = div(i, m) \quad (4.15)$$

$$y = g(i, m) = (m - 1) - mod(i, m) \quad (4.16)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_{m-1} & \cdots & s_0 \\ \vdots & \ddots & \vdots \\ s_{(m*n)-1} & \cdots & s_{m*(n-1)} \end{pmatrix} \quad (4.17)$$

En la Figura 4.3.4 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

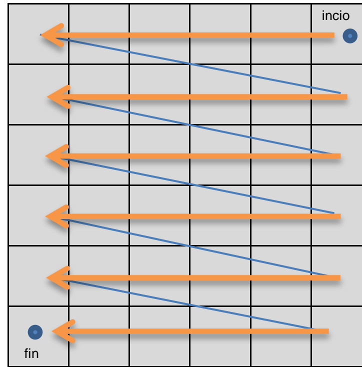


Figura 4.3.4 Esquema del algoritmo de ubicación $diab$.

4.3.1.4 Derecha a izquierda y de abajo a arriba (diba)

Con esta estrategia las muestras de sonido se ubican de derecha a izquierda y de abajo a arriba ($diba$).

En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i, m) = (n - 1) - div(i, m) \qquad (4.18)$$

$$y = g(i, m) = (m - 1) - mod(i, m) \qquad (4.19)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_{(m*n)-1} & \cdots & s_{m*(n-1)} \\ \vdots & \ddots & \vdots \\ s_{m-1} & \cdots & s_0 \end{pmatrix} \qquad (4.20)$$

En la Figura 4.3.5 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

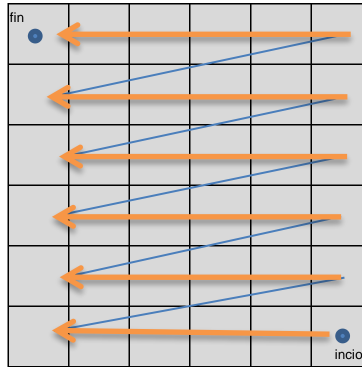


Figura 4.3.5 Esquema del algoritmo de ubicación $diba$.

4.3.1.5 Arriba a abajo y de izquierda a derecha (abid)

Con esta estrategia las muestras de sonido se ubican de arriba abajo y de izquierda a derecha ($abid$).

En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,n) = mod(i,n) \qquad (4.21)$$

$$y = g(i,n) = div(i,n) \qquad (4.22)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_0 & \cdots & s_{n*(m-1)} \\ \vdots & \ddots & \vdots \\ s_{n-1} & \cdots & s_{(m*n)-1} \end{pmatrix} \qquad (4.23)$$

En la Figura 4.3.6 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

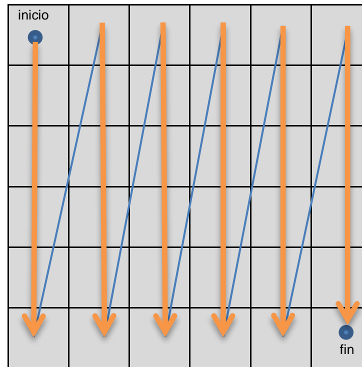


Figura 4.3.6 Esquema del algoritmo de ubicación $abid$

4.3.1.6 Arriba a abajo y de derecha a izquierda (abdi)

Con esta estrategia las muestras de sonido se ubican de arriba abajo y de izquierda a derecha ($abdi$).

En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,n) = mod(i,n) \qquad (4.24)$$

$$y = g(i,n) = (m-1) - div(i,n) \qquad (4.25)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_{n*(m-1)} & \cdots & s_0 \\ \vdots & \ddots & \vdots \\ s_{(m*n)-1} & \cdots & s_{n-1} \end{pmatrix} \qquad (4.26)$$

En la Figura 4.3.7 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

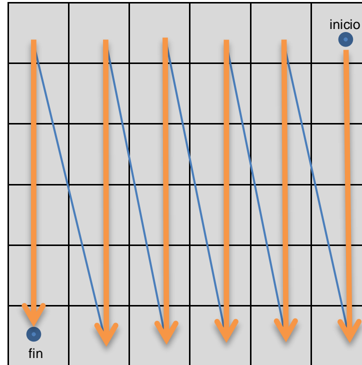


Figura 4.3.7 Esquema del algoritmo de ubicación $abdi$

4.3.1.7 Abajo a arriba y de izquierda a derecha (baid)

Con esta estrategia las muestras de sonido se ubican de abajo a arriba y de izquierda a derecha ($baid$).

En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,n) = (n-1) - mod(i,n) \quad (4.27)$$

$$y = g(i,n) = div(i,n) \quad (4.28)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_{n-1} & \cdots & s_{(m*n)-1} \\ \vdots & \ddots & \vdots \\ s_0 & \cdots & s_{n*(m-1)} \end{pmatrix} \quad (4.29)$$

En la Figura 4.3.8 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

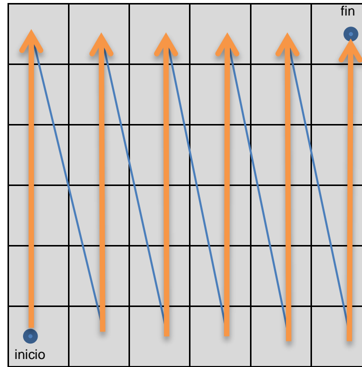


Figura 4.3.8 Esquema del algoritmo de ubicación $baid$

4.3.1.8 Abajo a arriba y de derecha a izquierda (badi)

Con esta estrategia las muestras de sonido se ubican de abajo a arriba y de derecha a arriba ($badi$).

En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,n) = (n-1) - mod(i,n) \qquad (4.30)$$

$$y = g(i,n) = (m-1) - div(i,n) \qquad (4.31)$$

Con esta estrategia la $imagen$ quedará de la siguiente forma:

$$lienzo = \begin{pmatrix} s_{(m*n)-1} & \cdots & s_{n-1} \\ \vdots & \ddots & \vdots \\ s_{n*(m-1)} & \cdots & s_0 \end{pmatrix} \qquad (4.32)$$

En la Figura 4.3.9 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

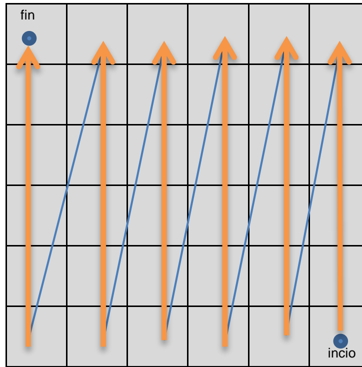


Figura 4.3.9 Esquema del algoritmo de ubicación $badi$

4.3.1.9 Alternando de izquierda a derecha y de arriba abajo (idzab)

Con esta estrategia las muestras de sonido se ubican alternativamente de izquierda a derecha, según la fila sea par o impar y de arriba abajo ($idzab$). En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,m) = div(i,m) \tag{4.32}$$

$$y = g(i,m) = \begin{cases} mod(i,m) & si\ div(i,m)\ es\ par \\ (m-1) - mod(i,m) & si\ div(i,m)\ es\ impar \end{cases} \tag{4.32}$$

En la Figura 4.3.10 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

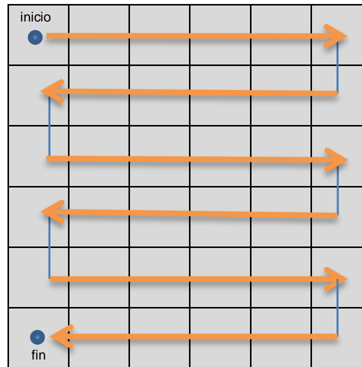


Figura 4.3.10 Esquema del algoritmo de ubicación $idzab$

4.3.1.10 Alternando de izquierda a derecha y de abajo a arriba (idzba)

Con esta estrategia las muestras de sonido se ubican alternativamente de izquierda a derecha, según la fila sea par o impar y de abajo a arriba ($idzba$). En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,m) = (n-1) - div(i,m) \tag{4.35}$$

$$y = g(i,m) = \begin{cases} mod(i,m)\ , n\ es\ impar\ y\ x\ es\ par \\ (m-1) - mod(i,m)\ , n\ es\ impar\ y\ x\ es\ impar \\ (m-1) - mod(i,m), n\ es\ par\ y\ x\ es\ par \\ mod(i,m), n\ es\ par\ y\ x\ impar \end{cases} \tag{4.36}$$

En la Figura 4.3.11 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

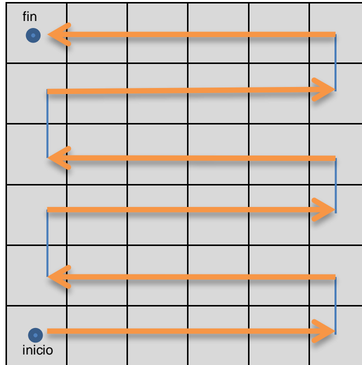


Figura 4.3.11 Esquema del algoritmo de ubicación $idzba$

4.3.1.11 Serpenteando de derecha a izquierda y de arriba abajo (dizab)

Con esta estrategia las muestras de sonido se ubican alternativamente de derecha a izquierda, según la fila sea par o impar y de arriba abajo ($dizab$). Es la misma estrategia que $idzab$ pero empezamos por la derecha. En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i, m) = div(i, m) \tag{4.37}$$

$$y = g(i, m) = \begin{cases} mod(i,m) & si\ div(i,m)\ es\ impar \\ (m-1) - mod(i,m) & si\ div(i,m)\ es\ par \end{cases} \tag{4.38}$$

En la Figura 4.3.12 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

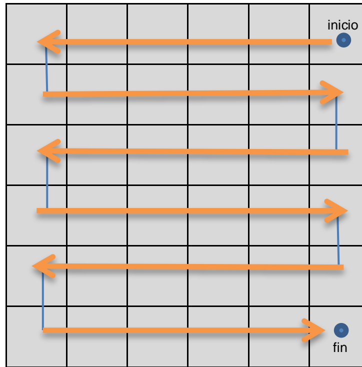


Figura 4.3.12 Esquema del algoritmo de ubicación $dizab$

4.3.1.12 Serpenteando de derecha a izquierda y de abajo a arriba (dizba)

Con esta estrategia las muestras de sonido se ubican alternativamente de derecha a izquierda, según la fila sea par o impar y de abajo a arriba ($dizba$). Es la misma estrategia que $idzba$ pero empezamos la última fila por la derecha en vez de por la izquierda. En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$x = f(i,m) = (n-1) - div(i,m) \tag{4.39}$$

$$y = g(i,m) = \begin{cases} mod(i,m)\ , n\ es\ impar\ y\ x\ es\ impar \\ (m-1) - mod(i,m)\ , n\ es\ impar\ y\ x\ es\ par \\ (m-1) - mod(i,m), n\ es\ par\ y\ x\ es\ impar \\ mod(i,m), n\ es\ par\ y\ x\ par \end{cases} \tag{4.40}$$

En la Figura 4.3.13 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

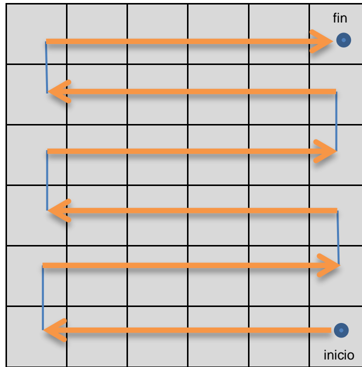


Figura 4.3.13 Esquema del algoritmo de ubicación $dizba$

4.3.1.13 Serpenteando de arriba abajo y de izquierda a derecha (abzid)

Con esta estrategia las muestras de sonido se ubican alternativamente de arriba a abajo, según la fila sea par o impar y de izquierda a derecha ($abzid$). En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$y = g(i,n) = div(i,n) \tag{4.41}$$

$$x = f(i,n) = \begin{cases} mod(i,n) & si\ y\ es\ par \\ (n-1) - mod(i,n) & si\ y\ es\ impar \end{cases} \tag{4.42}$$

En la Figura 4.3.14 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

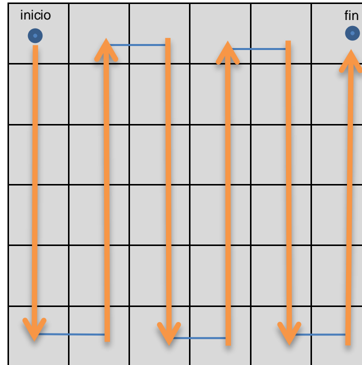


Figura 4.3.14. Esquema del algoritmo de ubicación $abzid$

4.3.1.14 Serpenteando de arriba abajo y de derecha a izquierda (abzdi)

Con esta estrategia las muestras de sonido se ubican alternativamente de arriba a abajo, según la fila sea par o impar y de derecha a izquierda ($abzdi$). En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$y = g(i, n, m) = (m - 1) - div(i, n) \tag{4.43}$$

$$x = f(i, n) = \begin{cases} mod(i, n)\ , n\ es\ impar\ e\ y\ es\ par \\ (n - 1) - mod(i, n)\ , n\ es\ impar\ e\ y\ es\ impar \\ (n - 1) - mod(i, n), n\ es\ par\ e\ y\ es\ par \\ mod(i, n), n\ es\ par\ e\ y\ impar \end{cases} \tag{4.44}$$

En la Figura 4.3.15 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

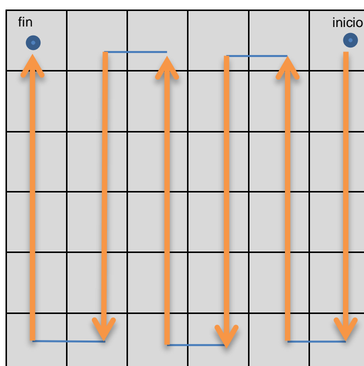


Figura 4.3.15. Esquema del algoritmo de ubicación $abzdi$

4.3.1.15 Serpenteando de abajo a arriba y de izquierda a derecha (bazid)

Con esta estrategia las muestras de sonido se ubican alternativamente de abajo a arriba, según la fila sea par o impar y de izquierda a derecha ($bazid$). Es la misma estrategia que $abzid$ pero empezamos la primera columna de abajo a arriba. En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$y = g(i, n) = div(i, n) \qquad (4.45)$$

$$x = f(i, n) = \begin{cases} mod(i,n) & si\ y\ es\ impar \\ (n-1) - mod(i,n) & si\ y\ es\ par \end{cases} \qquad (4.46)$$

En la Figura 4.3.16 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

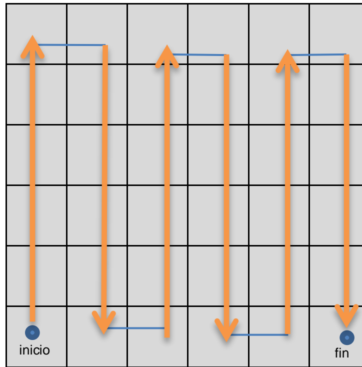


Figura 4.3.16. Esquema del algoritmo de ubicación $bazid$

4.3.1.16 Serpenteando de abajo a arriba y de derecha a izquierda (bazdi)

Con esta estrategia las muestras de sonido se ubican alternativamente de abajo a arriba, según la fila sea par o impar y de derecha a izquierda ($bazdi$). Es la misma estrategia que $abzdi$ pero empezamos la última columna de abajo a arriba. En este caso para cada muestra s_i las funciones f y g para calcular x e y son:

$$y = g(i, n, m) = (m - 1) - div(i, n) \qquad (4.47)$$

$$x = f(i, n) = \begin{cases} mod(i, n)\ , n\ es\ impar\ e\ y\ es\ impar \\ (n-1) - mod(i, n)\ , n\ es\ impar\ e\ y\ es\ par \\ (n-1) - mod(i, n), n\ es\ par\ e\ y\ es\ impar \\ mod(i, n), n\ es\ par\ e\ y\ par \end{cases} \qquad (4.48)$$

En la Figura 4.3.17 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para este método.

Figura 4.3.17. Esquema del algoritmo de ubicación $bazdi$

4.3.2 Estrategias simétricas en espiral.

Las estrategias en espiral se corresponden con una familia donde las muestras se van ubicando en la imagen formando una espiral[85]. De nuevo como en los casos vistos anteriormente, esta idea de espiral se puede abordar de muchas formas. Si interpretamos una imagen, con el mismo número de filas que de columnas, como una estructura formada por niveles (ver figura 4.3.18), la familia de estrategias plantea recorrer las posiciones de la imagen por niveles de manera que primero recorremos el primer nivel, luego el siguiente y así sucesivamente. Podemos usar varias estrategias, para realizar el recorrido del nivel, para pasar de un nivel a otro y para recorrer los niveles.

[85] Este estrategia también recuerda al efecto de coriolis.

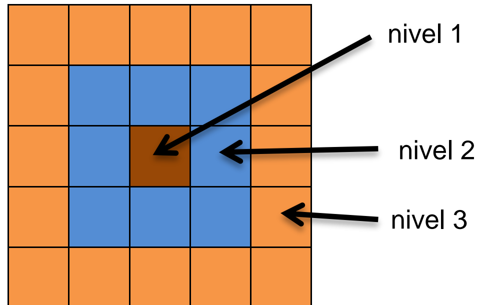


Figura 4.3.18. Estructura de una imagen por niveles.

En la Figura 4.3.19 podemos ver un esquema de cómo se van ubicando las muestras en la imagen para una de las variaciones de los métodos en espiral. Para este caso lo que tenemos es recorrido de nivel superior a inferior, empezando por la esquina superior izquierda de la imagen y avanzando según las agujas del reloj, para pasar de un nivel a otro lo hacemos por el vecino ortogonal más cercano.

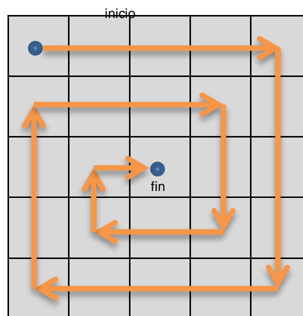


Figura 4.3.19. Esquema del algoritmo de ubicación en espiral

4.3.3 Otras estrategias simétricas.

En este apartado presentamos algunas estrategias simétricas que apoyándose en las básicas introducen variaciones. Inicialmente

veremos las estrategias que varían para las muestras pares o impares, y luego las estrategias que varían si las muestras de sonido llevan una línea ascendente o descendente y las estrategias que varían si las muestras de sonido son negativas o positivas. Por último comentaremos algunos criterios para resolver colisiones de posición.

4.3.3.1 *Estrategias combinadas en función si la muestra es par o impar*

Una de las estrategias consiste en aplicar estrategias básicas diferentes en función de si la muestra ocupa una posición par o impar en el $buffer$. Cuando se plantean este tipo de estrategias hay que tener en cuenta que las dos estrategias básicas pueden solapar posiciones. En los casos que se produzca una colisión se tendrá que tener previsto un método de resolución. Cada una de las estrategias simétricas vistas anteriormente posee su correspondiente estrategia combinada inversa, por ejemplo $idab$ y $diba$. A continuación mostramos algunos casos:

- Muestras pares $iadab$ y muestras impares $diba$. En este caso no tendríamos conflictos. En la figura 4.3.20 podemos ver un gráfico de la estrategia, donde las flechas azules se corresponden a muestras pares y las flechas naranjas a muestras impares. Un caso muy similar lo tendríamos mostrando las muestras pares con $diba$ y las impares con $iadab$. También son similares las estrategias con muestras pares con $idab$ y las impares $idba$ y al contrario.

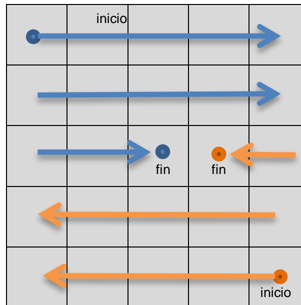


Figura 4.3. 20 Esquema combinado $iadab$ y $diba$

- Muestras pares con $abid$ e impares con $badi$. Este caso es similar al anterior pero las muestras se van ubicando de forma vertical en vez de horizontal. En la figura 4.3.21 podemos ver un gráfico de la estrategia, donde las flechas azules se corresponden a muestras pares y las flechas naranjas a muestras impares De forma similar tendríamos el caso contrario en el que las muestras pares se ubican con $badi$ y las pares con $abid$. Tambin son similares las estrategias

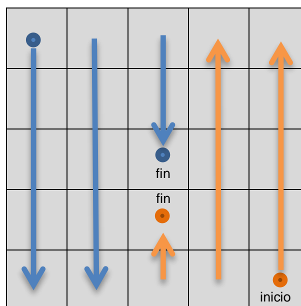


Figura 4.3.21. Esquema combinado $iadab$ y $diba$

- Muestras pares con $abzid$ e impares con $badi$. Este caso es similar al anterior pero las muestras se van ubicando de

forma vertical en vez de horizontal. En la figura 4.3.22 podemos ver un gráfico de la estrategia, donde las flechas azules se corresponden a muestras pares y las flechas naranjas a muestras impares De forma similar tendríamos el caso contrario en el que las muestras pares se ubican con $badi$ y las impares con $abid$.

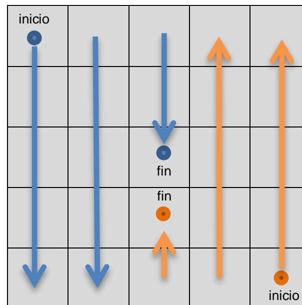


Figura 4.3.22. Esquema combinado $iadab$ y $diba$

La estrategia de diferenciar entre las muestras pares y las impares genera una gran cantidad de casos que no vamos a mostrar en esta memoria. Lo que si resulta interesante establecer es el mecanismo general que seguiremos cuando al combinar dos estrategias tengamos que ubicar dos muestras en una misma posición.

4.3.3.2 *Estrategias combinadas en función si la muestra está en línea ascendente o descendente, o si son negativas o positivas.*

La característica principal del sonido es que se trata de una señal de naturaleza vibratoria. En el contexto digital, la vibración se traduce en variaciones numéricas entre los valores $-1\ 0$ y 1. Para saber si una muestra está en línea ascendente o descendente seguiremos el siguiente criterio:

- Cuando el valor de una muestra es mayor o igual que el valor de la muestra anterior, diremos que la muestra está en *línea ascendente*.
- Cuando el valor de una muestra es menor que el valor de la muestra anterior, diremos que la muestra está en *línea descendente.*

Teniendo en cuenta esto, podemos aplicar las mismas variaciones que hemos planteado en el apartado anterior, para muestras que ocupan posiciones pares o impares. También podemos aplicar las mismas consideraciones en el caso de muestras positivas o negativas. Este tipo de variaciones mantienen parte de la estructura del sonido.

4.3.3.3 *Criterios para la resolución de conflictos por acceso duplicado a una posición de la imagen.*

Cuando a través de las estrategias combinadas se produce una colisión en una posición de la imagen, tendremos que dotar al método de un mecanismo para resolver esas situaciones.

Si dadas dos muestras la estrategia le asigna la misma posición, podemos resolverlo siguiendo alguno de los siguientes criterios:

- La última muestra en orden sustituye el valor de la anterior y por la tanto el valor de la muestra inicial se pierde, o viceversa.
- Los valores de las dos muestras se suman.
- Dejamos la muestra de mayor valor o la de menor.
- Colocamos el valor cero en esa posición.

Cada una de las posibles soluciones tiene sus ventajas e inconvenientes, nosotros pensamos que la más acorde con el método

que proponemos es la de sumar los dos valores, de forma que sus intensidades quedan reflejadas en el resultado final.

4.3.4 Estrategias basadas en reglas de comportamiento.

En este apartado vamos a presentar las estrategias que incorporan reglas de comportamiento más complejas que las que hemos visto anteriormente.

4.3.4.1 La hormiga de Langton

La hormiga de Langton es un una máquina de Turing[86] bidimensional con un conjunto de reglas muy sencillo, que sin embargo da lugar a comportamientos emergentes complejos (Langton 1986).

Cada cuadrado de la matriz se colorea o bien blanco o bien negro. Identificamos arbitrariamente a un cuadrado como la hormiga. La hormiga siempre está mirando en una de las cuatro direcciones cardinales y se mueve de una celda a otra, de acuerdo con las siguientes reglas:

- Si está sobre un cuadrado blanco, cambia el color del cuadrado, gira noventa grados a la derecha y avanza un cuadrado.
- Si está sobre un cuadrado negro, cambia el color del cuadrado, gira noventa grados a la izquierda y avanza un cuadrado.

La hormiga de Langton también se puede describir como un autómata celular[87], donde cada posición de la matriz se pinta de blanco o negro

[86] Una máquina de Turing es un dispositivo que manipula símbolos sobre una tira de cinta de acuerdo a una tabla de reglas. A pesar de su simplicidad, una máquina de Turing puede ser adaptada para simular la lógica de cualquier algoritmo que se ejecute en un ordenador.

y la hormiga se pinta de uno de ocho colores diferentes, dependiendo del color del cuadrado sobre el que esté y de la dirección en que esté mirando.

Lo que vamos a realizar es una adaptación del problema de la hormiga de Langton a nuestro método de visualización. Siguiendo con el esquema simplificado que hemos estado usando para presentar las estrategias anteriores, lo que tenemos es un $buffer$ y una $imagen$:

$$buffer = \{s_0, \dots, s_r\} \tag{4.5}$$

$$imagen = \begin{pmatrix} p_{0,0} & \cdots & p_{0,m-1} \\ \vdots & \ddots & \vdots \\ p_{n-1,0} & \cdots & p_{n-1,m-1} \end{pmatrix} \tag{4.6}$$

Donde $buffer$ tiene $r+1$ elementos y $imagen$ tiene $n \times m = r+1$ elementos.

En este contexto, la tortuga la definimos como una tupla formada por una posición inicial de $imagen$ (x, y), siendo $0 \leq x < n$ y $0 \leq y < m$ y una orientación $o \in \{west, north, , est, south\}$, que se corresponde con los 4 puntos cardinales, para simplificar asumimos el conjunto $o \in \{0,1,2,3\}$ respectivamente.

La idea de la adaptación consiste en tener una única imagen que vamos actualizando con las muestras de los $buffers$. En las otras estrategias cada vez que llegaba un buffer construíamos una imagen, colocando cada muestra en una posición, ahora cada vez que llega

---

[87] Un autómata celular es un modelo matemático para un sistema dinámico que evoluciona en pasos discretos. Es usado habitualmente para modelar sistemas naturales.

un buffer lo que hacemos es actualizar la matriz sobre los valores que ya tenía anteriormente. En este contexto $p_{x,y}$ denota el valor de una posición cualquiera de la imagen y $p'_{x,y}$, el nuevo valor de esa posición tras la actualización.

El proceso que vamos a seguir para realizar cada una de las actualizaciones es el siguiente:

- Para cada $s_i, \text{donde } 0 \leq i \leq r$, se realiza la siguiente actualización de la imagen y de la tortuga:
  a. Si $(p_{x,y} + s_i) \leq 1$, $p'_{x,y} = (p_{x,y} + s_i)$, y la nueva posición de la tortuga es (x', y') y $o' = mod\big((o + 1), 4\big)$, donde:
     - $(x', y') = (x + 1, y) \text{ si o} = 0$
     - $(x', y') = (x, y - 1) \text{ si o} = 1$
     - $(x', y') = (x - 1, y) \text{ si o} = 2$
     - $(x', y') = (x, y + 1) \text{ si o} = 3$
  b. Si $(p_{x,y} + s_i) > 1$, $p'_{x,y} = 0$, y la nueva posición de la tortuga es (x', y') y $si\ o = 0\ o' = 3\ \ si\ o \neq 0\ o' = o - 1$, donde:
     - $(x', y') = (x + 1, y) \text{ si o}' = 0$
     - $(x', y') = (x, y - 1) \text{ si o}' = 1$
     - $(x', y') = (x - 1, y) \text{ si o}' = 2$
     - $(x', y') = (x, y + 1) \text{ si o}' = 3$
  c. Si $(p_{x,y} + s_i) \leq 1$ entonces tomamos la muestra siguiente s_{i+1} y volvemos a aplicar las reglas.

En el caso a) para calcular la nueva posición de la tortuga primero aplicamos la regla de rotación de la orientación, que en este caso lo que tenemos es 90º a la izquierda y se avanza una posición, desde el punto de vista numérico este proceso consiste en realizar el siguiente cálculo $o' = mod\big((o + 1), 4\big)$, siendo mod la operación resto de la

división entera. En este caso la muestra se consume y la siguiente muestra a considerar es la s_{i+1}.

En el caso b) para calcular la nueva posición de la tortuga primero aplicamos la regla de rotación de la orientación, que en este caso lo que tenemos es 90º a la derecha y se avanza una posición, desde el punto de vista numérico este proceso consiste en realizar el siguiente cálculo $si\ o = 0\ o' = 3\ \ si\ o \neq 0\ o' = o - 1.\ (x', y') = (x + 1, y)$ si o = En este caso la muestra no se consume y la muestra se pierde.

Desde el punto de vista formal se mantiene la regla que sólo se usan las muestras de sonido para realizar el dibujo. En este caso conviven las muestras de buffers anteriores con el actual que puede provocar que determinadas zonas de la imagen queden estabilizadas y además no sabemos cómo va a reaccionar la evolución del sistema dado que las reglas son un poco diferentes del algoritmo original.

Al realizar el avance de la tortuga nos podemos encontrar que la nueva posición se corresponde con una posición fuera de los límites de la imagen, en ese caso simplemente volvemos a aplicar la actualización de la tortuga sobre la orientación anterior.

En el caso de tener varias líneas de entrada tendremos que tener una hormiga para cada una de las líneas.

4.3.4.2 El juego de la vida

En este apartado vamos a presentar una adaptación del conocido juego de la vida de Conway (Gardner 1970), orientado al problema de ubicar muestras en una imagen. Inicialmente definiremos el problema clásico del juego de la vida y luego plantearemos su adaptación.

Definición del juego de la vida

El juego de la vida es un autómata celular. Desde el punto de vista teórico es interesante ya que es equivalente a una máquina universal de Turing, esto quiere decir que todo lo que se puede computar algorítmicamente se puede computar en el juego de la vida[89]. Inicialmente fue concebido como un juego pero pronto se empezó a usar como un sistema para estudiar la evolución de determinados patrones.

El juego de la vida se basa en tres reglas sencillas: nacimiento, muerte y supervivencia. El juego se desarrolla sobre una rejilla cuadriculada, que en nuestro contexto es una matriz o imagen. Cada posición de la matriz puede estar vacía u ocupada por una célula. A partir de una configuración inicial en la que tenemos una serie de posiciones ocupadas, el juego evoluciona obteniendo diferentes configuraciones, siguiendo las siguientes reglas:

- Si una posición está rodeada por tres células vecinas entonces se crea vida, ocupándose esa posición por una célula.
- Si una célula tiene menos de 2 células vecinas, muere por aislamiento y su posición se queda vacía, y si tiene más de 3 células vecinas, muere por sofocación y también deja libre la posición.
- Si una célula tiene exactamente 2 o 3 células vecinas sobrevive al siguiente estado del juego.

Sorprendentemente dado un estado inicial, en el que colocamos una serie de células en la matriz, aplicando estas reglas tan simples el sistema evoluciona desarrollando patrones verdaderamente complejos. Algunos son estáticos, otros son recurrentes, otros

[89] Juego de la vida. En Wikipedia. Recuperado el 8 de noviembre 2015. de https://es.wikipedia.org/wiki/Juego_de_la_vida.

parecen desplazarse a lo largo de la matriz. En la figura 4.3.23, vemos tres estados del juego.

Figura 4.3.23. Estados consecutivos del juego de la vida

El juego de la vida es un juego de cero jugadores, donde su evolución está determinada por su estado inicial y no necesita ninguna entrada de datos posterior.

Desde la aparición del juego se han desarrollado múltiples variantes. Las reglas estándar se pueden sintetizar en la codificación 23/3, donde los dos primeros números indican cuantas células vecinas se requieren para que una célula permanezca viva y los números detrás de la barra inclinada "/" indican el requisito para su nacimiento. Por ejemplo 16/3 indica que una célula nace si tiene 3 células vecinas y sobrevive si tiene 1 o 6 células vecinas. Se conocen muchas variaciones del juego, aunque la mayoría son o demasiado caóticas o demasiado desoladas. Además se ha realizado variantes para un universo unidimensional y tridimensional.

El juego de la vida como estrategia de ubicación de muestras de sonido

El juego de la vida mediante unas reglas muy sencillas es capaz de crear patrones extremadamente complejos. Esta ha sido la base para tratar de adaptar el juego a nuestros intereses. El primer problema que nos encontramos en esta adaptación es que el juego de la vida

es un juego que no requiere jugadores y en nuestro contexto los jugadores son las muestras de sonido.

Esta situación nos ha llevado a usar el juego de la vida realmente no como elemento principal de visualización sino como estructura interna para decidir si mostramos una muestra o no y en qué lugar. De manera que su adaptación se presenta como complementaria para cualquiera de las estrategias que hemos comentado.

Por un lado tenemos la imagen que vamos a generar con las muestras y la estrategia de ubicación de muestras que se quiera. Y por otro tendremos una matriz de las mismas dimensiones que la imagen con la evolución del juego de la vida para una configuración inicial. En esta situación lo que haremos es mostrar la muestra en la posición que le corresponde si y solo si esta posición consultando su equivalente en la matriz del juego, está viva. En caso contrario la muestra se pierde.

De esta forma veremos las muestras de sonido pero filtradas por la evolución del juego de la vida. La idea es que el juego de la vida hace de máscara con respecto a la imagen. Con lo que de alguna forma estamos modificando la idea más purista del método, de visualizar sólo las muestras de sonido, introduciendo un elemento adicional que aunque simple, una matriz de dos estados: vivo o muerto, rompe la esencia de la visualización.

Formalmente el método lo podemos plantear de la siguiente forma:

Sea $imgSonido$ una imagen de dimensión $n\, x\, m$, obtenida a partir de un secuencia de muestras de sonido $buffer$ usando cualquiera de las estrategias sin pérdida estudiadas en este capítulo.

$$imgSonido = \begin{pmatrix} p_{0,0} & \cdots & p_{0,m-1} \\ \vdots & \ddots & \vdots \\ p_{n-1,0} & \cdots & p_{n-1,m-1} \end{pmatrix} \quad (4.49)$$

Donde $p_{x,y} \in [0..1]$ siendo $0 \leq x < n$ y $0 \leq y < m$, es una muestra del $buffer$.

Y sea $imgVida$ una imagen de dimensión $n\,x\,m$, resultado de un estado de la evolución del juego de la vida, donde cada uno de sus elementos o tienen el valor 0 o tienen el valor 1.

$$imgVida = \begin{pmatrix} q_{0,0} & \cdots & q_{0,m-1} \\ \vdots & \ddots & \vdots \\ q_{n-1,0} & \cdots & q_{n-1,m-1} \end{pmatrix} \quad (4.50)$$

Donde $q_{x,y} \in \{0,1\}$ siendo $0 \leq x < n$ y $0 \leq y < m$, es una célula del juego de la vida.

Como resultado de la estrategia combinada obtendremos una nueva matriz $imgResultado$ resultado de multiplicar las dos matrices anteriores elemento a elemento.

$$imgResultado = \begin{pmatrix} r_{0,0} & \cdots & r_{0,m-1} \\ \vdots & \ddots & \vdots \\ r_{n-1,0} & \cdots & r_{n-1,m-1} \end{pmatrix} \quad (4.51)$$

Dónde:

$$r_{x,y} = q_{x,y} * p_{x,y} \quad (4.52)$$

Desde el punto del tratamiento de matrices es una operación aritmética sencilla entre matrices. Desde un punto de vista creativo esta pequeña variación sobre la visualización introduce nuevas posibilidades a la hora de presentar las muestras de sonido. Estas variaciones las podemos encapsular bajo el concepto de variaciones

mediante operaciones lógicas y aritméticas. En esta situación la matriz principal es la creada por alguna de las estrategias anteriormente comentadas y luego se realiza algún tipo de operación aritmética con otra matriz.

4.3.5 Generalización de los algoritmos de ubicación cuando tenemos más de una línea de sonido.

Hasta el momento en todas las estrategias presentadas se han estudiado mediante un esquema simplificado en el que solo había una línea de sonido, esto es un buffer, y la imagen únicamente tenía un canal, de manera que a cada muestra le asignábamos una posición de la imagen. En un entorno real, lo que tendremos es un conjunto de líneas de sonido e imágenes con tres canales de color.

Una sola imagen

Cuando tenemos tres líneas de entrada y queremos obtener una única imagen, la estrategia más sencilla es asignar a cada línea de sonido uno de los canales de color. De esta forma independizamos claramente las líneas de sonido por cada uno de los canales. En el caso de tener más de tres líneas podemos aplicar estrategias que contemplen colisiones. El método de resolución más sencillo, como hemos visto antes, sería el sumar los valores de las muestras asignados a la misma posición de la imagen y al mismo canal de color.

Varias imágenes

Otra posible estrategia es distribuir las líneas de sonido en varias imágenes. Para cada una de las imágenes podemos aplicar algoritmos de ubicación básicos diferentes. En el caso de imágenes

en color, también se puede realizar la distribución que se crea oportuna. En cada instante de tiempo lo que tendremos es un conjunto de imágenes con los valores de las diferentes líneas de entrada. Finalmente usando operaciones aritméticas de la teoría de imágenes digitales[90], obtendremos un imagen final como resultado de componer las imágenes previas. El esquema general para dos imágenes lo podemos ver de la siguiente manera:

- Obtenemos las dos imágenes resultado de aplicar alguna de las estrategias de ubicación: $imgSonido1, imgSonido2$.
- Aplicamos una transformación aritmética a esas dos imágenes y obtenemos la $imgResultado$.

El modelo general de estrategia con pérdida se basa en realizar operaciones aritméticas con esas dos matrices. La idea es realizar transformaciones que usan la información contenida en la misma localización (posición en píxeles) de las dos matrices de entrada $imgSonido1$ y $imgSonido2$ para crear una nueva matriz $imgResultado$. Con este esquema es muy fácil extenderlo a varias imágenes. La dimensión de las dos imágenes tiene que ser igual. La función de transformación f_T, la aplicamos a todos los pares de píxeles en las imágenes de entrada. La función la podemos formalizar como:

$$imgResultado_{x,y} = f_T\left(imgSonido1_{x,y}, imgSonido2_{x,y}\right) \qquad (4.53)$$

Donde f_T es una función de dos variables y $0 \leq x < n \;\; y$ varían de 0 a $0 \leq y < m$. La función f_T, puede ser adición, substracción, multiplicación, división o cualquier otra función que opere con valores numéricos. Hay que tener cuidado con que la función no genere

[90] En (Pajares 2003) un desarrollo completo sobre las operaciones aritmético y lógicas en imágenes digitales.

desbordamientos o valores negativos, por lo que deberá tener un factor de escala apropiado para mantener los valores de salida dentro del rango de apropiado.

En la figura 4.3.24 podemos ver el proceso. Cuando aplicamos la transformación se obtiene un nuevo valor en la misma localización espacial de las matrices de entrada.

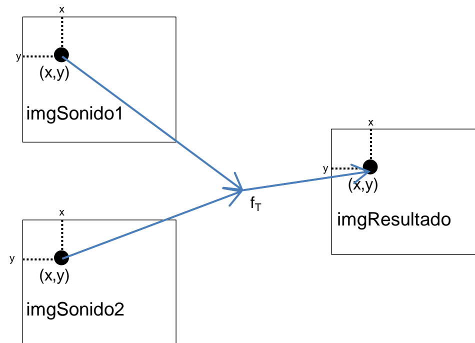


Figura 4.3.24. Transformación aritmética

4.4 El lenguaje del cambio.

El sonido es por excelencia el medio del cambio, en las estrategias que hemos revisado a lo largo del capítulo las imágenes se construyen conforme llegan nuevos datos de sonido, por lo tanto van cambiando sus valores también a lo largo del tiempo. No obstante si queremos acercarnos más todavía a esa idea de cambio propia del sonido, debemos dotar al método de un mecanismo que posibilite que todas las decisiones vayan cambiando a lo largo del tiempo. No solo las muestras, sino que podamos cambiar las estrategias de ubicación, que podamos cambiar la asignación de líneas de sonido a imágenes y a canales de color. En definitiva necesitamos un lenguaje en el que poder expresar todos estos cambios a lo largo del tiempo.

Aunque el diseño formal de este lenguaje se sale de las limitaciones que nos hemos planteado en este trabajo de investigación, si hemos planteado una simplificación de estas ideas en la implementación del método, apoyándonos en el propio lenguaje de programación.

Para poder hablar del cambio al menos será necesario manejar la idea de tiempo, de forma que en cada momento podamos consultar en que momento temporal nos encontramos. Para hacernos una pequeña idea de las características del lenguaje que necesitamos debería ser capaz de expresar cosas como:

- En el segundo 2 la línea de sonido 1, debe dibujarse en el canal rojo de la imagen1 siguiendo una estrategia $idab$.
- En todos los segundos que sean números pares la línea de sonido 1, debe dibujarse en el canal azul de la imagen1 siguiendo una estrategia $idab$ y en el canal verde de la imagen2 siguiendo una estrategia en espiral.
- La pantalla final de visualización tiene que dividirse en cuatro partes de forma que en la parte superior izquierda se muestre la imagen1, en la derecha la imagen2, en parte inferior izquierda a imagen3 y en la derecha la imagen3.
- También las dimensiones de las imágenes y su adecuación a la pantalla final pueden cambiar de forma dinámica.

Esto son simples ejemplos de lo que debería ser capaz de expresar el lenguaje. Para este trabajo de investigación hemos embebido este lenguaje dentro del propio lenguaje de programación, como veremos en el siguiente apartado. En trabajos futuros sería mucho más adecuado el definir un lenguaje independiente para poder expresar el cambio en la visualización del método.

“Ser digital es poder crecer. Ya de entrada, no tenemos que poner los puntos sobre las íes. Podemos construir enlaces para futuras expansiones y desarrollar protocolos de modo que unas cadenas de bits puedan informar a las demás sobre sí mismas.”

Nicolas Negroponte[91]

5 CICLOPE: UNA APLICACIÓN PARA LA VISUALIZACIÓN DE SONIDO EN UN ENTORNO DE TIEMPO REAL

Con la finalidad de poder experimentar adecuadamente las posibilidades visuales, que presenta el método de visualización propuesto, hemos desarrollado una aplicación, que cariñosamente hemos bautizado como Cíclope[92]. En la figura 5.1 podemos ver el logotipo de la aplicación.

Debido al carácter interdisciplinar del trabajo no vamos a ser excesivamente rigurosos en el uso de metodologías de ingeniería del software[93] y vamos a tratar de darle un carácter más didáctico que el que emplearíamos en un ámbito puramente informático. Además

[91] Cita tomada de (Negroponte 1995, 24)

[92] En la mitología griega, los cíclopes eran una raza de gigantes que tenían un solo ojo en mitad de la frente, siendo consideradas criaturas de fuerza y poder. Esa capacidad de ver de forma especial que tenían los cíclopes al tener un solo ojo, es la que ha inspirado el nombre de nuestra aplicación.

[93] La ingeniería del software es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas de software (Zelkovitz 1978).

intentaremos establecer algunos nexos de unión entre las disciplinas de informática y arte.

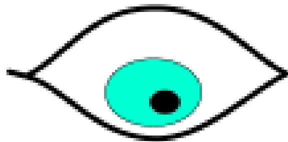


Figura 5.1 Logotipo de la aplicación Cíclope.

Nos gustaría dejar claro que el objetivo del desarrollo de este programa no ha sido realizar un producto de uso general, sino lo que hemos buscado ha sido diseñar un entorno de trabajo para probar la mayoría de los algoritmos propuestos en el método, siendo la única forma de evaluar la idoneidad del método en el campo de las artes visuales y comprobar la hipótesis de investigación que nos habíamos propuesto. En el apartado de casos prácticos analizaremos los resultados visuales del método. Además esta aplicación posteriormente pretendemos usarla para realizar conciertos multimedia en directo.

El desarrollo de la aplicación no ha seguido estrictamente un ciclo de vida clásico de proyectos en Ingeniería del Software debido a que inicialmente desarrollamos un prototipo rápido con la finalidad de evaluar la viabilidad del método. Si en el desarrollo de este primer prototipo no hubiéramos obtenidos resultados favorables posiblemente se hubiera abandonado el proyecto. Para el desarrollo

del prototipo inicial nos hemos basado en el modelo RAD[94], que se caracterizan por tener un desarrollo corto, sobre todo en el diseño de la interfaz gráfica de la aplicación (ver figura 5.2). Este primer prototipo se tuvo que desechar casi en su totalidad debido a que la estructura no era lo suficientemente robusta para poder sustentar todos los requerimientos que queríamos que tuviera la aplicación.

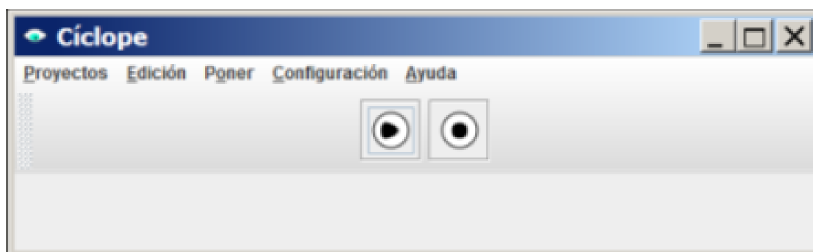


Figura 5.2. Interfaz gráfica del prototipo inicial de Cíclope

En el ámbito del desarrollo de software existe una gran variedad de metodologías que tienen como objetivo el modelado de los procesos involucrados en la solución del problema. En (Alonso 2005) encontramos una descripción pormenorizada de la mayoría de modelos. Todos ellos persiguen establecer pautas para el correcto y eficiente desarrollo de aplicaciones y dependiendo de la envergadura del proyecto, los informáticos son más o menos estrictos en su uso.

En el desarrollo de Cíclope hemos usado un ciclo de vida clásico inherente en casi todos los modelos, dentro del paradigma objetual. Desde el principio tuvimos claro que el desarrollo de la aplicación se abordaría desde el paradigma objetual, debido a que ha sido el paradigma de programación que hemos usado en los últimos años y porque pensamos que responde de forma eficiente a las necesidades del problema.

[94] RAD, es el acrónico de *Rapid Application Development.*

Inicialmente nos planteamos usar para el modelado de la aplicación el lenguaje UML[95], pero debido a las características del problema vimos que no era necesario, usar una notación tan rigurosa.

El ciclo de vida clásico o modelo clásico lo podemos encontrar en la mayoría de textos sobre ingeniería del software como (Sánchez 2003) y (Pressman 2001). Además pensamos que es el modelo más universal ya que se aplica en casi todas la áreas del conocimiento en las que se deben desarrollar proyectos, ya sea ingenieriles, artísticos, audiovisual,… El modelo consiste en responder de forma lineal a las preguntas: ¿qué quiero resolver? y ¿cómo lo puedo resolver? para finalmente obtener una solución al problema. Pese a que cada área de conocimiento especializa cada una de estas fases y añade otras como fases finales de prueba y mantenimiento, la sencillez del modelo lo hace universal y aplicable a la mayoría de problemas. Este modelo conlleva un mecanismo de realimentación debido a errores cometidos de diseño o simplemente por la necesidad de incluir requisitos nuevos que hacen que se replantee parte del sistema (ver figura 5.3). Esta concepción circular y dinámica del ciclo de vida hace que el desarrollo de proyectos se considere como una entidad viva, siempre en proceso de cambio. En el ámbito de la Informática la evolución de las aplicaciones es algo innato, prácticamente no se desarrolla ningún programa como producto final inamovible. Es práctica común etiquetar la aplicación con el número de versión, de manera que si el programa es útil se irá revisando y actualizando con los nuevos requisitos del sistema.

[95] UML es el acrónico de *Unified Modeling Languaje*, es un lenguaje formado por una serie etiquetas, diagramas y reglas orientado al desarrollo de software.

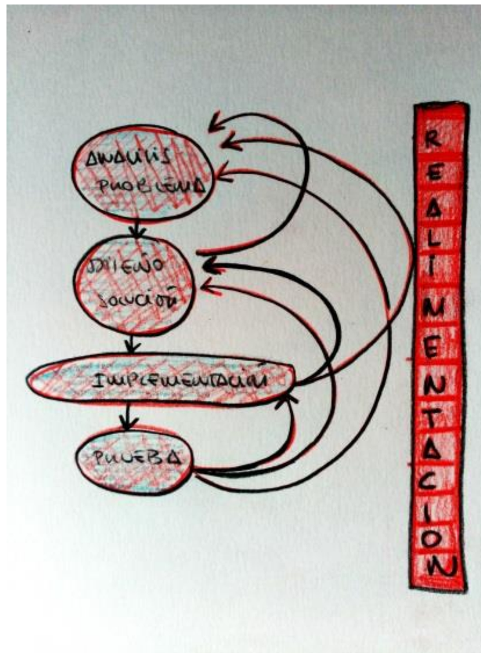


Figura 5.3. Fases en el desarrollo de la aplicación con realimentación.

El paradigma orientado a objetos se basa en la idea de abordar la solución del problema a través de los objetos, cosas o conceptos que intervienen en el problema. Podemos decir que es una forma natural de expresar los procesos del mundo real, actualmente es el modelo imperante en programación. En (Rumbaugh 1995) podemos encontrar un desarrollo pormenorizado del paradigma. Todos los sustantivos que usemos en la descripción de un problema, se convertirán en objetos y los objetos con características similares formarán clases. Pese a la complejidad que supone el desarrollo del software, este modelo es el que más claramente se acerca a la realidad. Por poner un ejemplo muy sencillo, si tenemos que diseñar un programa para dibujar, claramente tendremos que tener como objetos a los elementos básicos del dibujos como son las figuras

geométricas que a su vez se especializarán en rectángulos, óvalos, rectas, polígonos,…

Esta naturalidad con la que se aborda el desarrollo de software y la experiencia personal que tenemos en el paradigma objetual nos han decantado para elegirlo como paradigma de programación.

Este capítulo lo hemos divido de acuerdo a las fases que hemos seguido para el desarrollo de la aplicación. Lo primero que veremos es la fase de análisis en la que buscamos definir el problema y establecer todos los requisitos necesarios. Al final de esta etapa debemos podemos responder a la pregunta ¿qué queremos resolver?. Lo siguiente es la fase de diseño donde desarrollaremos una solución al problema y decidiremos las diferentes estrategias para resolver el problema. Se abordan problemas como lenguaje de programación, librerías y detectaremos los objetos y procesos fundamentales que manejará la aplicación. Para terminar el capítulo presentaremos la implementación de la solución y entraremos en algunos detalles relacionados con el código, aunque no seremos muy rigurosos. La fase de prueba queda implícita en el siguiente apartado de casos prácticos.

5.1 Análisis de la aplicación

Como punto de partida lo que pretendemos es desarrollar una aplicación o sistema[96] donde poder experimentar el método de visualización de muestras de sonido propuesto en el apartado 4.

El método en sí es un marco de referencia, lo que es realmente importante son los algoritmos de ubicación espacial de las muestras.

[96] A lo largo de este capítulo usaremos indistintamente el término sistema referido a un sistema informático y aplicación, entendiendo que una aplicación es un sistema informático.

La aplicación debe ser capaz de experimentar con varios algoritmos simultáneamente y que además puedan variar a lo largo del tiempo. También pretendemos que la aplicación se pueda usar en conciertos, performances o proyectos de artes escénicas.

En la figura 5.1.1 vemos un entorno de directo multisensorial en el que la música y los visuales se van generando en tiempo real, del grupo pdp11. Este tipo de evento sería el idóneo para la aplicación que proponemos.

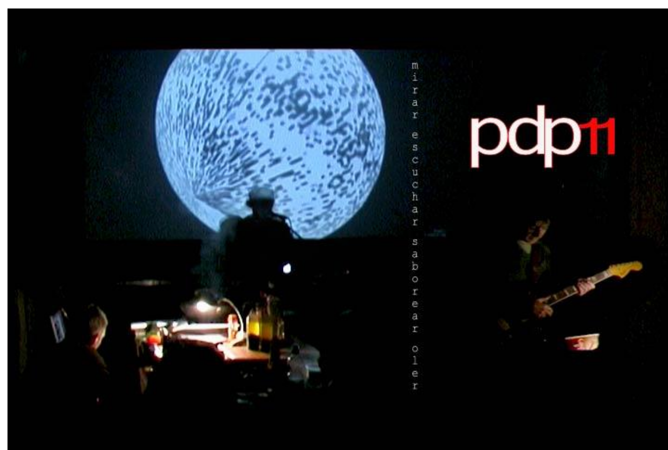


Figura 5.1.1. Plantilla para cartel de conciertos del grupo pdp11.

La figura 5.1.2 muestra de forma esquemática el concepto de aplicación que hemos desarrollado. Los elementos que encontramos son:

- Un conjunto de líneas de sonido de entrada de diversas procedencias, generadas en tiempo real.
- El procesamiento de dichas entradas.

- Un conjunto de resultados visuales. Una o varias imágenes resultantes de ubicar espacialmente las muestras de sonido recogidas por las entradas.

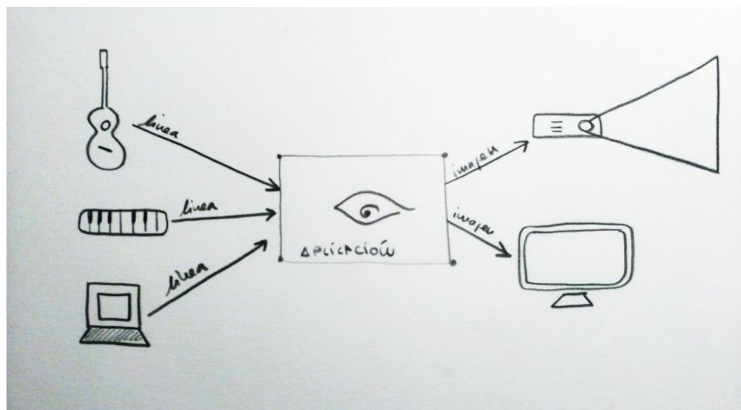


Figura 5.1.2. Esquema de entradas y salidas de la aplicación

Tras esta primera aproximación al concepto del problema, estudiaremos como el usuario de forma más precisa interactúa con el sistema y obtendremos los requisitos funcionales o no funcionales del sistema.

5.1.1 Casos de uso

El diseño de casos de uso introducido por Iván Jacobson (1992), es un método muy efectivo para entender con mayor profundidad las características del problema, ayudándonos a obtener los requerimientos funcionales de la aplicación. Como ya hemos comentado no hemos sido muy rigurosos en la aplicación de metodologías pero si las hemos tenido en cuenta en cada una de las fases.

Un caso de uso es una forma de expresar cómo alguien o algo ajeno al sistema lo usa. La idea es especificar el sistema en función de su

uso, describiendo las acciones que los usuarios deberán realizar en el sistema. El punto de vista de toda esta fase de análisis es pensar que el sistema es una caja negra, de la que no sabemos nada más que para qué sirve, pero no nos preguntamos cómo realiza su trabajo y el actor principal es el usuario. En la figura 5.1.3 podemos ver el diagrama de uso de la aplicación, donde distinguimos la interacción de tres actores o usuarios con el sistema por un lado el que hemos denominado técnico, el músico y el público. El rol de técnico y público se corresponde con personas físicas pero el rol de músico podría ser otro sistema que generara la señal sonora.

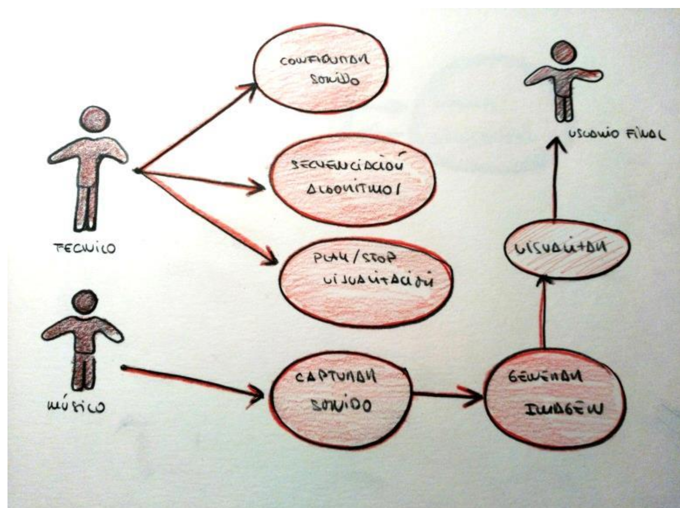


Figura 5.1.3. Diagrama de casos de uso de la aplicación

El esquema relacionado con el músico es el principal de la aplicación. El músico va generando la señal sonora y el sistema, si está el modo adecuado, irá capturando dicha señal y generando la imagen. El músico interactúa con la aplicación a través del sonido que emite.

Por otro lado el actor que hemos etiquetado como técnico realiza tres funciones diferentes. La primera la hemos denominado configuración

del sonido y consiste en determinar aspectos técnicos del sonido como el número de líneas sonoras que vamos a usar y otras cuestiones que tendrán que ver con los dispositivos de sonido. La segunda y quizá el requerimiento más complejo que tenemos en la aplicación es la de establecer una secuenciación de cambios a lo largo del tiempo de la visualización, con la finalidad de experimentar sin interrupciones con varios algoritmos de ubicación. Esto supone definir un pequeño lenguaje para que el técnico pueda dar las órdenes oportunas al programa. Por último el técnico debe poder arrancar, pausar y parar la visualización, esta funcionalidad se corresponde con la funcionalidad que tienen los reproductores de sonido y video a través de los botones *play*, *stop* y *pause*.

El usuario final tiene un rol pasivo, en el sentido que debe disponer de una buena presentación de las imágenes generadas. Representa el consumidor final del proceso de visualización.

5.1.2 Requisitos funcionales y no funcionales

Del análisis de los casos de uso, es fácil identificar los requisitos funcionales, que son las funciones de la aplicación descritas como un conjunto de entradas y de salidas. En esta fase las entradas de las funciones ya no son sólo usuarios, sino objetos que se comunican con otros objetos. Todos aquellos requisitos que no entren en el esquema de entrada, comportamiento y salida, son requerimientos no funcionales, por ejemplo que la interfaz gráfica sea amigable y fácil de usar.

Los requerimientos funcionales y no funcionales que hemos considerado para el desarrollo de la aplicación, en parte obtenidos de los casos de uso[97], son los siguientes:

a) La aplicación debe ser capaz de capturar y almacenar múltiples señales sonoras simultáneamente
b) La aplicación debe ser capaz de generar imágenes a partir de las muestras de sonido que estarán almacenadas en uno o varios buffers o contenedores.
c) La aplicación debe ser capaz de visualizar mediante un dispositivo de visualización las imágenes generadas con las muestras de sonido.
d) La aplicación debe manejarse con una amplia variedad de algoritmos de ubicación espacial de muestras.
e) La aplicación debe ser dependiente del tiempo real
f) La aplicación una vez programada debe ser autónoma, de manera que en un directo no necesite interacción con el usuario.
g) Debe ser sencillo incorporar nuevos algoritmos de ubicación espacial de muestras
h) La aplicación debe proporcionar al usuario un entorno amigable

En lo que resta de la fase de análisis detallaremos cada uno de estos requisitos, resaltando los conceptos y procesos más importantes que denotan. Como veremos en la fase de diseño estos conceptos se convertirán en clases y los procesos en métodos de esas clases.

[97] Hemos etiquetado cada uno de los requisitos con la finalidad de poder referenciarlos en el apartado 5.2 de diseño de la aplicación

a) ***CAPTURAR. La aplicación debe ser capaz de capturar y almacenar múltiples señales sonoras simultáneamente.***

Este requisito se obtiene del rol del músico que veíamos en el diagrama de casos de uso, añadiendo una de las características esenciales que debe tener la aplicación que es la posibilidad de manejar simultáneamente más de dos entradas de sonido. En el campo del sonido digital el concepto de múltiples entradas hace referencia siempre a más de dos, debido a que los protocolos internos de los sistemas operativos cambian de una modalidad a otra. Este por lo tanto es un requisito importante y que va a tener gran repercusión en la fase de diseño e implementación de la aplicación.

La función de captura es una de las principales de nuestra aplicación, a través de un dispositivo físico la señal sonora se digitaliza (ver apartado 3) y se almacena temporalmente en *buffers*. Nuestra aplicación debe ser capaz de acceder a dichos *buffers* para recuperar las muestras y almacenarlas de forma adecuada para su posterior visualización. Por lo que para cada una de las líneas de sonido tendremos un conjunto de *buffers*, que contienen las muestras que hemos ido capturando.

Adicionalmente, como veíamos en el diagrama de uso antes de proceder a la captura del sonido, se deberá configurar la aplicación para activar las líneas de sonido necesarias y realizar las conexiones físicas entre el dispositivo de captura de sonido y el ordenador, y establecer un estado de ejecución apretando un botón tipo *play*.

De este requisito concluimos que nuestra aplicación deberá manejar el concepto de línea de Entrada, una para cada una de las posibles entradas y de almacén de muestras de sonido, que deberán estar

asociadas a las líneas. Además de necesitar un dispositivo para la captura física del sonido dotado de más de dos entradas. También necesitaremos configurar los parámetros del sonido y dotar a la aplicación de estados para saber cuándo estamos en ejecución o cuando estamos en reposo.

b) GENERAR. La aplicación debe ser capaz de generar imágenes en color a partir de las muestras de sonido que estarán almacenadas en uno o varios buffers.

Otra de las funciones principales del sistema es la generación automática de imágenes a partir de las muestras de sonido. Esta función supone ir a los *buffers* de cada una de las líneas de entrada, visitar sus elementos y ubicarlos en una imagen o varias imágenes. De este requisito obtenemos otra idea importante que es que se debe poder generar más de una imagen.

Además tendremos que tener en cuenta que los elementos de las imágenes son pixeles y la mayoría de los modelos de color trabajan con 3 o 4 canales. Esto supone que deberemos establecer una distribución de muestras a canales de color. Para poder generar las imágenes, en cada momento debe estar definido el algoritmo de ubicación espacial a aplicar. Estas últimas funciones son las que aparecían en el caso de uso, relacionadas con la función del técnico de secuenciación de algoritmos.

Es muy importante que las muestras no sufran ni transformaciones, ni aproximaciones, ni se pierdan debido a que justamente la originalidad de este método de visualización radica en la pureza con la que se generan las imágenes a partir de dichas muestras.

De momento de este requisito podemos concluir que nuestra aplicación deberá manejar el concepto de Imagen. Estas imágenes

deberán sincronizarse con las líneas de entrada, y para construir la imagen en cada momento tendremos que saber las asociaciones de líneas de entrada con canales de color de la imagen y algoritmos usados.

c) PINTAR. La aplicación debe ser capaz de presentar mediante un dispositivo de visualización las imágenes generadas con las muestras de sonido.

La función de visualización de las imágenes es la etapa de presentación de resultados de nuestra aplicación, en la que las imágenes generadas deben poder visualizarse en la pantalla del ordenador o a través de un proyector y ser vistas por el usuario final como habíamos indicado en el diagrama de casos de uso.

De este requisito concluimos la necesidad de manejar el concepto de pantalla o lienzo final, donde visualizar la imagen.

d) SECUENCIAR. La aplicación debe manejarse con una amplia variedad de algoritmos de ubicación espacial de muestras.

El objetivo principal de la aplicación es experimentar con diferentes algoritmos de ubicación de muestras y por lo tanto es un requisito el poder manejar una gran variedad de algoritmos. Además también se pretende generar muchos resultados visuales combinado los algoritmos básicos. De manera que en un determinado momento se puedan aplicar varios algoritmos simultáneos y podamos usar muestras de sonido actuales y también pasadas. Esto conlleva guardar un histórico de muestras de sonido por línea y como ya hemos indicado tener la máxima flexibilidad a la hora de establecer las correspondencias entre muestras y canales de video. Si además le sumamos que la aplicación puede generar varias imágenes, estas finalmente se podrán combinar también entre sí.

En el diagrama de casos de uso, hemos indicado como una función principal que el técnico establezca una secuenciación de algoritmos. Esta función aparece por un lado para cubrir la necesidad de poder analizar diferentes algoritmos sin tener que parar cada vez la aplicación, y para poder usar la aplicación como herramienta de visualización en conciertos, *performance* o proyectos escénicos en directo. La idea es poder realizar una programación o secuenciación en el tiempo de los algoritmos.

De este requisito concluimos que tendremos que definir un pequeño lenguaje para explicitar en cada instante de tiempo:

- Los algoritmos básicos de ubicación de muestras,
- El número de imágenes a generar.
- Los *buffers* de muestras.
- La asociación de muestras a canales en las imágenes.
- Etiquetas temporales para saber cuándo pasan las cosas.

De este lenguaje tendremos que obtener una lista de secuenciación. Y como esa lista se tendrá que ir recorriendo de acuerdo al tiempo, tendremos que manejar el concepto de tiempo o reloj.

e) CONTROL. La aplicación debe ser dependiente del tiempo real

En el diagrama de casos de uso una de las funciones del técnico es poner en marcha el proceso de captura, generación de imagen y visualización, y pararlo. Esto nos indica la alta dependencia del sistema con el tiempo, el sistema debe dar respuesta lo más rápido posible a las entradas. Se trata de un proceso continuo basado en el tiempo porque su entrada, la señal sonora, es un medio basado en tiempo y la salida una secuencia de imágenes representativas del sonido también.

Los sistemas basados en tiempo real deben responder a estímulos externos en un tiempo finito y especificado. Su funcionamiento no depende únicamente de que los resultados sean correctos, sino también del tiempo en el que se producen. En nuestra aplicación el tiempo de respuesta visual ante la llegada de un estímulo sonoro debe ser lo más rápido posible.

Para entender mejor la relación temporal que se produce entre el sonido y la imagen podemos usar como símil inverso el fenómeno de los relámpagos y los truenos. En éste fenómeno a partir de una descarga eléctrica se genera un efecto visual que es el relámpago y otro sonoro que es el trueno, como la velocidad de la luz es mayor que la del sonido, cuando nos encontramos alejados del fenómeno se produce un desfase entre la percepción sonora y visual, viendo primero el efecto visual y luego el sonoro. Conforme el fenómeno se encuentra más cerca de nosotros la simultaneidad entre ambos es mayor.

En el entorno que proponemos la percepción visual y la sonora deberían ser casi simultáneas y decimos casi, puesto que siempre se va a producir cierta latencia o retraso entre ambos. Por lo tanto el tiempo que pasa desde que se captura el sonido hasta que se genera la imagen correspondiente debe ser lo más pequeño posible, o lo que es lo mismo debemos intentar que la latencia sea la mínima posible.

De este requisito concluimos que la aplicación seguirá un ciclo de vida típico de los procesos dependientes del tiempo, donde tendremos los estados de ejecución, pausa y parada. Y para manejar esa dependencia del tiempo tendremos que definir un reloj interno a la aplicación que marque el tiempo de ejecución. Estas conclusiones vienen a corroborar necesidades que ya habíamos obtenido en los requerimientos anteriores.

f) La aplicación una vez programada debe ser autónoma.

Este requisito está íntimamente relacionado con el requisito d), que establece la necesidad de poder usar una gran variedad de resultados visuales y propone un lenguaje sencillo. La idea de autonomía supone posibilitar el almacenamiento de la secuenciación de cambios visuales, para posteriormente activarlo cuando la aplicación se ponga en ejecución.

g) Debe ser sencillo incorporar nuevos algoritmos de ubicación espacial de muestras

La aplicación debe proporcionar un mecanismo para introducir nuevos algoritmos sin necesidad de plantear muchos cambios en la aplicación.

h) INTERFAZ DE USUARIO. La aplicación debe proporcionar al usuario un entorno amigable y separado lo más posible de la lógica[98] de la aplicación.

Un requisito importante de la aplicación y la mayoría de aplicaciones orientadas a usuarios finales es que la interfaz con el usuario sea lo más sencilla y clara posible. Además se debe buscar la máxima independencia entre la interfaz de usuario y la lógica del programa, de manera que futuros cambios en la interfaz con el usuario no afecten a la lógica del programa.

De este requisito concluimos que tendremos que manejar en la aplicación el concepto de interfaz de usuario.

[98] Se entiende por lógica de la aplicación la parte que se encarga de codificar las reglas del negocio.

5.2 Diseño de la aplicación

Una vez hemos definido el problema que queremos resolver, a través de la fase de análisis es el momento de plantear, ¿cómo solucionarlo?. La metodología que vamos a seguir como en la fase anterior no va a ser del todo rigurosa y estará basada en el modelo orientado a objetos. Al elegir la metodología orientada a objetos damos por hecho que el lenguaje de programación que usaremos para la parte de implementación será también un lenguaje orientado a objetos. De hecho es una de las decisiones que tuvimos claras desde el principio que fue el usar el lenguaje de programación orientado a objetos JAVA para la aplicación, motivada fundamentalmente por la experiencia en el uso del lenguaje.

El lenguaje de programación Java

El lenguaje de programación Java es un lenguaje generalista basado en el paradigma orientado a objetos, todos los elementos del lenguaje salvo los tipos básicos son objetos que pertenecen a clases. Las clases están organizadas de forma jerárquica, de manera que existe una clase principal o raíz que es la clase `Object`[99] y todo el resto de clases son especializaciones de dicha clase. Es un lenguaje que ha tenido mucho éxito en aplicaciones distribuidas en internet.

Debido a su particular forma de compilación[100] los programas no son dependientes del sistema operativo. Esto se cumple siempre y cuando no usemos librerías[101] definidas exclusivamente para un

---

[99] A partir de este momento, cuando aparezca una palabra que tiene su contrapartida en el lenguaje de programación la mostraremos usando el tipo de letra Courier New.

[100] Compilar es el proceso de traducir el código escrito a código que entiende la máquina, denominado código máquina.

[101] Una librería o biblioteca es un conjunto de programas con una interfaz bien definida cuyo propósito es ser usada por otros programas.

sistema operativo como será nuestro caso. Actualmente junto al lenguaje de programación C es de los lenguajes más utilizados a nivel mundial.

El lenguaje al ser de propósito general puede abordar una gran variedad de problemas, incluidos el tratamiento de sonido e imagen. Otra de las cosas interesantes que sea un lenguaje ampliamente utilizado, es que además del conjunto de librerías que proporciona el propio lenguaje hay una gran cantidad de librerías especializadas, desarrolladas por la comunidad de programadores y accesibles de forma gratuita. Uno de los criterios que hemos seguido en todo el desarrollo de la aplicación ha sido el usar librerías libres, frente a las de pago.

Metodología orientada a objetos

La metodología orientada a objetos es la que genera menor distancia conceptual entre el mundo real a modelizar y la aplicación. Lo que tenemos que hacer es, a partir de los requisitos que hemos detectado en la fase anterior, identificar objetos, procesos u operaciones y mensajes que modelicen el sistema. Los objetos se agrupan en clases que a su vez se pueden especializar. Los procesos son acciones que deben realizar los objetos y las comunicaciones se establecen realizando las invocaciones de los procesos de las clases.

Lo primero es modelizar a grandes rasgos la estructura de la aplicación, identificando de esta forma los principales módulos, y posteriormente ir realizando refinamientos sucesivos hasta llegar a la fase de implementación. Esta forma de abordar la solución del problema por distintos niveles de complejidad en informática se denomina abstracción. Este principio aísla la información no relevante en un determinado nivel de la solución. Es interesante resaltar que el

movimiento de arte abstracto se base en los mismos principios que el diseño de programas.

La propia etapa de diseño tiene diferentes niveles de detalle. Los niveles más altos describen los módulos o componentes principales y los más bajos los detalles de cada uno de los módulos.

Por lo tanto partiremos de grandes conceptos o módulos principales de la aplicación y poco a poco iremos realizando refinamientos hasta encontrar los objetos y clases ya directamente traducibles en el lenguaje de programación. En la etapa de diseño es difícil saber hasta qué nivel de detalle llegar siendo su frontera con la implementación una zona poco definida. Lo que si resulta evidente es que tras la fase de implementación debemos tener un código de la aplicación que se pueda ejecutar.

Estructura de la aplicación

A grandes rasgos y teniendo en cuenta el requisito h), tendremos dos grandes bloques. Por un lado todas aquellas clases, objetos y procesos relacionados con la interfaz con el usuario, gráfica o no, y luego otro bloque grande que sería lo que denominamos la lógica de la aplicación (ver figura 5.2.1).

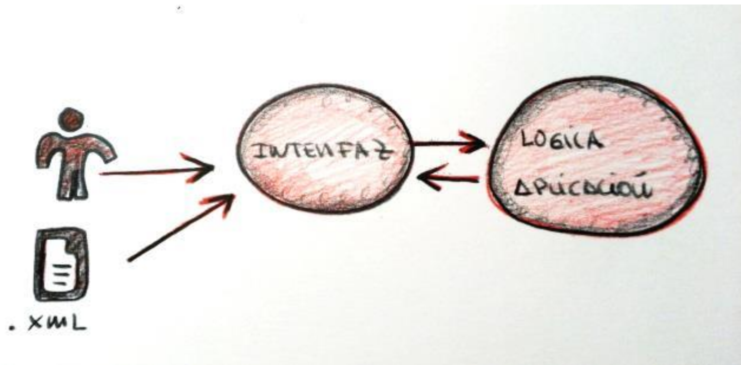


Figura 5.2.1. División lógica de la aplicación

El objetivo principal de la interfaz de usuario es establecer el puente entre el usuario y la lógica de la aplicación, de forma que todos sus componentes únicamente deben desarrollar esa función. La lógica del programa es la aplicación en sí, y se relaciona con el usuario a través de la interfaz. En todo el proceso de diseño es muy importante mantener una visión modular del sistema hasta llegar a los conceptos que formarán las clases y son los módulos básicos que tendremos que programar.

Siguiendo con la descomposición, la lógica de la aplicación la vamos a dividir de acuerdo a los principales requisitos funcionales que hemos detectado en el apartado anterior:

- Captura y almacenamiento del sonido
- Generación de la imagen.
- Control de todo el proceso: control de los estados y control del tiempo.

Todo el bloque relacionado con la configuración, captura y almacenamiento del sonido lo vamos a agrupar en el **módulo de sonido**. Todo lo relacionado con la generación de la imagen que

conlleva el acceso a los buffers de sonido y la selección en cada momento de los parámetros de visualización y por último la visualización de las imágenes lo agrupamos en el **módulo de imagen**. Por último tendremos el **módulo de control** que será el encargado de controlar los procesos anteriores y por lo tanto todos los estados de la aplicación.

En lo que resta del capítulo presentaremos el conjunto de decisiones de diseño que hemos considerado en cada uno de los módulos de la aplicación, obteniendo como resultado el conjunto de clases que formarán la aplicación. Empezaremos por el módulo de control ya que es el más general y nos da una visión en conjunto de la aplicación. Luego pasaremos a detallar el **módulo de la interfaz de usuario** que nos ofrece la visión que el usuario va a tener de la aplicación, a continuación presentaremos el módulo de sonido como primer paso para la generación de la imagen y por último la parte final del proceso de visualización: secuenciación de decisiones de visualización, generación de imágenes y visualización de las imágenes. En la figura 5.2.2 vemos un esquema general de los principales módulos de la aplicación, y cómo se relacionan entre sí.

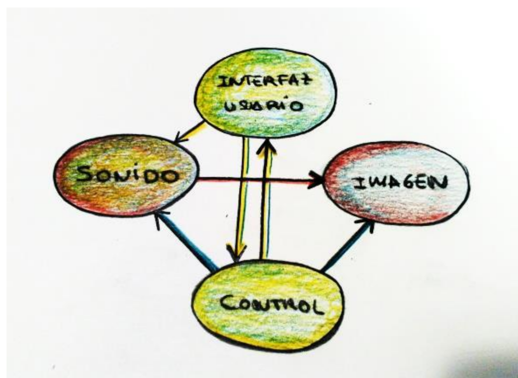


Figura 5.2.2. Módulos principales de la aplicación

5.2.1 Módulo de control

La idea de control es inherente a cualquier aplicación. De los requisitos observados en el apartado anterior hemos visto que se deberá configurar el sonido, se deberán definir las variaciones que sufrirá la visualización a lo largo del tiempo, tendremos que poner en marcha la visualización, tendremos que pararla, podremos también pausarla, tendremos que finalizar la visualización y la aplicación y tendremos que controlar el tiempo. Cada una de estas acciones repercutirá en diferentes módulos. La mejor forma de controlar estos diferentes procesos es definiendo un ciclo de vida propio de aplicación formado por diferentes estados. Salvo que el problema a resolver sea muy complejo, normalmente con pocos estados podemos manejar el control de la aplicación. Esta estrategia de diseño se utiliza en todas las aplicaciones.

En la figura 5.2.3 vemos los distintos estados por los que va a pasar la aplicación. Las flechas establecen el orden y la dirección de los cambios de estado.

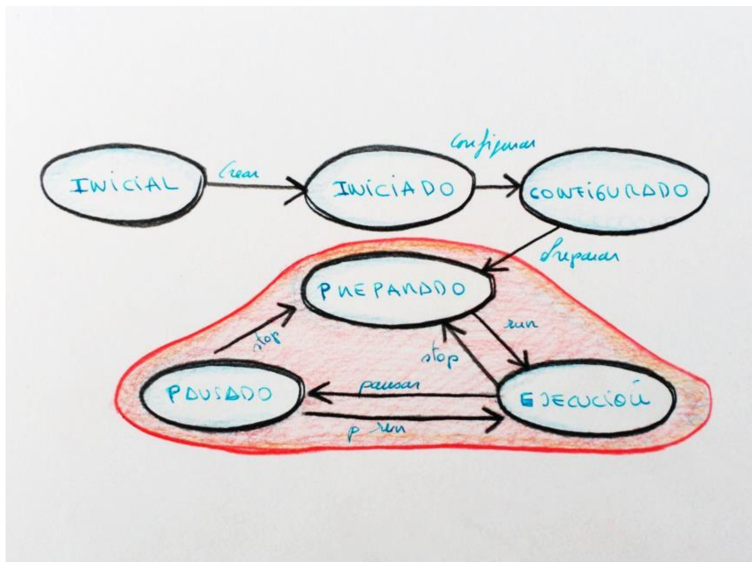


Figura 5.2.3 Ciclo de vida de la aplicación

Estado INICIAL[102]

Este estado representa el inicio de la aplicación todavía no ha pasado nada. Podríamos decir que es el estado de reposo. A este estado llegamos cuando arranca la aplicación o cuando desde cualquier estado queremos terminar la aplicación, a través de la acción de `finalizar()`. De esta forma INICIAL también lo podemos ver como el estado final.

Estado INICIADO

Para llegar a este estado previamente tendremos que estar en el estado de INICIAL. Cuando creamos los primeros objetos de la aplicación, por ejemplo como es nuestro caso los objetos de control de los módulos, la aplicación pasa al estado de INICIADO. La acción que conduce a este estado es la de `crear()` [103]. En la mayoría de los procesos este paso no cambiará nada.

Estado CONFIGURADO

La acción que conduce a este estado es la de `configurar()` y previamente deberemos estar en el estado INICIADO. Prácticamente todas los procesos de la aplicación requieren de una configuración. En el módulo de sonido deberemos especificar la tarjeta de sonido y el protocolo de comunicación.

---

[102] En nomenclatura inglesa se suele denominar *UNLOAD*, que viene a ser algo así como todavía no activado.

[103] Los paréntesis de la acción sugieren que dicha acción se convertirá en un método en la fase de implementación. Para remarcar esta situación modificamos el tipo de letra que pasará a ser Courier New. El motivo es que la tipología Courier ha sido de forma histórica la que se ha usado para escribir código en los textos de programación.

Estado PREPARADO

En ocasiones nos resultará más adecuado proporcionar a la aplicación un estado de configuración más avanzado como es el de PREPARADO. Básicamente es un remate final a la etapa de configuración y también nos servirá como estado de reposo cuando estemos en la actividad principal de la aplicación. Para llegar a este estado realizaremos la acción de `preparar()`. Podemos llegar a este estado de diversas formas bien porque en el estado previo estábamos en el estado CONFIGURADO o estábamos en el estado de EJECUCIÓN o en el estado de PAUSADO (ver figura 5.2.3)

Estado EJECUCIÓN

Este estado indica que la aplicación está capturando sonido a través de la tarjeta de sonido y visualizando las imágenes a través de la tarjeta de video. Para llegar a este estado deberemos realizar la acción de `ejecutar()` y previamente deberemos estar en el estado PREPARADO o en el estado PAUSADO.

Estado PAUSADO

Este estado indica un estado de reposo temporal, el tiempo de la secuenciación se ha parado, pero no se ha inicializado y todo esta como se quedó en el estado anterior. Para pasar a este estado tendremos que realizar la acción `pausar()` y previamente deberemos estar en el estado de PREPARADO o de EJECUCIÓN. Realmente se suele permitir pausar el estado de preparado, aunque no tendrá ningún efecto especial.

En cada momento de la ejecución de la aplicación se podrá consultar el estado de la aplicación, con lo que tendremos que manejar el concepto de `Estado`. A su vez necesitaremos una entidad que se

encargará de manejar los estados de la aplicación, a esta entidad la vamos a denominar `ControlProcesos`, su quehacer será llevar el control de los estados y provocar los cambios de estado de la aplicación. Puesto que cada cambio de estado va a suponer realizar un montón de tareas en cada uno de los módulos para simplificar la funcionalidad de `ControlProcesos` se ha introducido un control individual para cada uno de los proceso principales :

- `ControlCapturar`,
- `ControlGenerar`,
- `ControlContar`,
- `ControlSecuenciar`.

Para cada uno de estos controles tendremos las mismas acciones que para el proceso general, esto es `crear()`, `configurar()`, `preparar()`, `ejecutar()`, `parar()`, `pausar()` y `finalizar()`, para poder realizar los correspondientes cambios de estado.

Como estrategia general de diseño hemos optado por definir una clase de configuración en cada uno de los módulos y ubicadas en sus respectivos módulos, así tendremos `ConfiguraciónImagen`, `ConfiguracionSonido`, `ConfiguracionInterfazUsuario` y `ConfiguracionControl`.

La clase configuración `ConfiguracionControl` además de tener referencias[104] a los objetos principales del módulo, como es el `ControlProcesos`, tendrá una referencia a cada una de las configuraciones de los módulos de la aplicación. Esto posibilita

[104] En Java un identificador es la dirección de memoria donde se encuentra un determinado objeto, que es una instancia de una clase.

acceder a cualquier elemento de configuración desde cualquier parte de la aplicación.

Resulta interesante resaltar que en muchas ocasiones y este es el caso de `ConfiguracionControl`, solo tendremos un objeto de dicha clase. El modelo orientado a objetos supone crear objetos que pertenecen a clases aunque estos sean únicos.

Según los requisitos e) y d) otro de los elementos que necesitamos diseñar es un `Reloj`, de manera que cuando se ponga la aplicación en estado de ejecución empiecen a contar los segundos y cuando pause la aplicación se detenga y cuando se pare se ponga a cero. Una características de este objeto y de lo que todavía no hemos comentado es que la aplicación como es normal tendrá tantos hilos de ejecución independientes como por lo menos procesos hemos detectado en los requerimientos funcionales.

Para aclarar un poco el concepto de hilo de ejecución, diremos que cuando el programa se ejecuta se crea el hilo de ejecución principal, que por regla general lanza la interfaz gráfica del programa y se queda esperando a que ocurran eventos. Cuando llegan lanza a ejecución la parte de código que esté acordada. Además de ese hilo de ejecución principal los lenguajes de programación tienen previstos mecanismos para poder lanzar más hilos independientes, a esta parte de la programación se la denomina programación concurrente ya que posibilita que varios hilos de ejecución se ejecuten al mismo tiempo.

Como criterio de diseño de la aplicación, de entre las posibilidades que ofrece el lenguaje para generar hilos independientes, nosotros para toda la aplicación hemos optado por el mecanismo de tarea y programador de la tarea. El concepto es sencillo definimos un objeto que contiene una tarea que se debe repetir un determinado número

de veces y mediante otro objeto programador de la tarea, la ponemos en marcha indicándole cada cuanto tiempo debe repetirse. En la librería `java.util` que proporciona Java, estas dos acciones las realizamos mediante las clases `Timer` y `TimerTask` respectivamente.

Cuando trabajamos con hilos es habitual tener que establecer algún tipo de sincronización entre ellos. Esto nos conducirá a tener que crear objetos para controlar el acceso a determinados recursos de la aplicación.

Finalmente incorporaremos también en este módulo una clase muy sencilla que únicamente lanzará a ejecución la aplicación, `LanzarAplicacion`. En Java un programa siempre se empieza a ejecutar por el método `main()`, que es un método especial.

En la figura 5.2.4 podemos ver un gráfico con el conjunto de clases que hemos definido en el módulo de control.

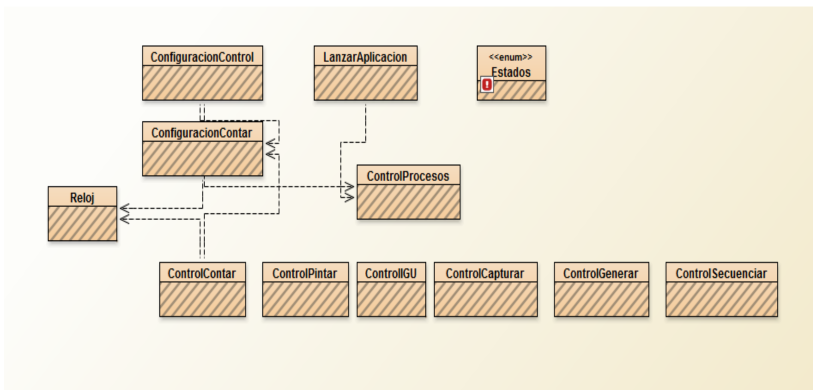


Figura 5.2.4. Esquema de clases del módulo de control

5.2.2 Interfaz de usuario

La interfaz de la aplicación con el usuario es una de las partes más importantes del desarrollo de una aplicación. El usuario normalmente no conoce los detalles de la implementación y ni si quiera el lenguaje de programación con el que ésta programada, pero sí conocerá la funcionalidad de la aplicación a través de la interfaz. El usuario busca en la interfaz cualidades de amigabilidad, sencillez y usabilidad, incluso si se trata de una herramienta no orientada a un público general, esas características siempre son apreciadas.

Inicialmente pensamos en abordar toda la interacción a través una interfaz gráfica y de hecho como ya hemos comentado se planteó un diseño inicial prototipo en el que preveíamos estas cuestiones. Posteriormente decidimos diseñar parte de la interacción de forma textual en vez de gráfica, con la finalidad de no introducir una complejidad adicional en el desarrollo de la aplicación. Esto no quiere decir que en un futuro se diseñe la totalidad de la interacción mediante gráficos, pero para este trabajo se optó por una solución intermedia.

Con esto lo que tendremos es que el usuario no accederá a la aplicación sólo de forma gráfica sino también de forma textual.

Como veíamos en el apartado 5.1, en el diagrama de casos de uso, el usuario interactúa con el sistema de cuatro formas diferentes:

- Para configurar el sistema de sonido
- Para secuenciar la visualización
- Para cambiar los estados de la aplicación
- Para visualizar el resultado

Empezaremos por presentar las decisiones de diseño de la parte textual de la interfaz, que se corresponden con las funcionalidades de configurar el sistema de sonido y secuenciar la visualización.

5.2.2.1 Interfaz de usuario para configurar el sistema de sonido

Para configurar el sonido de la aplicación tendremos dos acciones: una física que consistirá en realizar el conexionado de los instrumentos con la tarjeta de sonido y de la tarjeta de sonido con el ordenador y otra para decirle a la aplicación datos de configuración del sonido.

En un principio como se realiza en la mayoría de aplicaciones, se pensó en diseñar una serie de componentes gráficas, para que el usuario realizara las elecciones oportunas con respecto al sonido. Más tarde se cambió de idea con la finalidad de simplificar al máximo la interfaz gráfica. Para introducir la configuración del sonido vamos a utilizar el lenguaje de etiquetas XML[105]. Esta estrategia de usar un fichero de texto estructurado mediante etiquetas es ampliamente usada en programación, de hecho en la mayoría de las aplicaciones se usa como almacén interno de la aplicación. Esto quiere decir que aunque el usuario introduzca los datos de forma gráfica, la aplicación organizará mediante etiquetas dichos datos y generara un fichero interno XML de consulta. Un usuario experimentado puede acceder y modificar directamente dichos ficheros sin tener que introducir los datos desde la interfaz gráfica. Siguiendo esta línea de actuación, la configuración del sonido y como veremos más adelante la configuración de la secuenciación, se introducirá mediante un fichero en formato XML.

[105] XML, es le acrónico de *eXtensible Markup Languaje*, se trata de un lenguaje de marcas desarrollado por World Wide Consortium.

Para crear un fichero XML, solo necesitamos un editor de texto básico, disponible en todos los sistemas operativos. Lo que hay que tener cuidado es en exportar dicho fichero con la extensión `.xml`. No obstante existen aplicaciones específicas para la ayuda en la escritura de ficheros de este tipo. El lenguaje XML es muy sencillo y está formado por elementos que se señalan con etiquetas, atributos de dichos elementos y los datos. En (Quin 2005)[106], podremos consultar la especificación del lenguaje. A continuación vemos un ejemplo de la configuración del sonido:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuracionSonido>
	<driver>ASIO4ALL v2</driver>
	<buffer>4096</buffer>
	<numBuffer>4</numBuffer>
	<frecuenciaMuestreo>96000</frecuenciaMuestreo>
	<lineaEntrada>1</lineaEntrada>
	<lineaEntrada>2</lineaEntrada>
</configuracionSonido>
```

La etiqueta principal es `<configuracionSonido>`, e indica que todo lo que quede dentro de ella será la configuración del sonido. El siguiente elemento `<driver>`, es el *driver* que vamos a usar para conectar el programa con la tarjeta de sonido. Con independencia del dato que pongamos en esta etiqueta esta es la única información que pediremos al usuario mediante la interfaz gráfica, con lo que hemos mantenido la etiqueta pero realmente no se tiene que poner nada. Esto es debido a que el nombre con el que internamente el sistema operativo identifica al *driver,* el usuario no tiene porqué conocerlo.

[106] No sé muy bien cómo poner esta parte www.w3.org/XML/, esa es la página web.

El siguiente elemento es la etiqueta `<buffer>` en la que debemos indicar el número de elementos que queremos que maneje la aplicación. El *buffer* es el contenedor principal para el intercambio de muestras de sonido entre la aplicación y la tarjeta de sonido. En la mayoría de los algoritmos de visualización cada uno de los *buffers* se debe corresponder con las dimensiones de una imagen. Se tiene que mantener la relación:

$$tamañoBuffer = n \times m$$

Donde n y m son el ancho y el alto de la imagen respectivamente. El tamaño del *buffer* siempre debe ser múltiplo de 2, y dependiendo del *driver* de la tarjeta de sonido hay casos en los que no se puede seleccionar cualquier tamaño y hay que ceñirse a unos determinados valores. Como estrategia de diseño y tratando de buscar la máxima generalidad vamos a poder trabajar únicamente con dos tamaños de *buffer*: 1024 y 4096. De forma que si trabajamos con un buffer de 1024 estas muestras generarán una imagen de 32x32, y si el buffer es de 4096 entonces tendremos una imagen de 64x64.

El que la relación de aspecto de las imágenes sea 1:1, en vez de usar relaciones panorámicas como es habitual en el ámbito audiovisual es una cuestión particular, ya que no se debe presuponer que la visualización se realice según el modelo de la televisión y el cine. Es por eso que hemos decidido generar matrices cuadradas y relegar las opciones de relación de aspecto, a las transformaciones de la imagen en el lienzo. Mediante un simple escalado podremos modificar la relación de aspecto.

El siguiente elemento es `<numBuffer>` que representa al número de *buffers* históricos queremos mantener. Este parámetro es importante para poder desarrollar algoritmos de visualización complejos, donde

no se utilizan únicamente las muestras actuales, sino también las muestras pasadas.

La etiqueta `<frecuenciaMuestreo>` contiene la frecuencia con la que la tarjeta de sonido va capturando las muestras que serán almacenadas en los *buffers*. Para saber cada cuanto tiempo estará lleno el buffer, sólo tendremos que dividir el tamaño del buffer por la frecuencia de muestreo, por ejemplo si el tamaño del buffer es 1024 y la frecuencia de muestreo es de 44100, tendremos un *buffer* cada 23 ms. Si cada 23 ms generamos una imagen y la mostramos tendremos una tasa de aproximadamente 43 imágenes por segundo. Una buena relación se establece con un tamaño de *buffer* de 4096 y una frecuencia de muestreo de 96000 Hz, estos datos nos conducen a una tasa de casi 24 imágenes por segundo.

Por último lo que tenemos es una lista de etiquetas de la forma `<lineaEntrada>`, cuyo dato es el número de entradas que vamos a considerar en la aplicación. La restricción en este caso que tenemos es que no se pueden poner líneas de entrada que no tenga la tarjeta de sonido.

El manejar la información referente a las líneas supondrá diseñar en la aplicación el concepto de `LineaEntrada`.

La lectura y generación de los datos de programa de este fichero de configuración de sonido se tratará en el módulo de sonido. El usuario escribe el fichero y el módulo de sonido lo interpretará. Aunque lo recordaremos en el módulo de sonido, la interpretación del fichero xml conllevará la creación de un objeto de la clase `ConfiguracionSonido` y para leer los datos de este fichero, actualizarlo y gestionarlo tendremos la clase `ConfiguracionSonidoXML`. En un diseño más estricto la

funcionalidad de la clase `ConfiguracionSonidoXML` debería estar dentro de `ConfiguracionSonido`.

5.2.2.2 Interfaz de usuario para configurar la visualización

Otro de los datos importantes que debe suministrar el usuario al programa es la configuración de la visualización. Se tienen que suministrar datos referentes al lienzo, las imágenes que se van a generar y las variaciones a lo largo del tiempo de los algoritmos de visualización empleados. En un principio se planteó la posibilidad de diseñar un entorno gráfico para introducir estos datos, pero ante la dificultad que presentaba, se optó como en el caso de la configuración de sonido por una versión textual, usando el lenguaje XML, quedando esta posible interfaz gráfica como trabajo futuro.

Por un lado tendremos que introducir un conjunto de datos que vamos a denominar estáticos. Y por otro lado, como consecuencia de los requisitos d) y e), tendremos los datos de las imágenes que queremos que evolucionen a lo largo del tiempo como son: las líneas de sonido usadas y su asociación a los canales de color, el algoritmo de ubicación de muestras y el método de fusión entre muestras.

Como datos estáticos tendremos:

- Dimensiones del lienzo final.
- Posición del lienzo en el dispositivo de visualización.
- Imágenes que se visualizarán en el lienzo. Para cada una de las imágenes:
    - Dimensiones.
    - Posición respecto al lienzo.
    - Escalado.
    - Rotación.

En la figura 5.2.5 vemos gráficamente un esquema de convivencia de varias imágenes dentro de un lienzo.

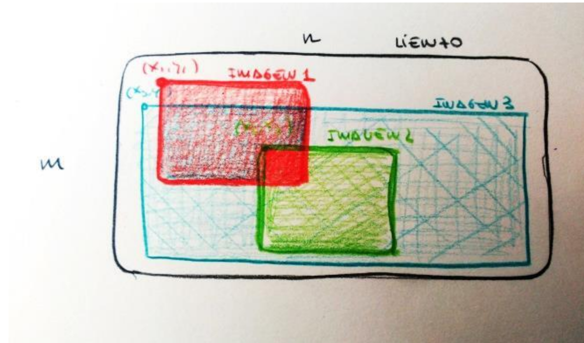


Figura 5.2.5. Esquema de lienzo final con varias imágenes

Y para cada una de las imágenes podrán variar a lo largo del tiempo los siguientes parámetros:

- Líneas de sonido cuyas muestras se visualizarán, donde para cada una de las líneas:
    - Identificador de la línea.
    - El *buffer* que vamos a usar de la línea
    - Algoritmo que vamos a usar,
    - Canales a utilizar
    - Método de fusión a usar con la imagen anterior.

Lo primero que tenemos es un dispositivo de visualización, que en el contexto digital que nos encontramos se corresponde con una tarjeta de video. El tamaño máximo del lienzo y la forma con la que se verá, dependerá de las características de la tarjeta. Aunque la tarjeta de video permita conectar varios monitores o proyectores al mismo tiempo desde la aplicación solo veremos un único dispositivo, cuya dimensión será la suma de las resoluciones de los monitores o

proyectores. El usuario puede discernir entre los dos monitores en base a colocar las imágenes en la posición adecuada del lienzo.

El tamaño que hemos decidido fijar para las imágenes es 32x32 o 64x64, debido a la relación que tienen estos números con los tamaños de los *buffers* 1024 y 4096. No obstante como vemos en la figura 5.2.5, en la última fase, cuando pintamos la imagen en el lienzo, hemos permitido realizar unas transformaciones básicas de la imagen con la finalidad de poder adecuarla al lienzo. En definitiva la idea es poder acomodar las imágenes al espacio donde se realiza la visualización. Las transformadas son las básicas en imagen 2D: posición, rotación (que supondrá determinar un punto como eje de rotación) y escalado.

Desde un punto de vista conceptual estas transformaciones básicas no alteran la esencia de la imagen original puesto que el cambio de posición no afecta a la estructura de la imagen, la rotación funciona como una última variación en la ubicación de las muestras y el escalado supone un duplicado o eliminación de muestras. Con lo que una imagen después de estas alteraciones al menos en lo que concierne a su relación con el sonido sigue manteniendo su esencia.

Para cada una de las imágenes los parámetros variables en el tiempo son las líneas de sonido que intervienen y para cada línea el *buffer* involucrado, el algoritmo de ubicación, los canales a los que afectan las muestras de la línea y el método de fusión entre muestras. Para expresar la variación temporal de estos parámetros hemos incluido una marca temporal que indique en que momento de la visualización se deben considerar estos datos y durante cuánto tiempo. La marca temporal la colocaremos asociada a la línea de entrada y *buffer* implicado.

El introducir el buffer ha estado motivado por el hecho de querer mostrar también estados anteriores de los *buffers*. De forma que una línea estará etiquetada con el número del *buffer* que queremos visualizar. La numeración de los *buffers* está relacionada con la cercanía o alejamiento del momento actual y con el número de *buffers* históricos que hemos indicado en la configuración del sonido.

El algoritmo de ubicación de muestras lo que hace es colocar de acuerdo a un plan, cada muestra de sonido en la imagen. Haremos referencia a algoritmos concretos a través de su número, por ejemplo el número 1 es el más simple de todos que obedece a la estrategia de ir colando las muestras de izquierda a derecha y de arriba abajo.

De la misma forma que los algoritmos, los canales de color $(r, g, b, alfa)$, estarán numerados del 1 al 4.

Otro dato que debemos considerar es el método de fusión que usaremos cuando vayamos a pintar una muestra en una posición que ya ha sido pintada. Por regla general lo que haremos, y eso es lo que se corresponde con el método de fusión 1, es sumar los valores.

En la figura 5.2.6, vemos gráficamente lo que serían dos nodos de la secuenciación de la línea 1 con el buffer 1.

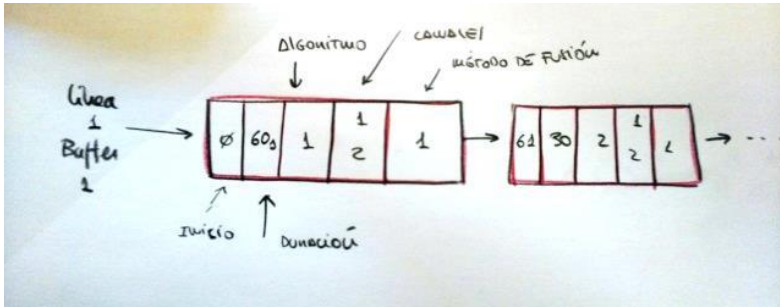


Figura 5.2.6. Ejemplo de nodos de secuenciación.

Cada nodo de secuenciación nos informa del instante de tiempo en el que se debe activar, en el ejemplo de la figura 5.2.6, el primer nodo está en el instante 0, con una duración de 60s, el algoritmo es el 1, los canales el 1 y el 2 (esto es en el rojo y el verde) y el método de fusión es el 1. En la figura 5.2.7, vemos un ejemplo más completo con dos imágenes.

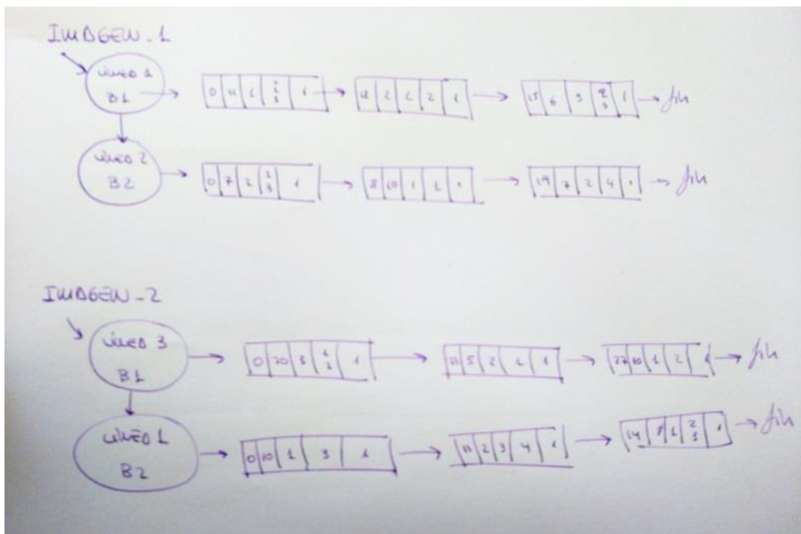


Figura 5.2.7. Ejemplo completo de nodos de secuenciación.

Para que el usuario pueda introducir todos estos datos relativos a la configuración de la imagen, como en el caso del sonido hemos diseño un pequeño lenguaje de etiquetas en XML. El usuario escribirá el fichero y la aplicación en el módulo de imagen lo leerá y generará los objetos oportunos para el funcionamiento de la aplicación.

A continuación vemos un ejemplo del fichero XML con cada una de sus etiquetas:

```
<?xml version="1.0" encoding="UTF-8"?>
<lienzo id="secuenciacion1">
<dimension>
<ancho>1680</ancho>
<alto>1050</alto>
</dimension>
            <posicion>
<punto>
<x>0</x>
<y>0</y>
</punto>
</posicion>
<imagen id="1">
<dimension>
<ancho>64</ancho>
<alto>64</alto>
</dimension>
<transformadas>
<puntoAnclaje>
<punto>
<x>0</x>
<y>0</y>
</punto>
</puntoAnclaje>
<posicion>
<punto>
<x>0</x>
<y>0</y>
```

```
</punto>
</posicion>
<escala>
    <dimension>
<ancho>100</ancho>
<alto>100</alto>
</dimension>
</escala>
<rotacion>0</rotacion>
</transformadas>
<lineaSonido id="1" numBuffer="1">
<nodoTiempo>
<inicio>0</inicio>
<duracion>6000</duracion>
<algoritmo>2</algoritmo>
<canal>2</canal>
        <metodoFusion>1</metodoFusion>
</nodoTiempo>
                  </lineaSonido>
                  <lineaSonido id="2" numBuffer="1">
<nodoTiempo>
<inicio>0</inicio>
<duracion>6000</duracion>
<algoritmo>1</algoritmo>
<canal>1</canal>
<metodoFusion>1</metodoFusion>
</nodoTiempo>
                  </lineaSonido>
            </imagen>
</lienzo>
```

La etiqueta <lienzo> engloba a todas las demás etiquetas y hace referencia al lienzo de la visualización. De alguna forma el contenido de este fichero XML lo que indica es cómo queremos que sea nuestro lienzo.

La etiqueta `<dimensión>` con sus elementos `<alto>` y `<ancho>`, indica las dimensiones del lienzo, también la usaremos para las dimensiones de las imágenes.

A continuación tendremos la etiqueta `<posición>` que indica la posición del lienzo con respecto al dispositivo de visualización. Esta etiqueta también la usaremos posteriormente.

Cada etiqueta `<imagen id="1">`, tiene como atributo el identificador de la imagen. La etiqueta <dimensión> indica la dimensión de la imagen. La etiqueta `<transformadas>` contiene a las etiquetas `<puntoAnclaje>`, `<escala>`, `<rotación>` y `<posición>`. Para cada línea de sonido tenemos un conjunto de nodos temporales con la etiqueta `<nodoTiempo>` en los que indicamos: el inicio con la etiqueta `<inicio>`, la duración con la etiqueta `<duracion>`, el algoritmo usado con la etiqueta `<algoritmo>`, el conjunto de canales a los irán dirigidas las muestras, mediante la etiqueta `<canal>` y por último el método de fusión empleado mediante la etiqueta `<metodoFusion>`.

La lectura y generación de los datos de programa de este fichero de configuración de imagen se tratará en el módulo de imagen. El usuario escribe el fichero y el módulo de imagen lo interpretará. Aunque lo recordaremos en el módulo de imagen, la interpretación del fichero XML conllevará la creación de la clase `ConfiguracionImagen` y para leer los datos de este fichero, actualizarlo y gestionarlo hemos definido la clase `ConfiguracionImagenXML`. En un diseño más estricto la funcionalidad de la clase `ConfiguracionImagenXML` debería estar dentro de `ConfiguracionSonido`.

Desde el punto de vista del diseño orientado a objetos, el contenido del fichero XML define los principales conceptos que la aplicación tendrá que manejar: `Lienzo`, `Imagen`, `NodoTemporal`, `TransformadasImagen` y `LineaSonidoGenerar`. Otros de los conceptos que aparecen en el fichero XML, los tiene ya el propio lenguaje desarrollados como es el caso de dimensión, en la librería `java.awt.geom`, tenemos la clase `Dimension`, y el concepto de punto del espacio de dos dimensiones mediante la clase `Point2D`, de la misma librería.

5.2.2.3 Componentes de la interfaz gráfica de usuario

Lo primero que vamos a describir es el conjunto de elementos de la interfaz gráfica que hemos diseñado sin resaltar su funcionalidad que veremos después.

Para la interfaz gráfica hemos tratado de buscar un diseño lo más sencillo y simple posible. Como es habitual en la mayoría de interfaces gráficas de programas hemos respetado las normas y signos de uso habituales a los que estamos acostumbrados.

La interfaz gráfica está delimitada por un marco principal, que se mantiene a lo largo de toda la interacción (ver figura 5.2.8). El marco no se puede redimensionar y contiene el logo de la aplicación, el nombre y los tres botones típicos de las ventanas, que son la minimización, maximización y el cierre.

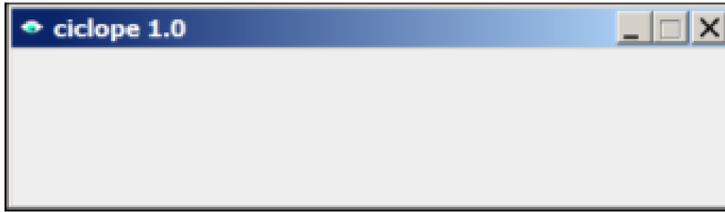


Figura 5.2.8 Marco principal de la interfaz gráfica.

Al menos en el lenguaje de programación Java, desde el punto de vista estético tenemos muy pocas posibilidades de actuar sobre la barra superior. La idea que maneja Java es que el aspecto visual de esa zona esté controlada por el sistema operativo, de manera que los colores y las formas de los botones serán los que tenga seleccionados el sistema operativo en ese momento. Esto es debido a que el sistema operativo también trata de mantener una uniformidad visual en todas las ventanas de las aplicaciones que se están ejecutando.

Puesto que la interacción gráfica del usuario es muy básica la dimensión del marco no es muy grande y su diseño va a estar inspirado en las interfaces de los reproductores de casete y *compact disk*, donde los elementos principales son los botones *play*, *stop*, *pause,…* (ver figura 5.2.9).

Figura 5.2.9. Reproductor de sonido Aiwa XR-EM400.

Lo primero que se decidió es que siguiendo la idea de botones de reproductores, el diseño de la interfaz gráfica se estructura alrededor del concepto de botón. Inicialmente se diseñó el modelo de botón inspirado en un botón clásico circular sustituyendo los agujeros para el hilvanado por imágenes icónicas de su funcionalidad (ver figura 5.2.10).

Figura 5.2.10. Modelo de botón clásico.

Con estas ideas en mente se diseñaron las primeras versiones de los botones como mostramos en la figura 5.2.11. Se optó por la sencillez del blanco y negro. Para remarcar más la idea original de botón circular se suavizaron los contornos de los iconos clásicos de reproducción de audio, de manera que se integraban mejor en el círculo. Se diseñaron versiones contrarias con fondo negro e icono blanco, pero al final se optó por la primera versión. El desarrollo de los botones se realizó usando el software After Effects[107] que aunque no es específico de imagen estática también se puede usar para tal fin.

[107] After Effects es un software comercial de la empresa Adobe, orientado a realización de efectos digitales de cine y video.

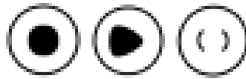


Figura 5.2.11. Primeras versiones de los botones.

El siguiente elemento que se diseñó fue el logotipo de la aplicación. Como ocurre con todos los logotipos deben tener la fuerza suficiente para representar adecuadamente a la aplicación. El paso previo que ya teníamos era el nombre de la aplicación, Ciclope, y al ser el elemento distintivo de los cíclopes el ojo, pues ese fue el concepto a desarrollar para el logotipo, el ojo. Los nombres de las aplicaciones en Informática muchas veces no son representativos de su funcionalidad. Un caso interesante es el nombre del lenguaje de programación Java. Java simplemente era el tipo de café que el equipo de desarrollo del lenguaje tomaba en los descansos, de ahí vino el logo del lenguaje que es una taza de café (ver figura 5.2.12).

Figura 5.2.12. Logotipo del lenguaje de programación Java.

Inicialmente dibujamos a mano varias versiones del ojo como vemos en la figura 5.2.13 y finalmente con estas ideas se diseñó el logotipo definitivo, basado de nuevo en la sencillez y simplicidad. A diferencia de los botones anteriores para el logotipo decidimos poner el color azul. Esto fue motivado principalmente por el hecho que el icono de una aplicación debe convivir en la barra de herramientas del sistema operativo, con el resto de las aplicaciones iniciadas. El color azul hace que se pueda distinguir con mayor claridad del resto de los

programas. En la figura 5.2.14 podemos ver cómo quedó el logotipo definitivamente.

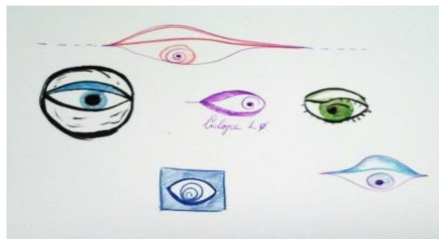


Figura 5.2.13. Bocetos iniciales del logotipo de la aplicación.

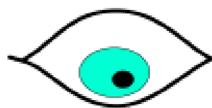


Figura 5.2.14. Logotipo de la aplicación.

Realmente quedamos muy satisfechos con el logo de la aplicación, aunque no se apreciaba demasiado debido a las pequeñas dimensiones con las que se inserta en la cabecera del marco principal (ver figura 5.2.8) y es por eso que decidimos modificar el diseño de los botones de la aplicación para que también representaran el concepto de ver de forma diferente que había inspirado el cíclope. Con lo que cambiamos el concepto de botón circular al de botón ovalado, en forma de ojo. Con respecto a los colores e iconos los dejamos igual. Los tres botones principales de la aplicación quedaron como muestra la figura 5.2.15.

Figura 5.2.15. Botones de reproducción.

En las interfaces gráficas los botones tienen tres estados, no seleccionado, pulsado y seleccionado. En muchas ocasiones solo es necesario manejar visualmente dos estados, seleccionado o no y pulsación. El estado de pulsación dura mientras el usuario aprieta el botón del ratón, puesto que en el momento que lo suelta ya hemos cambiado el estado. De esta forma un botón puede tener 3 imágenes representativas, una para el estado no seleccionado, otra para la pulsación y otro para el estado seleccionado.

La imagen que decidimos para el estado de pulsación o selección de los botones fue la misma para todos, un ojo cerrado (ver figura 5.2.16), significando de esta forma que habíamos elegido dicha opción. Para el estado de seleccionado, una opción era mantener la misma imagen de pulsación, el ojo negro. Pero al menos en el botón de *play*, su significado se alejaba de la acción que realmente estaba ocurriendo que es mostrar la visualización del sonido. Así que decidimos poner el ojo del logotipo cuando una de las tres acciones estuviera seleccionada.

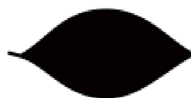


Figura 5.2.16. Imagen de botón pulsado.

Finalmente y siguiendo el concepto de botones de los reproductores de sonido se diseñaron dos botones más, para acceder a la configuración de la aplicación, que como veremos su interacción gráfica se ha simplificado mucho y otro para realizar confirmaciones, se corresponde con el botón típico de aceptación. En este punto pensamos en diseñar un botón de cancelación pero al final decidimos usar el botón de cierre del marco principal para realizar las cancelaciones. Esta decisión rompe con el uso habitual del botón de cierre. Para diseñar estos dos botones hemos seguido las mismas ideas de diseño que para los otros, y también usaremos como imagen de pulsación el ojo cerrado. El resultado visual de estos dos botones los podemos ver en la figura 5.2.17.

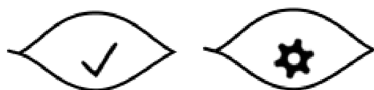


Figura 5.2.17. Botones de aceptación y configuración.

Para las etiquetas de la aplicación hemos usado el tipo de letra Seagoe UI[108], es un tipo de letra de corte humanista, simplemente la hemos elegido por su sencillez y claridad (ver figura 5.2.18).

Segoe UI

Figura 5.2.18. Tipo de letra Seagoe UI

[108] Seagoe es una familia tipográfica desarrollada por la empresa estadunidense Monotype Imaging, Inc. Uno de sus productos más conocidos es la fuente tipográfica Times New Roman.

Finalmente el otro elemento de interfaz gráfica que hemos usado es el combo box o cuadro combinado, se trata de un elemento típico de las interfaces gráficas que combina una lista desplegable con un campo de texto. Desde el punto de vista de diseño lo único que podemos seleccionar es su tamaño.

Para concluir esta primera parte donde hemos visto todos los elementos que vamos a considerar en la interfaz gráfica mostramos la figura 5.2.19, que de forma desordenada los contiene a todos.

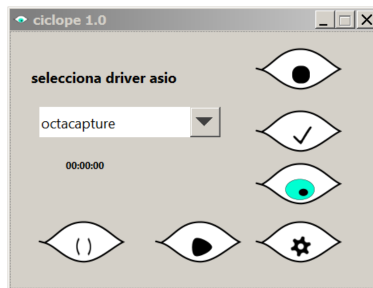


Figura 5.2.19. Marco con todos los elementos desordenados de la IGU.

5.2.2.4 *Interfaz gráfica de usuario para cambiar los estados de la aplicación*

Como hemos visto en el apartado 5.2.1, la aplicación funciona con 6 estados. El objetivo de la interfaz gráfica de entrada es dotar al usuario de un mecanismo sencillo para que pueda ir generando los cambios de estado de la aplicación. La introducción de datos más importante se ha diseñado mediante ficheros XML, esto supone poder simplificar al máximo la interfaz gráfica. Salvo para la introducción del nombre del driver que maneja la tarjeta de sonido, el resto de acciones de la interfaz de usuario están orientadas únicamente al

cambio. Desde la lógica de la aplicación será el módulo de control quien reparta las órdenes al resto de módulos.

La mayoría de lenguajes de programación proveen en sus librerías los componentes gráficos necesarios En Java los componentes gráficos los encontramos en las librería `java.awt` y `javax.swing`, donde esta última es una extensión de la primera.

El diseño de la interfaz gráfica es muy sencillo. El contenedor principal o marco contendrá los diferentes botones (ojos) que provocan los cambios de estado. La clase que hemos usado para el contenedor principal es `JFrame`, que de forma automática dispone de tres botones: cierre, minimizar y maximizar. Estos tres botones vienen con una programación básica que se puede alterar. El tamaño del contenedor principal es fijo y debe poder albergar como máximo tres botones. La idea que hemos desarrollado es que entrando (pulsando) en los ojos nos vamos acercando hacia la visualización y generamos los cambios de estado. La fase inicial consiste en prepararlo todo para empezar la visualización.

Cuando arranca la aplicación se presentará una pantalla inicial de bienvenida, que indica que estamos en el estado INICIAL, que también podemos entenderlo como el estado final (ver figura 5.2.20). Esta es una idea interesante de diseño puesto que el desarrollo de la aplicación es un recorrido circular. Solamente podremos cerrar la aplicación cuando estando en el estado INICIAL, pulsemos el botón de cierre del marco principal. Si estamos en un estado diferentes al INICIAL, al pulsar el botón de cierre pasaremos al estado INICIAL.

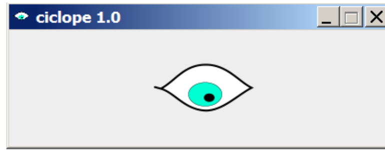


Figura 5.2.20. Pantalla inicial de bienvenida.

Para cambiar del estado de INICIAL al estado de INICIADO pulsamos el ojo. En el figura 5.2.21, podemos ver los cambios de botones que se producen al pulsar el ojo.

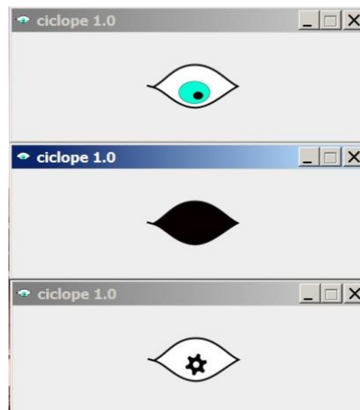


Figura 5.2.21. Cambio de estado de INICIAL a INCIADO.

Par iniciar el cambio hacia el estado de CONFIGURADO, tendremos que pulsar el ojo con el icono de configuración. Si todos los datos de configuración los obtuviéramos a través de los ficheros XML, la pulsación del botón generaría el cambio de estado. Pero el usuario además de haber editado los ficheros XML, tendrá que seleccionar el driver de la tarjeta de sonido. Para seleccionar el *driver* usaremos un combo box o cuadro combinando y pulsando el botón de confirmación realizaremos primero el cambio al estado de CONFIGURADO y luego el cambio a PREPARADO (ver figura 5.2.22).

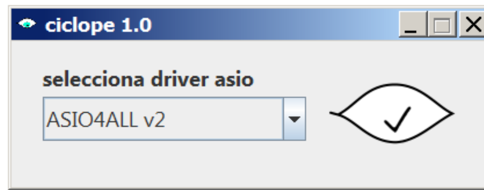


Figura 5.2.22. Cambio de estado de INICIADO a CONFIGURADO.

Una vez está preparada toda la aplicación para realizar la visualización tendremos que accionar el botón de reproducción, con lo que la aplicación pasará al estado de EJECUCIÓN. Mediante los botones de pausado y parada iremos pasando del estado de EJECUCIÓN al de PAUSADO y al de PREPARADO (ver figura 5.2.23). Por último como habíamos comentado al principio pulsando el botón de cierre del marco principal volveremos al estado INICIAL.

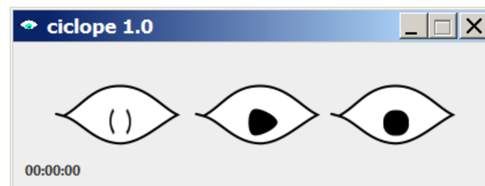


Figura 5.2.23. Pantalla para controlar la visualización

Como conclusión de esta primera parte del diseño de la interfaz gráfica, que se corresponde con la entrada de datos, podemos decir que se ha buscado la máxima simplicidad a nivel gráfico y también a nivel conceptual. El presionar cada uno de los botones supone realizar un cambio de estado en la aplicación. Por lo que la programación de los botones únicamente supondrá acciones de esconder o mostrar botones e indicar al `ControlProceso` el cambio de estado producido.

5.2.2.5 Interfaz gráfica de usuario para visualizar el resultado

Para terminar de describir el módulo de interfaz con el usuario solo nos falta comentar las decisiones realizadas en torno a la salida de resultados, esto es el lienzo de visualización. Para este elemento de la interfaz hemos buscado la máxima simplicidad. Una vez establecido su tamaño y posición con respecto al dispositivo de visualización, aparece como un lienzo limpio sin bordes (ver figura 5.2.24).

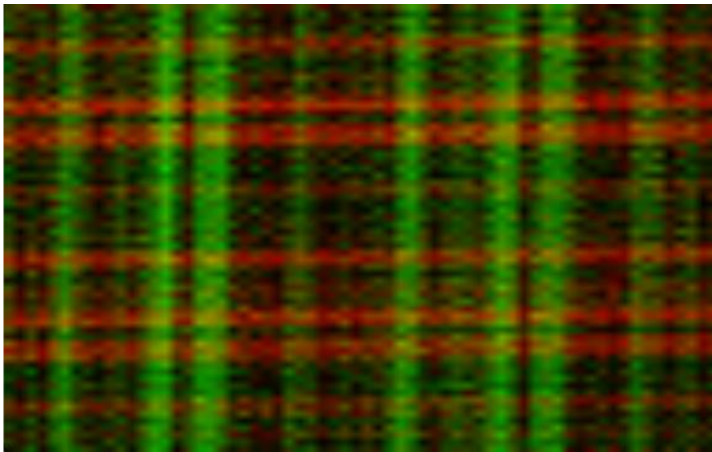


Figura 5.2.24. Lienzo de visualización.

Como conclusión de este apartado, la aplicación debe manejar el concepto de `Lienzo`, donde tendremos la información referente a las características del lienzo, como es su tamaño y su posición con respecto al dispositivo de visualización. Además tendremos que tener una componente de interfaz gráfica donde dibujar el lienzo, con que manejaremos el concepto `IULienzo`. Por último nos gustaría que la visualización estuviera en una compontente diferente al marco principal. Esto nos conduce a tener que tener el marco del lienzo, `IUMarcoLienzo`.

5.2.3 Módulo de sonido

El diseño del módulo de sonido ha sido una de las partes que más tiempo nos ha llevado, puesto que es la que se relaciona directamente con la tarjeta de sonido.

5.2.3.1 Comunicación entre la aplicación y la tarjeta de sonido

Las tarjetas de sonido como dispositivos físicos se comunican con las aplicaciones de usuario a través de los controladores de dispositivos o *drivers*, que vienen a ser partes de software intermediarias entre los dispositivos físicos, como la tarjeta de sonido, el sistema operativo y la aplicación. Aunque desde el punto del programador el sistema operativo sea transparente todo lo que concierne al hardware del sistema pasa por él.

El lenguaje de programación Java está dotado de una serie de librerías pensadas para el tratamiento del sonido. Si queremos capturar las muestras de audio de la tarjeta de sonido, la librería `javax.sound.sampled`, nos proporciona las clases adecuadas. En principio para las necesidades de la aplicación sería suficiente ya que no necesitamos realizar ninguna manipulación del sonido, simplemente lo tenemos que recuperar y almacenar en *buffers* (vectores). La idea inicial de usar la librería estándar de sonido de Java nos parecía la mejor opción. Además de proporcionarnos independencia del sistema operativo, dado que Java es un lenguaje multiplataforma. El problema se plantea cuando se quiere trabajar con más de dos entradas de audio. En esa situación las librerías estándar de lenguajes de programación se desentienden de dicha situación y toda la captura del sonido se delega a protocolos especiales dependientes del sistema operativo. Esta situación provoca que se deba realizar una programación específica para cada sistema operativo.

Inicialmente estuvimos revisando los protocolos de sonido del sistema operativo Linux[109]. Realmente la arquitectura del sonido en Linux es bastante compleja, esto nos supuso invertir mucho tiempo en estudiar los diferentes protocolos y pese a que al final vislumbramos una solución a través del protocolo Jack[110], descartamos este protocolo para realizar las pruebas.

En el sistema operativo Mac OS-X el protocolo de comunicación para múltiples entradas es CoreAudio. Para poder acceder a la tarjeta de sonido a través de Java con el protocolo CoreAudio podemos usar la librería `FrogDisco`[111], realmente es un *wrapper*[112], una especie de intermediario entre Java y el *driver* de la tarjeta de sonido.

En el sistema operativo Windows es donde obtuvimos el mecanismo más sencillo para acceder a las diferentes entradas de la tarjeta de sonido. Dentro de este sistema los protocolos que admiten múltiples entradas son el DirectSound que es una componente de la librería DirectX, especializado en sonido para videojuegos, y ASIO[113] que se trata de un protocolo, bastante sencillo, que surge con la finalidad de poder acceder a diferentes entradas y salidas de la tarjeta de sonido con muy baja latencia. Además mediante el *wrapper* `JAsioHos`t[114] resulta muy sencillo la programación de la captura de sonido desde Java.

[109] En http://www.tuxradar.com/content/how-it-works-linux-audio-explained, hay un artículo que explica con todo nivel de detalle estas cosas.

[110] Jack es el acrónico de Jack Audio Connection Kit. Se trata de un servidor de sonido para todas las aplicaciones que usan sonido en Linux.

[111] La librería FrogDisco es open source y se puede descargar de https://github.com/mhroth/FrogDisco (visitado 12/10/2015)

[112] La traducción al castellano sería empaquetador o adaptador, hemos querido mantener el término original como es habitual en el campo de la informática.

[113] ASIO son las siglas de *Audio Stream Input Output* es un protocolo de comunicación para audio de la empresa Steinberg.

[114] https://github.com/mhroth/jasiohost

Finalmente nos decantamos por el sistema operativo Windows utilizando el protocolo de sonido ASIO. La mayoría de las tarjetas de sonido de un nivel alto o medio proporcionan el *driver* ASIO correspondiente. Además aunque no dispongamos de una tarjeta de sonido con protocolo ASIO, siempre se puede ejecutar la aplicación usando el driver virtual ASIO4ALL[115]. El driver ASIO4ALL establece una transformación entre este protocolo y el protocolo WDM[116].

Si queremos que nuestra aplicación funcione con los tres sistemas operativos mayoritarios, Windows, Linux y OS X, tendremos que realizar una programación específica para cada uno de ellos. De manera que cuando arranquemos la aplicación se detecte el sistema subyacente y se active la parte de código adecuada. Como el objetivo de nuestra aplicación no tiene fines comerciales ni de uso generalista, hemos seleccionado un protocolo asociado a un sistema operativo y hemos trabajado sobre él. Al final la decisión ha sido pragmática, la comunicación con la tarjeta de sonido usando el protocolo ASIO es rápida, la mayoría de tarjetas de sonido del mercado tienen el *driver* ASIO y la conexión de la aplicación con la tarjeta a través del protocolo también es sencilla, mediante la librería `JAsioHost`.

5.2.3.2 Capturar el sonido

La tarea fundamental del módulo de sonido es ir a la tarjeta de sonido, recuperar las muestras de sonido a intervalos regulares de tiempo y almacenarlas. Para acceder a la tarjeta de sonido como hemos comentado hemos usado la librería `JAsioHost`. Esta librería está basada en el modelo de eventos que Java usa para manejar las

[115] ASIO4ALL es un proyecto de Michael Tippach que se inicia en 2003 con la idea de conseguir soporte del protocolo ASIO para la tarjeta de sonido de un ordenador portátil.

[116] WDM son las siglas de Windows Driver Model, se trata de un modelo para desarrollar driver en el sistema operativo Windows, como por ejemplo drivers para tarjetas de sonido, aunque no es específico de sonido.

componentes gráficas. La idea es muy sencilla, el driver pacta con la aplicación una zona de código que se ejecutará cuando tenga disponible un conjunto de muestras. Desde el punto de vista del programador tendremos que programar el código de un método pactado. Y desde el punto de vista del driver (sistema) se realizará una invocación a dicho método. La frecuencia con la que llegan las muestras dependerá de la frecuencia de muestreo y el tamaño del *buffer,* que hayamos configurado en la tarjeta de sonido.

De forma más esquemática, el proceso de captura sigue los siguientes pasos:

1. Se llena el *buffer* de la tarjeta de sonido
2. El driver coge el buffer y lo envía a la aplicación como parámetro de un método que han pactado.
3. La aplicación al ejecutar dicho método puede acceder al buffer y guardar las muestras. Volvemos al paso 1.

Este esquema se repite para cada una de las entradas de audio que tengamos conectadas, hasta que se detiene el servidor del driver.

De estas consideraciones concluimos que necesitamos una clase `LineaEntrada`, que tendrá como atributos la lista de los buffers que vamos recuperando y el método principal será el método pactado con el driver para realizar el proceso de captura. Como lo debe conocer el driver o la librería que comunica con el driver el nombre del método viene fijado, y en este caso es `bufferSwich`. Dentro del lenguaje Java, los asuntos relacionados con los pactos entre el sistema y la aplicación se resuelven mediante el concepto de interface. Para conocer más detalles sobre el funcionamiento de Java se puede consultar (Eckel, 2002).

Como estrategia de diseño hemos realizado la captura de sonido de forma independiente para cada una de las líneas. Realmente en un principio pesamos en realizar una captura global de todas las líneas, es decir, un método que capturara todos los valores de las líneas. Pero en las pruebas preliminares esta opción no funcionaba y se perdían las muestras de muchas de las líneas así que lo desechamos.

Debido a que el concepto de *buffer* es algo más que un almacén de datos, y puesto que tendremos que etiquetar los buffers con marcas temporales para saber cuándo se generaron, también introducimos la clase `BufferLinea`. Realmente es un almacén de datos con el añadido de las etiquetas temporales y la posibilidad de almacenar los números en diferentes formatos.

5.2.3.3 Configuración del sonido

Completan el módulo las clases relacionadas con la configuración del sonido. El usuario introduce los datos de sonido a través del fichero XML configuracionSonido, y la aplicación recoge estos datos y con ellos construye la clase `ConfiguracionSonido`. Para ello hemos definido una clase `ConfiguracionSonidoXML`, que se encarga de leer el fichero XML y construir a partir de esos datos un objeto de la clase `ConfiguracionSonido`, donde reside toda la información necesaria para manipular el sonido.

Desde el lenguaje de programación Java, existen diferentes librerías para acceder a un fichero XML y recuperar los datos de acuerdo a sus etiquetas. Para nuestra aplicación hemos seleccionado la librería `jdom2`[117], para realizar la lectura y escritura de las etiquetas del

[117] `Jdom2` es una librería *open source* (código abierto) de Java para manipular, acceder y guardar ficheros en formato XML, desarrollada por Jason Hunter y Rolf Lear.

lenguaje XML. En la librería el fichero XML es un objeto de la clase `Document`, y a partir de ahí se manejan las etiquetas que se corresponden con el concepto de `Element`, En ambas clases hay métodos para ir accediendo a los datos de los elementos, sus atributos y los elementos anidados. De esta forma resulta muy sencillo acceder a los datos de las diferentes etiquetas.

Finalmente como en los otros módulos tendremos que dejar una referencia de la configuración en la clase `ConfiguracionControl` del módulo de control.

5.2.4 Módulo de Imagen

El módulo de la imagen desde el punto de vista del diseño es el más complejo de todos. El objetivo final es realizar la visualización de las muestras ya ubicadas en las imágenes, pero en su diseño debemos plantear en sí las estrategias de ubicación espacial de las muestras y el control de la secuenciación, es decir de las variaciones temporales de la presentación de las imágenes.

Para presentar las decisiones realizadas en este módulo vamos a dividirlo en 4 partes:

- Configuración del lienzo y las imágenes
- Secuenciación temporal de las variaciones de visualización
- Generación de las imágenes en función de los parámetros actuales de visualización
- Presentar en el lienzo las imágenes.

5.2.4.1 Configuración de la imagen

Para realizar la configuración de la imagen lo que haremos es leer el fichero XML de configuración y obtener los datos referentes al lienzo y a las imágenes. Como en el caso del sonido hemos seguido el mismo

esquema. Hemos dividido la configuración en dos clases una para almacenar el acceso a las clases principales del módulo de la imagen y otra para realizar la lectura y escritura del fichero XML. La librería para acceder al fichero XML es la misma que en el caso del sonido, `JDOM2`.

Como hemos visto en el apartado de Interfaz de usuario, de la lectura del fichero XML obtendremos los datos para crear un objeto de la clase `Lienzo`. Esta clase representa una superficie de dibujo donde colocaremos las imágenes. Sus atributos principales son las dimensiones del lienzo y su posición respecto al dispositivo de visualización. Esta clase tiene su equivalente gráfica en el módulo de la interfaz gráfica `IULienzo`.

El otro concepto importante del módulo de imagen es la propia imagen. Esta es sin duda la clase principal de todo el módulo, incluso la clase lienzo podríamos incluirlo en este concepto, ya que no deja de ser una imagen también conteniendo a otras imágenes. Además de los atributos que obtenemos del fichero XML: dimensiones, posición respecto al lienzo, rotación, centro de rotación y escalado. Tendremos una serie de atributos adicionales relativos a la lista de variaciones o lista de secuenciación que veremos a continuación, y una serie de atributos heredados de la clase `BufferedImage` de la librería `java.awt.image` que es la que representa en si una imagen dentro del lenguaje Java. Con esto lo que realmente hemos definido como imagen es una serie de características estáticas y otra serie de características dinámicas, que irán variando a lo largo del tiempo. La variación se realizará en función de los nodos temporales que estén activos en cada momento.

La clase `BuffedImage` está formada por varios elementos y el que nos interesará para generar la imagen es el `Raster`, que es el objeto

que usamos para acceder y modificar directamente a los pixeles de la imagen. Con esta clase accederemos a cada elemento de la imagen a través de su posición, cómo esta almacenada internamente no nos preocupa para construirla o leerla.

5.2.4.2 Secuenciación

Como hemos visto en el módulo de la interfaz de usuario, el diseño del lenguaje de secuenciación ha sido una de las partes más conflictivas de la aplicación. En cualquier caso una vez resuelto, aunque sea de forma textual lo que resta es traducir la secuenciación del fichero XML en un conjunto de clases que proporcionen una estructura adecuada. Lo importante es que en tiempo real podamos manejar las variaciones de parámetros.

Como ya veíamos en la figura 5.2.7, tendremos para cada una de las imágenes una lista de líneas. Cada línea llevará asociado el *buffer* a visualizar y una lista de nodos que en su momento llamamos temporales donde cada uno indicaba: inicio, duración, algoritmo, canales a los que afecta y método de fusión. Puesto que el concepto de línea es un poco diferente al concepto de línea de sonido de secuenciación, hemos decidido crear una nueva clase `LineaSonidoGenerar`, que contiene a `LineaEntrada`, pero además incluye la lista de nodos temporales. Cuando comienza la secuenciación tendremos que manejar dicha lista indicando en cada momento cuál es el elemento distinguido de la lista.

De manera que lo que tenemos es una imagen, con sus datos generales y luego para cada imagen una lista de líneas cuyas muestras deben ser ubicadas en la imagen y para cada línea una lista de decisiones a aplicar e información del nodo temporal que actualmente está activo con los datos concretos de visualización.

Un aspecto importante en el diseño es resolver la secuenciación cuando la aplicación se encuentra en estado de ejecución. La idea es diseñar una tarea que cada segundo compruebe de cada una de las listas de las líneas de cada imagen, si el nodo que está marcado como actual sigue siéndolo o tenemos que cambiarlo. Para realizar este proceso, tendremos que ir consultando el reloj de la aplicación, que como ya hemos visto está disponible en la clase `ConfiguracionControl`.

Este proceso de secuenciación se debería traducir, de forma natural, en un método de la clase `Imagen`, pero puesto que debe ser un hilo independiente se ha introducido como una clase a parte. No obstante queremos insistir en esta situación especial que se volverá a producir en el proceso de generación de la imagen.

Del proceso de secuenciación concluimos que tendremos la clase `Imagen`, con la lista de las `LineaSonidoGenerar`, cada línea tendrá a su vez la lista de nodos temporales, `NodoTemporal`, que tendrá la información del inicio, duración,… Además asociado a cada lista tendremos un objeto `Secuenciar`, que consiste en un hilo de ejecución independiente del resto, cuya misión es en cada segundo comprobar si la hay que avanzar la lista de nodos temporales.

5.2.4.3 Generación de las imágenes en función de los parámetros actuales de visualización

El proceso principal del módulo de imagen es justamente generar las imágenes que luego se pintarán en el lienzo. Una vez todo coordinado y preparado será relativamente fácil realizar este proceso.

Como ya hemos visto el proceso secuenciar activa un nodo temporal para cada una de las líneas. A partir de la información que nos proporciona, tendremos otro proceso `Generar`, que es un hilo

independiente de ejecución, que lo que hace es cada cierto tiempo recoger las muestras de los buffers de las líneas implicadas y los ubica espacialmente de acuerdo a la información de los nodos temporales

Desde el punto del diseño el proceso de generar los pixeles de la imagen debería ser un método de la clase Imagen, pero como debe comportarse como un hilo de ejecución independiente lo hemos sacado de la clase `Imagen` y hemos creado una clase aparte, donde estarán todos los algoritmos de ubicación.

Para realizar las pruebas del método hemos diseñado los algoritmos que se han propuesto en el apartado 4, a grandes rasgos los podemos dividir en:

- Algoritmos simétricos.
- Algoritmos aleatorios.
- Algoritmos combinados: simétricos.
- Algoritmos combinados: simétrico y aleatorios.
- Algoritmos basados en reglas de comportamiento.

En el diseño de la clase `Generar` se ha buscado independizar el código de cada uno de los algoritmos.

Como conclusión para generar las imágenes en función de las muestras de sonido, tendremos asociado a cada imagen un objeto de la clase `Generar`, que es un hilo de ejecución independiente que recorre la lista de las líneas y en función de los nodos temporales distinguidos de las líneas ubica sus muestras. Tendremos tantos métodos como algoritmos de ubicación diferentes.

5.2.4.4 Pintar en el lienzo las imágenes.

El proceso de pintar en el lienzo las imágenes lo hacemos en la clase `IULienzo` del módulo de la interfaz gráfica. Todas las componentes gráficas en Java tienen el método `paintComponent`, que Java invoca cuando el componente se debe dibujar. Por lo tanto para dibujar las imágenes que hemos generado tendremos que recorrer las lista de imágenes, que está en la clase `ConfiguracionImagen` y presentarlas de forma visual de acuerdo a sus propiedades. Esta operación está plasmada en el código del método `paintComponent`.

Para terminar con el módulo de imagen podemos decir que la clase principal es la clase `Imagen` y luego sobre ella tendremos sus dos extensiones que son `Generar` y `Secuenciar`. Estas dos acciones son realmente métodos de la clase Imagen pero al tener que ser hilos independientes hemos decidido diseñarlos de forma separada.

5.2.5 Gestión de errores

Para la gestión de errores hemos seguido el modelo habitual del lenguaje Java, que usa el concepto de `Exception`, Para cada uno de los módulos en los que se divide la aplicación hemos definido una clase especializada en las excepciones de dicho módulo que extiende el concepto general de Java de `Exception`. Para más información sobre el modelo de excepciones de Java se puede consultar (Eckel, 2002,405-436).

De esta forma hemos definido las clases `SonidoException`, `ImagenException`, `ControlException` y `InterfazUsuarioException`, para controlar los errores en cada uno de los módulos.

Cuando en el desarrollo de la aplicación, tenemos objetivos comerciales el control de los errores es una tarea fundamental. En nuestra aplicación se han controlado la mayoría de ellos pero no todos.

5.2.6 Otras consideraciones de diseño

En el análisis de la aplicación hemos visto que sería interesante introducir nuevos algoritmos de ubicación de forma sencilla. Para poder llevar a cabo este requisito, de la manera más flexible posible, se debería diseñar un mecanismo para la modificación de código en tiempo real.

Cada vez es más habitual incluso en aplicaciones comerciales, que la aplicación deje una puerta abierta para que el código pueda ser modificado por el usuario. Históricamente en el ámbito de la imagen y el sonido, este mecanismo se ha desarrollado a través del concepto de complemento[118]. Un complemento o *plug-in* es una pequeña aplicación pensada para ampliar las capacidades de otra aplicación. Normalmente lo que se establece es un protocolo de comunicación pactado entre la aplicación madre y el *plug-in*.

Para satisfacer el requisito de poder introducir nuevos algoritmos lo único que hemos tenido en cuenta es el hecho de localizar e independizar lo máximo posible en el código la parte del algoritmo de ubicación de muestra, como hemos visto en el módulo de imagen, pero no vamos a prever un mecanismo externo para tal fin, como podría ser el modelo de *plugin*.

[118] El término complemento es más conocido por su nombre en inglés, *plug-in 5366QT3378*.

5.3 Implementación de la aplicación

Tras el proceso de diseño de la solución lo que resta es implementar todas las clases que hemos identificado para cada uno de los módulos. Como ya hemos comentado el lenguaje de programación con el que vamos a implementar el programa es Java.

Para poder trabajar en un entorno agradable donde poder escribir el código, consultar bibliotecas, realizar compilaciones y pruebas, y depurar el programar se usan aplicaciones orientadas al desarrollo integrado de la aplicación, normalmente conocidas como IDE[119].

Para la parte de implementación hemos optado por el IDE generalista Eclipse[120], se trata de un software libre que tiene módulos para programar en diferentes lenguajes y entre otros para la familia de lenguajes de Java. Se trata de un entorno de programación ampliamente utilizado con licencia de software libre. En la figura 5.3.1 podemos ver la interfaz gráfica principal de Eclipse.

Figura 5.3.1. Interfaz gráfica principal de Eclipse.

[119] IDE es el acrónimo de *Integrated Development Environment*,.

[120] Eclipse fue inicialmente desarrollado por la empresa estadunidense IBM, actualmente es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta la comunidad de código abierto.

Una de las acciones que hemos tratado de realizar desde la escritura de las primeras líneas de código ha sido documentar adecuadamente todas las clases y métodos que aparecen. El lenguaje de programación Java tiene una herramienta Javadoc, accesible desde todos los IDE, para generar de forma automática la documentación del programa. Esa herramienta genera la documentación en formato HTML[121], el estilo que resulta es similar a la documentación de las librerías estándares del lenguaje con las que todo [122]programador de Java está familiarizado. Como vemos en la figura 5.3.2 la página generada está formada por 3 marcos principales, en el de la esquina superior izquierda tenemos enumerados el conjunto de paquetes de la aplicación, en la esquina inferior izquierda tendremos el desglose de clases, interfaces y excepciones del paquete seleccionado y en la parte central tendremos la información del elemento seleccionado, en el caso de la figura hace referencia a la estructura general de la aplicación.

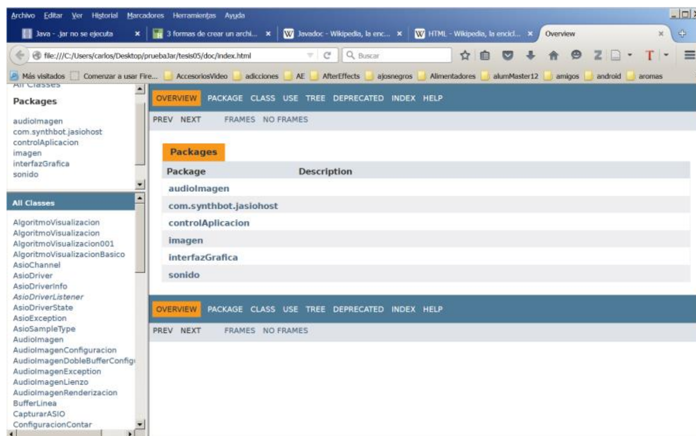


Figura 5.3.2 Página principal de la documentación de Ciclope.

[121] HTML, son las siglas de *Hyper Text Markup Languaje*, se trata de un lenguaje de marcas de hipertexto orientado a la elaboración de páginas web.
[122]

Toda la documentación de la aplicación la hemos añadido en el anexo 1. Realmente lo que hemos realizado es una conversión de los ficheros HTML de la documentación en ficheros PDF. Con lo que aparece en el anexo 1, es una concatenación de la documentación HTML.

En la propia documentación hemos incluido algunos de los detalles más interesantes de la implementación, pero principalmente lo que presenta es la estructura interna del programa.

En lo que resta de capítulo vamos a comentar algunas de las ideas que hemos plasmado en la implementación de la aplicación referentes a la sincronización de procesos y también algunas consideraciones finales sobre la eficiencia de la aplicación.

5.3.1 Acceso en exclusión mutua

En este apartado de implementación nos gustaría resaltar que hemos implementado una estrategia basadas en el problema de lectores y escritores para el acceso a las imágenes.

Cuando varios procesos quieren acceder a un mismo recurso, entiéndase por ejemplo a una imagen, el problema de los lectores y escritores distingue entre dos tipos de actuaciones, por un lado se accede al recurso para su lectura o consulta y el que accede para su escritura o modificación. Se trata de una modelización básica de acceso a una base de datos.

En nuestra aplicación hemos implementado un mecanismo de acceso a las imágenes usando la estrategia de lectores y escritores. La idea es que el proceso de generación de la imagen, que es el escritor, cuando este modificando los datos de la imagen impida que el lector, el proceso de render, acceda a la imagen y tenga que esperar. De

forma que no se produzcan situaciones en las que mientras se está presentando la imagen simultáneamente se esté actualizando.

5.3.2 Eficiencia de la aplicación

En el diseño de la aplicación hemos tratado de cuidar los detalles relacionados con la eficiencia computacional de la solución elaborada. Dentro del campo de la programación el tratamiento de la imagen siempre ha sido y es un ámbito que requiere mucho consumo de ciclos de CPU[123], de manera que desarrollando programas eficientes y utilizando hardware adecuado en la mayoría de situaciones podemos abordar sin generar grandes retrasos su correcta ejecución.

Cíclope trabaja principalmente con dos tipos de estructuras de datos, por un lado los vectores de muestras de sonido, que es lo que hemos denominado buffers, con una capacidad entre 1024 y 4096 elementos, y las matrices que construimos con dichos vectores que contienen normalmente el mismo número de elementos. El ciclo de vida básico de procesamiento es:

1. De la tarjeta de sonido llegan muestras, estas tendremos que almacenarlas en un vector, el $buffer$, uno por cada una de las líneas de sonido que tengamos activas.
2. Una vez los $buffers$ están cargados, recorremos de nuevo todos sus elementos y generamos una o varias imágenes, en función del algoritmo utilizado. Este proceso dependiendo de la complejidad del algoritmo utilizado y sobre todo del número de imágenes involucradas ya puede ser crítico.

[123] La CPU son las siglas de Central Processing Unit, unidad central de procesamiento, es el elemento de hardware dentro de un sistema informático que interpreta las instrucciones de los programas.

3. Por último esas imágenes se escalan, posicionan, … para adecuarlos al dispositivo de visualización. Este proceso también es crítico, puesto que hay que volver a visitar todos los elementos de la imagen.
4. Las imágenes se visualizan en el dispositivo final de visualización.
5. Se repite el ciclo.

En el diseño de la aplicación el ciclo no es exactamente secuencial, en el sentido que como ya hemos comentado tendremos varios procesos ejecutándose de forma paralela. Uno que se encarga de cargar los $buffers$, otro que se encarga de generar las imágenes y otro que se encarga de visualizar las imágenes. Existe un cuarto proceso menor que es el reloj, aunque computacionalmente no tiene consecuencias.

Para el paso 1), lo que hemos utilizado es una librería externa, $com.synthbot.jasiohost$, en la documentación de la misma no especifica nada sobre su eficiencia, pero asumimos que es lineal con respecto al tamaño del vector.

Para el paso 2), lo que hacemos es recorrer todos los buffers y cada muestra la ubicamos en las imágenes que le corresponda. En este caso dependerá del número de imágenes y del coste de encontrar su posición. De momento y para todos los algoritmos de ubicación este es constante y no depende del número de elementos del $buffer$. Por lo que este proceso también es lineal en función del número de elementos que tenga el $buffer$.

En la solución que hemos planteado, el paso 3) y el 4) son realizados por el mismo proceso y no se ha aplicado una estrategia de doble *buffering.* Si tenemos previsto aplicarla pero de momento no. Estos

dos pasos son los que realmente generan el cuello de botella de la aplicación, puesto que principalmente el escalado y el renderizado se realizan de forma simultánea. El mecanismo de doble *buffering* lo que hace es independizar esos dos procesos, de forma que cuando vayamos a presentar físicamente las imágenes estas ya estén procesadas. En cualquiera de los casos el coste computacional es lineal al número de elementos de las imágenes a renderizar, pero como la presentación final física es una operación de entrada/salida esta es la que más cuesta y tenemos que cuidar.

Como conclusión podemos decir que la complejidad computacional de la aplicación es razonable, puesto que es lineal al número de elementos a visualizar. No obstante hay que tener en cuenta que la carencia del ciclo de vida es muy rápida, y renderizamos entre 20 y 30 veces por segundo. Una mejora considerable sería el diseñar una estrategia de doble *buffering*.

6 CASOS PRACTICOS. ANÁLISIS DE ALGORITMOS DE UBICACIÓN DE MUESTRAS.

En este apartado vamos a mostrar algunos de los resultados obtenidos de las pruebas realizadas con la aplicación Ciclope para una serie de casos prácticos. Los casos prácticos que vamos a presentar tratarán de mostrar las relaciones que se producen entre el sonido y la imagen. El objetivo no es realizar piezas audiovisuales sino principalmente probar la aplicación para poder establecer las propiedades estéticas y técnicas conseguidas con el método de visualización.

Desde el punto de vista sonoro vamos a estudiar los resultados visuales de una serie de sonidos bien definidos. Algunas de las propiedades que hemos considerado son: complejidad de armónicos, intensidad sonora, frecuencia sonora, número de líneas sonoras y consonancia o disonancia de las diferentes líneas. Con el objetivo de mostrar esta gama de propiedades hemos usado para las pruebas sonidos directamente generados por sintetizadores físicos y mediante software, sonidos de guitarra eléctrica y sonidos de bajo acústico. En la mayoría de las pruebas se ha trabajado con esquemas de improvisación, en las que en una misma sesión se han considerado muchas propiedades sonoras.

Con respecto a la generación de la imagen hemos tratado de estudiar desde visualizaciones muy sencillas con una única imagen y estrategia de ubicación espacial hasta visualizaciones más complejas con varias imágenes distribuidas a lo largo del dispositivo de visualización. Para cada una de las líneas de sonido, tendremos en cuenta aspectos como: separación de líneas en canales y algoritmos

utilizados, separación de líneas en diferentes imágenes, complejidad de los algoritmos y evolución temporal de estos parámetros.

Para cada uno de los casos primero realizaremos una descripción general de los objetivos de la prueba, luego realizaremos una descripción más detallada de las características técnicas del sonido y de la imagen. Por último realizaremos una valoración de los resultados obtenidos, en base a la estética y características visuales y la correlación con las propiedades sonoras. No obstante los resultados visuales están publicados en formato de video en la red social Vimeo[124], en cada uno de los casos especificamos el nombre del fichero de video.

Antes de describir cada uno de los casos vamos a realizar una descripción más detalla de los elementos sonoros y visuales que hemos tenido en cuenta para la realización de los pruebas.

6.1 Descripción técnica del diseño de sonido

El objetivo que perseguimos es disponer de una gran cantidad de variaciones sonoras y para eso vamos a definir una clasificación de tipos de sonido en función de diferentes propiedades y luego en cada una de las pruebas especificaremos los tipos de sonidos utilizados.

Generalmente es complicado realizar una clasificación de sonidos debido a que son varios los factores que intervienen en su estructura. Lo primero que queremos dejar claro es que el sonido no es algo puntual, sino es un medio que fluye en el tiempo y para poder interpretarlo o clasificarlo tendremos que tener en cuenta su devenir temporal. De manera que cuando hablamos de sonido no siempre nos referimos a él a través de su percepción en un breve instante de

[124] El canal de Vimeo donde están publicados todos los resultados de la tesis es https://vimeo.com/user18072163 …

tiempo, como sería una nota musical del piano, sino a través de un periodo más largo sobre el que devienen muchas notas.

Empezaremos describiendo los tipos de sonidos en función de su complejidad armónica:

1. Sonidos estables puros. La idea es probar con sonidos sinusoidales sin armónicos o que tengan muy pocos armónicos.
2. Sonidos afinados de complejidad media. En este caso estamos haciendo referencia a sonidos que mantienen ya un gran número de armónicos, con una relación matemática entre ellos, como podrían ser los sonidos de un piano o de una guitarra.
3. Sonidos inarmónicos. En este caso se trata de sonidos casi afinados pero algunos de sus armónicos no mantienen una relación matemática con la frecuencia fundamental, como los sonidos de una guitarra distorsionada.
4. Sonidos no afinados de complejidad media. Este tipo de sonidos son sonidos complejos cercanos al ruido pero con muchos menos armónicos que los ruidos, son los sonidos de muchos de los componentes de una batería acústica.
5. Sonidos complejos, ruidos. En este caso probaremos diferentes tipos de ruido desde ruido blanco hasta ruidos más filtrados armónicamente, como el ruido rosa.

A continuación presentamos los tipos de sonidos en función de su relación con los sonidos anteriores, simultáneos y posteriores:

1. Sonidos Monofónicos o de una sola voz.

2. Sonidos Armónicos. En este caso los sonidos son polifónicos[125], tienen acordes[126] y el carácter estético es de equilibrio entre las diferentes partes que intervienen. Un sonido es armónico porque la construcción de los acordes sigue unas proporciones determinadas y su progresión temporal también obedece a una serie de principios.
3. Sonidos polifónicos no armónicos. Se trata de más de una voz sonando simultáneamente pero sin equilibrio aparente.
4. Sonidos disonantes. En este caso lo que tenemos es una secuencia de sonidos que se perciben con tensión. Muchas veces es difícil establecer la frontera entre disonancia y consonancia.
5. Sonidos consonantes, son aquellos sonidos que el oído acepta como equilibrados.
6. Sonidos melódicos. Se trata de una secuencia de sonidos que siguen unas reglas equilibradas y de alguna forma establecen la identidad de una pieza sonora en particular.
7. Sonidos rítmicos. Un sonido es rítmico cuando un fragmento de sonido de una duración superior a los 50 milisegundos se repite con muy pocas variaciones de forma periódica.
8. Sonidos arrítmicos o irregulares. Es cuando el sonido no tiene regularidad. Es el caso contrario al anterior.

Con respecto a la fuente sonora que emite el sonido podemos distinguir entre:

---

[125] Los sonidos polifónicos son aquellos que tienen más de una voz, un caso particular es el acorde.

[126] Un acorde en música es un conjunto de más de dos notas musicales que suenan simultáneamente. Las notas musicales son sonidos afinados que emiten los instrumentos musicales tradicionales.

1. Sonidos naturales. Son sonidos que se generan en el medio natural. Posteriormente mediante un transductor los tendremos que transformar en energía eléctrica.
2. Sonidos de instrumentos acústicos. Son sonidos naturales pero generados con instrumentos musicales con la finalidad de que estén afinados.
3. Sonidos de instrumentos eléctricos. Son sonidos que se generan en el medio natural y la onda sonora se transforma a través de un transductor en energía eléctrica.
4. Sonidos Sintéticos. Son aquellos generados de forma sintética, no existen en la naturaleza como tales. Se pueden generar de forma electrónica o digital.

Por último realizamos una clasificación en función del número de instrumentos involucrados en el sonido final. Por instrumentos entendemos no necesariamente un instrumento físico sino las fuentes sonoras conectadas a las entradas de audio de la tarjeta de sonido. Para esta clasificación tendremos:

1. Sonidos con una sola fuente de sonido.
2. Sonidos con dos fuentes de sonido relacionadas mediante efectos de retardos. Este tipo de fuentes son muy habituales en los efectos de sonido, se trata de duplicar la señal y enviar una de las copias a través de otra línea de sonora ligeramente retrasada y alterada tímbricamente.
3. Sonidos con varias fuentes de sonido.

A través de esta pequeña clasificación vamos a realizar una serie de composiciones musicales, basadas en combinaciones de diferentes tipos de sonido. Lo ideal sería realizar una composición que contuviera variaciones de todos ellos. No obstante en función del tipo

de parámetros de visualización empleados muchas de las variaciones no serán relevantes. Por ello para la presentación de conclusiones nos centraremos en las partes que entendemos son más significativas.

6.2 Descripción técnica de los parámetros de visualización

Para el diseño de la visualización vamos a considerar los siguientes parámetros:

1. Número de imágenes y sus transformadas básicas respecto del lienzo.
2. Número de líneas involucradas en cada imagen, donde para
2. cada una de las líneas tendremos:
    - el número de buffer utilizado,
    - el algoritmo,
    - su correspondencia con los canales de color y

En todos los casos las imágenes al final se escalarán para ubicarlas en la parte que nos interese del dispositivo final de visualización. Cuando realizamos el escalado de una imagen se aplican estrategias de interpolación, las más habituales son: la pura que genera un borde pixelado o duro, en este caso la idea es no realizar interpolación y para escalar o duplicamos muestras o las eliminamos, y la bilineal que genera bordes mucho más suaves, los nuevos pixeles se calculan en base a obtener promedios de sus vecinos. Aunque existen más modalidades de interpolación solo hemos usado estas dos.

6.3 Descripción del equipo de trabajo. El colectivo PDP11.

Para la realización de las pruebas hemos contado con el colectivo PDP11 del que formamos parte activa. A continuación presentamos una pequeña reseña sobre el colectivo.

El colectivo PDP11

PDP11, nació con la finalidad de establecer una plataforma abierta a la investigación del sonido y la imagen en su lado más cercano a la tecnología digital. Posteriormente el rango sensorial se amplió para abordar también arte culinario.

El nombre PDP11 es un homenaje a uno de los primeros miniordenadores de Digital, *Programmed Data Processor* (PDP versión 11), en cuyo nombre intencionadamente no se quería que apareciera la palabra *computer* por lo aparatoso del concepto en los años 70. En la serie 11 por primera vez se interconectan todos los elementos del sistema: procesador, memoria y periféricos en un espacio reducido. Esta idea de reunión de diferentes elementos es la que toma como referencia del colectivo.

Los intereses principales de investigación son:

- Arte sonoro. Experimentación con sonidos naturales, sonidos sintéticos e instrumentos tradicionales y no usuales.

- Arte visual. Experimentación con síntesis de imagen e imágenes reales.

- Arte culinario. Experimentación con nuevas formas de elaborar, mirar, oler y saborear la comida.

- Arte multisensorial como modelo de comunicación. Búsqueda de relaciones e interacciones entre el arte sonoro, artes visuales y arte culinario.

La actividad del colectivo se plantea en formato de proyectos abiertos. En cada uno de los proyectos se define un modelo conceptual sobre el que estructurar los contenidos. Cada uno de los participantes aporta sus ideas y en función de la disponibilidad personal y las características del espacio y audiencia se adecua el proyecto, dejando siempre un espacio abierto a la improvisación y tratando de buscar la integración de los principales intereses del colectivo: sonido, imagen, olores y sabores.

En el desarrollo de los casos de estudio, además del doctorando y el cotutor, ha participado el artista Deco Nascimento.

6.4 Descripción del material técnico usado para las pruebas

En este apartado vamos a describir el equipo físico que hemos usado para la realización de las pruebas, que detallamos a continuación:

- Ordenador Portátil Mountain Graphite 20 PRO, con procesador Intel (R) Core ™ i7-4710MQ CPU 2.5GHz y memoria RAM 16 GB
- Sistema Operativo Windows 7 64 bits
- Tarjetas de sonido:
    - Roland Oct-Capture, 8 líneas de entrada con preamplificación y 8 salidas. Soporta protocolos ASIO 2.0, WDM y Core Audio.
    - Behringer FCA610, 4 líneas de entrada dos de ellas con preamplicación y 8 salidas.
- Tarjetas de video:

    - Intel HD Graphics 4600.
    - Matrox TripleHead2Go.
- Monitores de Audio: Makie SRM 450 V3, con respuesta de frecuencia de 42Hz – 23kHz.
- Monitor de video Eizo FlexScan 22” 1680x1050.

Para algunas pruebas hemos utilizado esquemas de síntesis de sonido por software, sintetizadores e instrumentos musicales, en esos casos se comentarán las características específicas de ese material.

6.5 Esquema general de cada uno de los casos prácticos

Con la finalidad de establecer un modelo común para realizar la evaluación de los casos, vamos a establecer un esquema general que luego iremos adaptando a cada uno de los casos en particular.

Para cada uno de los casos comentaremos los siguientes ítems:

- **Descripción general**. En este ítem explicaremos los objetivos generales de la prueba.
- **Lugar**. Lugar donde se ha realizado la prueba.
- **Aspectos Técnicos.** En el caso de haber usado material adicional al general de las pruebas lo comentaremos en este ítem.
- **Sonido**. En este ítem realizaremos una descripción detallada de los sonidos y composiciones usadas para la sesión y el número de líneas de sonido que hemos usado.
- **Imagen**. En este ítem realizaremos una descripción detallada de las líneas involucradas en la generación de la imagen, los tipos de algoritmos usados y su secuenciación.
- **Grabación**. Aspectos a resaltar de la grabación del caso o la sesión

- **Conclusiones** finales.

6.6 Casos Prácticos

En este apartado vamos a presentar una serie de casos prácticos basados principalmente en el tipo de sonidos que vamos a emplear. Para cada uno de los casos trataremos de abarcar el máximo número de algoritmos de ubicación. En el caso 1 (C1) lo que planteamos es el estudio de los algoritmos ante sonidos muy periódicos, como es el de una sinusoidal y sonidos muy ruidosos como es el ruido blanco. En el caso 2 (C2) estudiaremos el comportamiento de los algoritmos con el sonido una guitarra procesada. En el caso 3 (C3) todas las pruebas están realizadas con sonidos de síntesis. En el caso 4 (C4) las pruebas se han centrado en una combinación de guitarra y sonidos de síntesis. En el caso 5 (C5) se presenta un ensayo improvisado de PDP11, con una visualización básica y dividida por instrumentos. Y finalmente en el caso caso 6 (C6), presentamos otro ensayo de PDP11, para el proyecto de Lux Interior.

6.6.1 Caso 1. Señal periódica

Descripción general. En objetivo principal de este caso es estudiar el comportamiento visual de sonidos muy periódicos, en su relación a la frecuencia.

Lugar. Se trata de una prueba de laboratorio, realizada en el estudio de PDP11.

Aspectos Técnicos. El material técnico usado es el general de todas las pruebas. Con respecto al sonido hemos diseñado un *patch*[127] en

[127] En el lenguaje de programación gráfico Pure Data, un *patch* es un programa. La terminología proviene del campo de la síntesis de sonido.

Pure Data (PD) que genera una sinusoidal pura y un armónico de ésta, integra también un generador de ruido (ver figura 6.6.1). También se ha utilizado una variación de este *patch* en el que se modula la intensidad del primer oscilador.

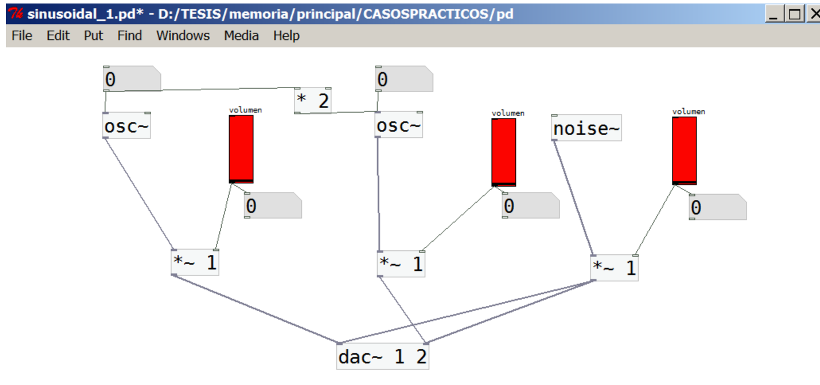


Figura 6.6.1 *Path* de Pure Data sinusoidal y ruido.

Debido a que no disponemos de una tarjeta de sonido virtual[128] hemos conectado las salidas físicas de PD con entradas físicas de la tarjeta de sonido. En la versión actual del programa no permite

Sonido. El sonido generado es de carácter periódico con un único armónico. El *patch* permite variar la frecuencia de la señal principal y modular su intensidad. Para tener un mayor grado de independencia, la señal principal y su armónico se distribuyen en canales diferentes. También hemos incorporado un generador de ruido blanco, que podemos mezclar con la señal sinusoidal y su armónico. Como resultado en cualquiera de las combinaciones tendremos dos líneas de sonido.

Imagen. Hemos realizado pruebas para las siguientes situaciones:

---

[128] Las tarjetas de sonido virtuales permiten interconectar el sonido entre diversas aplicaciones dentro del mismo sistema operativo.

a) Variación de la frecuencia de la sinusoidal sobre una única imagen y sobre sus tres canales de color. Algoritmos $(abid)$ y $(idab)$. El escalado final de la imagen con borde duro. Se utiliza únicamente el último buffer, el actual.
b) Ruido blanco sobre una única imagen, con el algoritmo $(abid)$, variando número de canales implicados y escalado duro. La intensidad del ruido blanco esta modulada por una sinusoidal que vamos variando, y la presentación de la imagen ocupa un tercio de la pantalla de visualización.
c) Variación de la frecuencia de la sinusoidal con un armónico sobre una única imagen. La frecuencia del armónico es siempre 8 veces más del de la fundamental. Cada una de las sinusoidales están en líneas diferentes, y van a un canal diferente de la imagen, la sinusoidal al rojo y el armónico al verde. En ambas líneas se aplica el mismo algoritmo que irá variando a lo largo del tiempo, pasando por todos los algoritmos simétricos presentados. El escalado es duro y la visualización es a pantalla completa.
d) Sobre el caso anterior mezclamos la señal resultante con ruido blanco.
e) Sobre el caso c) aplicamos los algoritmos que hemos denominado basados en reglas de comportamiento.

Grabación. La grabación de la sesión se ha realizado directamente sobre la pantalla usando el programa Camtasia Studio[129]. El sonido se ha grabado directamente del ambiente con el micrófono del ordenador portátil, con lo que es posible que además de los sonidos periódicos se incluyan algún sonido de ambiente. Se han generado los ficheros de video $c1a$, $c1b$, $c1c$, $c1d$ y $c1e$.

[129] El programa Camtasia Studio es un software que permite grabar video directamente desde la pantalla, en las últimas versiones también permite realizar una edición básica del video.

Conclusiones. Como conclusiones finales podemos apreciar los siguientes resultados para cada de los esquemas:

C1_a[130]

La visualización del algoritmo $abid$ presenta diferentes tipos de simetrías en función de las frecuencias que estemos visualizando. Es interesante resaltar que aproximadamente cada vez que incrementamos la frecuencia en 700Hz, se repite el mismo patrón, las simetrías primero son oblicuas con inclinación hacia la derecha, luego se estabilizan de forma vertical y finalmente vuelven a ser oblicuas pero con inclinación a la derecha. En la figura 6.6.2, vemos tres fotogramas de esta variación repetitiva, el primero con frecuencia de 2046, el segundo con 2066 y el tercero con 2091. Este patrón visual se vuelve a repetir tras 700Hz, esto es en 2746, 2766, 2791, y así sucesivamente.

[130] Los videos de este caso están en https://vimeo.com/user13711640, también se puede acceder poniendo en el buscador Vimeo Carlos García Miragall, y la referencia es c1a.

Figura 6.6.2. Patrón característico cada 700Hz[131].

También se ha observado que con ciertas frecuencias se genera una imagen que prácticamente permanece estática, como se puede apreciar en frecuencias en 409 Hz, 1174 Hz o 4500 Hz (ver figura 6.6.3)

[131] El indicador que tiene la imagen es una porción del patch de pd, que se ha incluido en la grabación para saber la frecuencia exacta que se estaba reproduciendo.

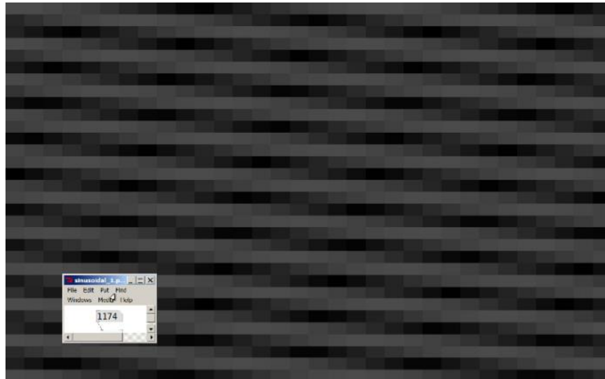


Figura 6.6.3. Simetrías estática en 1174 Hz.

Si en vez del algoritmo $abid$ usamos el algoritmo $idab$, obtenemos prácticamente los mismos resultados con pequeñas variaciones visuales. Se dan incluso los mismos patrones que hemos comentado anteriormente como vemos en la figura 6.6.4.

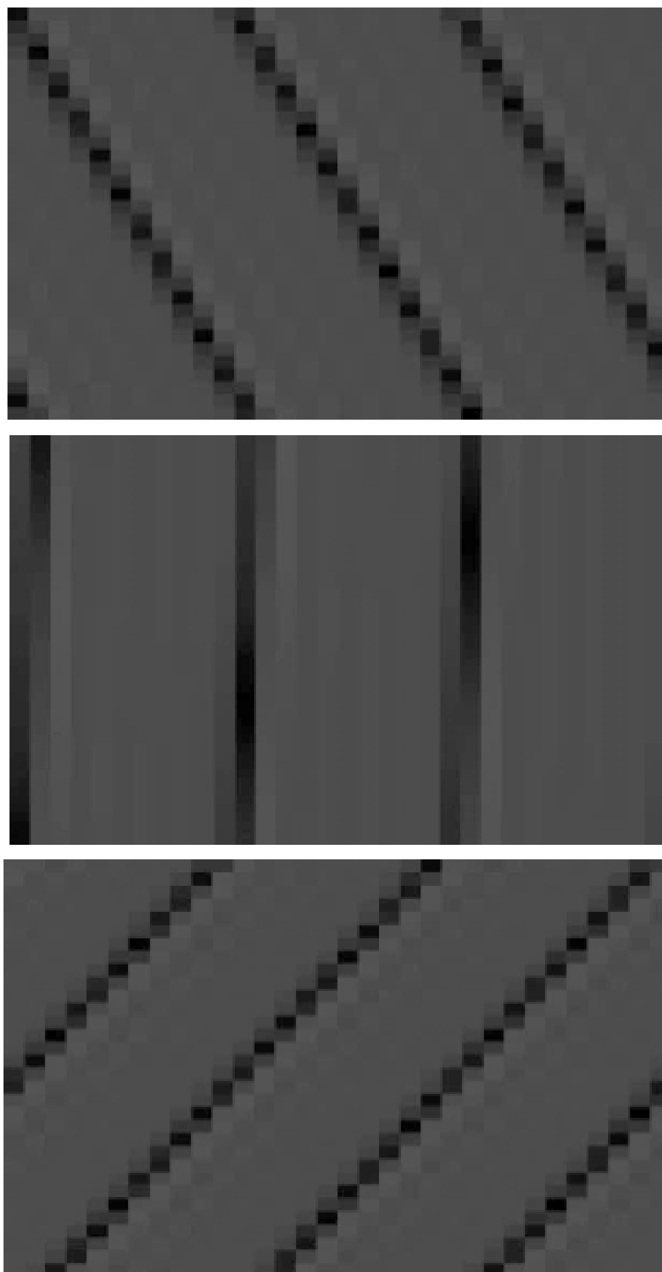


Figura 6.6.4. Patrón característico cada 700Hz.

C1_b

Cuando visualizamos ruido prácticamente el resultado, con independencia del algoritmo usado, siempre es el mismo, aleatoriedad y falta de simetrías (ver figura 6.6.5). Cuando el ruido se modula con una envolvente de intensidad, dicha variación sonora se aprecia visualmente, como podemos apreciar en la figura 6.6.6. En este caso a partir de frecuencias altas de la moduladora, sonoramente lo que percibimos es ruido blanco pero visualmente si vemos la modulación.

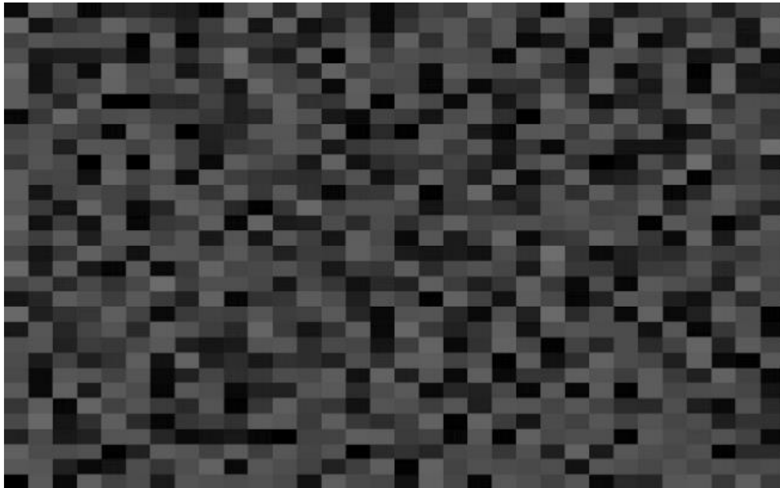


Figura 6.6.5. Visualización de ruido.

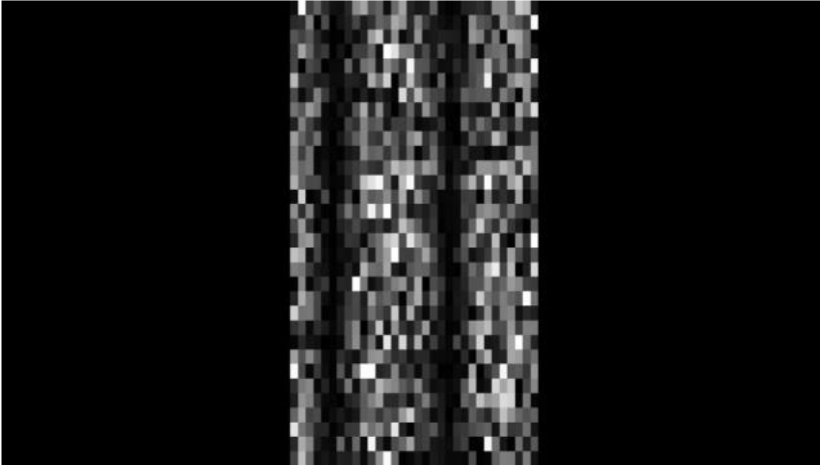


Figura 6.6.6. Visualización de ruido modulado.

C1_c

En este caso al generar cada canal con una línea diferente observamos la aparición de colores. Prácticamente el hecho de introducir un armónico no genera una gran variación con respecto a los casos anteriores. Aun así, podemos resaltar que para determinadas frecuencias se distinguen claramente las dos señales sonoras, ya que una sigue un patrón vertical y la otra un patrón horizontal, como vemos en la figura 6.6.7. En general se producen una gran cantidad de texturas como vemos en la figura 6.6.8, cuando variamos las frecuencias y el algoritmo.

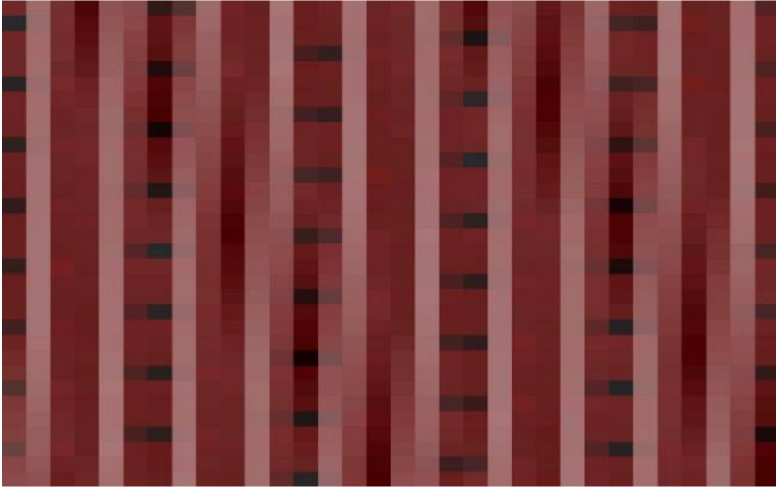


Figura 6.6.7. Sinusoidal y armónico.

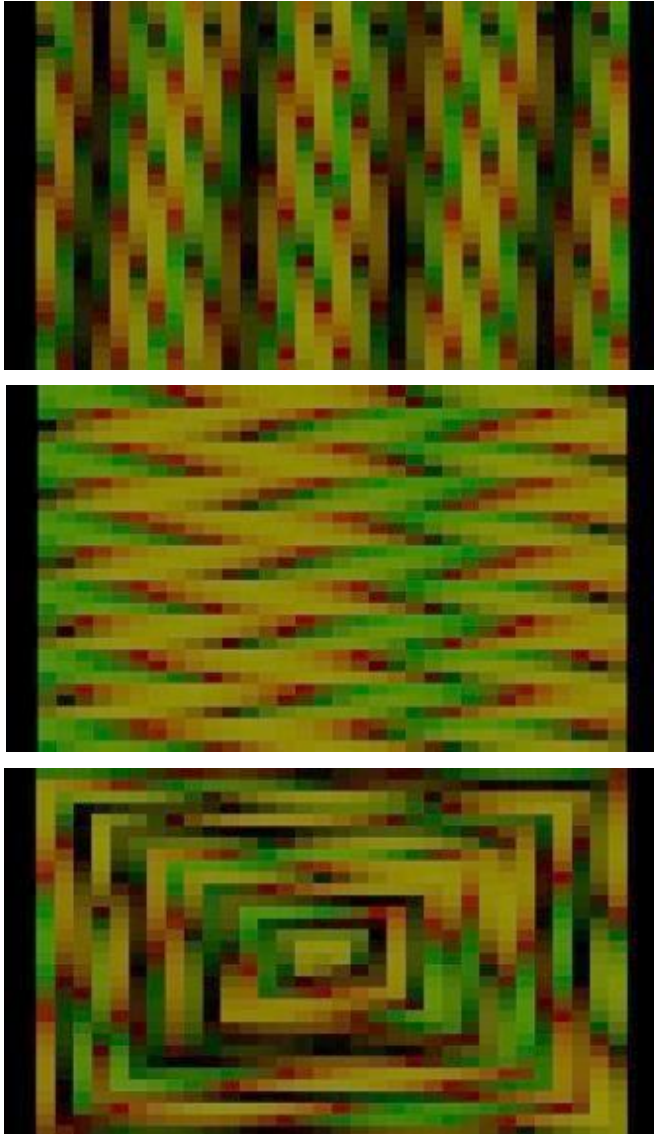


Figura 6.6.7. Fotogramas sinusoidal y armónico.

C1_d

Como en el caso b) al incorporar un ruido blanco en ambos canales el resultado que obtenemos es totalmente asimétrico. Dependiendo del

nivel de intensidad que tenga el ruido frente a los sonidos periódicos obtendremos diferentes resultados como podemos ver en la figura 6.6.8.

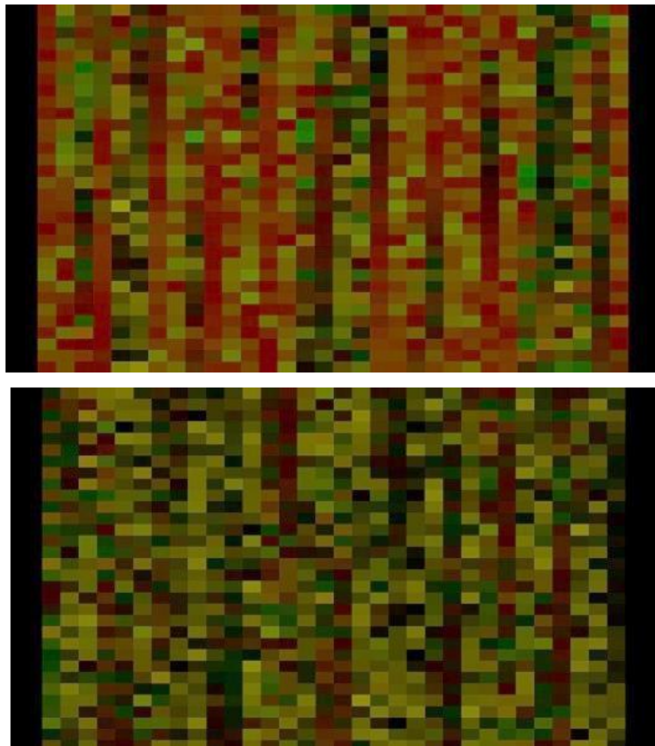


Figura 6.6.8. Visualización sinusoidal más ruido blanco con color.

C1_e

En esta última prueba de las señales periódicas hemos generado la imagen usando el algoritmo de la hormiga de Langton. El resultado parece similar al obtenido con el ruido, puesto que no mantiene simetrías, no obstante si se aprecia una sincronización entre el avance de la hormiga y la intensidad del sonido (ver figura 6.6.8a). Como conclusión podemos decir que el resultado es estéticamente

ruidoso pero se aprecia perfectamente la intensidad de la señal sonora. Debido a que el algoritmo es en sí una adaptación del problema de la hormiga de Langton, se pueden plantear muchas más variaciones, con la finalidad que el avance de la hormiga este mas sincronizado con la intensidad del sonido.

Figura 6.6.8a. Visualización sinusoidal con hormiga de Langton.

6.6.2 Caso 2. Guitarra eléctrica

Descripción general. En objetivo principal de este caso es estudiar el comportamiento visual de sonidos generados mediante guitarra eléctrica. Hemos realizado diversas pruebas buscando casos visualmente interesantes. Sobre el sonido principal de la guitarra se han aplicado diferentes efectos basados en retrasos. Como en el caso anterior se pretenden probar la mayoría de algoritmos estudiados en el apartado 4.

Lugar. Se trata de una prueba de laboratorio, realizada en el estudio de PDP11.

Aspectos Técnicos. El material técnico usado es el general de todas las pruebas. Para el sonido de la guitarra hemos utilizado una Fender JazzMaster de 1973 (ver figura 6.6.9), conectada a un *delay*[132] digital estéreo modelo DD-7 de la marca Boss y un *looper*[133] estéreo de la marca Ditto.

Figura 6.6.9. Guitarra Fender Jazzmaster.

Sonido. La gama de sonidos va desde sonidos limpios de guitarra sin procesar y procesados hasta sonidos almacenados y lanzados en bucle mediante el *looper*. Se han utilizado únicamente dos líneas de sonido. Debido a que el *delay* es estéreo, realmente en vez de duplicar una vez la señal, la duplica dos veces y efectúa un retraso diferente para cada una de las copias de la señal. Para alguno de los

[132] Un *delay* es un efecto de sonido que duplica la señal y la retrasa, en ocasiones sobre la señal retrasada se realiza algún proceso complementario.
[133] Un *looper* es un dispositivo capaz de grabar unos segundos de una señal sonora y posteriormente reproducirla de forma continua.

casos se ha usado el modelo de *delay* con reverso, esto quiere decir que la señal duplicada se invierte[134]. En general para todos los casos contemplados se han aplicado una gran variedad de tipos de sonido: acordes, melodías consonantes y disonantes, sonidos graves y agudos.

Imagen. Hemos realizado pruebas para las siguientes situaciones:

a) Las dos líneas de sonido van a la misma imagen y se aplica el mismo algoritmo para las dos $idab$ o $abid$. La línea 1 genera el canal verde y la línea 2 el canal rojo. Se usa el buffer principal. Escalado duro y suave.
b) Como en el caso anterior, una línea va al canal rojo y otra al verde, la diferencia es que cada uno de los canales sigue una estrategia diferente, en un caso tenemos que se aplica el algoritmo $idab$ y en el otro $abid$. Se usa el buffer principal y el escalado es duro.
c) Generamos una única imagen, donde las dos líneas de sonido van a los tres canales de color pero cada una de las líneas se genera mediante un algoritmo distinto, uno es $idab$ y el otro $abid$. Se usa el buffer principal y el escalado es duro.
d) Por cada una de las líneas generamos una imagen, y las líneas afectan a los tres canales de color y en cada imagen aplicamos un algoritmo diferente, en una $idab$ y en la otra $abid$. Las dos imágenes deben visualizarse en la misma pantalla, con lo que una de ellas debe generar un canal alfa con cierta transparencia para que deje ver a la otra imagen. Se usa buffer principal y escalado suave.

[134] El efecto de inversión supone reproducir la señal sonora desde el final hasta el principio, es como si reprodujéramos un disco de vinilo desde el final hacia el principio.

Grabación. La grabación visual de la sesión se ha realizado directamente sobre la pantalla usando el programa Camtasia Studio, y la sonora mediante el Reaper. Finalmente se han juntado en un único canal. La sesión se ha programado para que en una única toma se vaya pasando por las 4 situaciones que hemos planteado. Al final se genera un único archivo video con el resultado de los cuatro casos que irán apareciendo por orden, el archivo se denomina $c2$.

Conclusiones. Como conclusiones finales podemos apreciar los siguientes resultados para cada uno de los esquemas:

C2_a

El resultado visual es muy regular, para el algoritmo $idab$ se presenta una composición horizontal formada por bandas. Dependiendo de las frecuencias las bandas ascienden o descienden, los dos canales se distinguen claramente. En la figura 6.6.9, podemos observar tres fotogramas consecutivos de este caso, usando un escalado duro, y en la figura 6.6.10 observamos la misma situación pero con escalado suave. Se han apreciado algunos cambios en la textura básica cuando se tocan algunas notas muy agudas o se produce alguna distorsión generada por la forma de golpear las cuerdas.

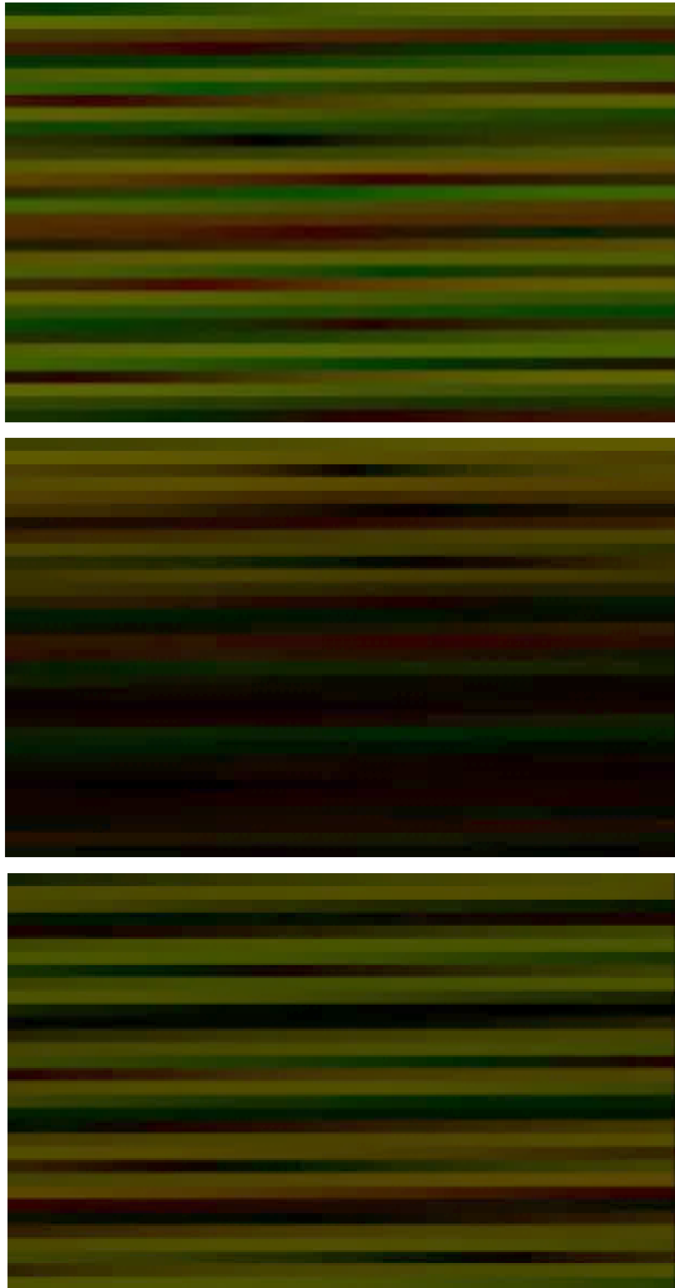
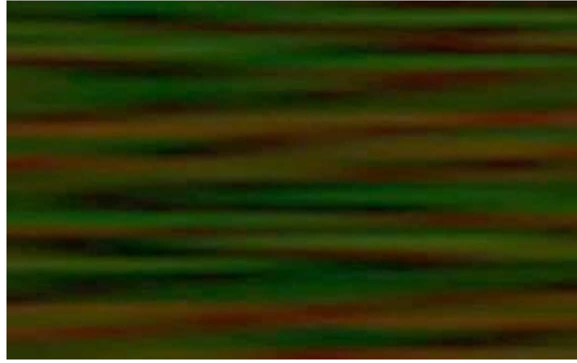
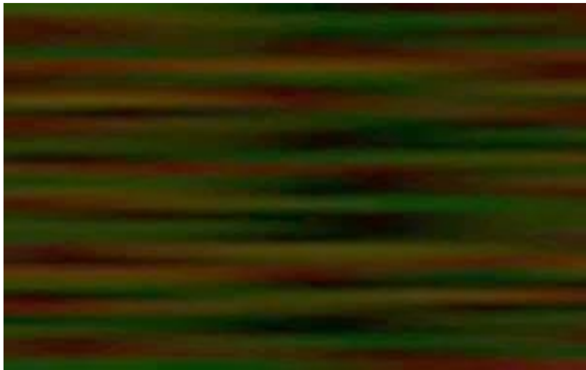


Figura 6.6.9. Fotogramas guitarra con algoritmo $idab$, escalado duro.





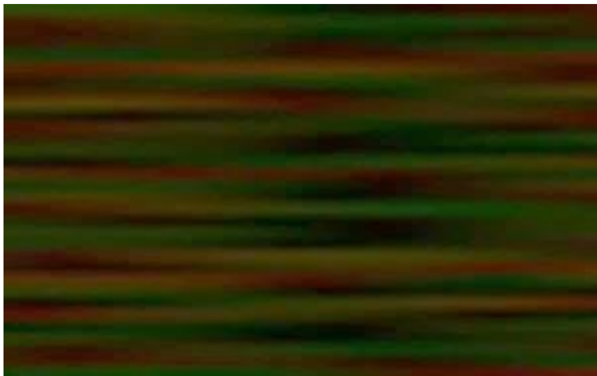


Figura 6.6.10. Fotogramas guitarra con algoritmo $idab$, escalado suave.

Cuando en vez del algoritmo $idab$, usamos el algoritmo $abid$ el resultado es similar pero las bandas se presentan de forma vertical, como vemos en la figura 6.6.11.

Figura 6.6.11. Fotogramas guitarra con algoritmo $abid$, escalado duro.

En ambos casos se presentan las simetrías y texturas con independencia del tipo se sonidos que se estén interpretando. En general se aprecia una sincronización perfecta con la intensidad del sonido, generando una correspondencia visual inmediata.

C2_b

Para este caso el resultado visual es también muy regular, como las líneas de sonido van retrasadas una de la otra por el *delay* de dos canales, se crea un patrón cuadriculado, al interaccionar las líneas horizontales con las verticales (ver figura 6.6.12). Las dos líneas de sonido se ven claramente.

Figura 6.6.12. Fotogramas guitarra con algoritmo combinado.

C2_c

Este caso es muy similar al anterior, la única diferencia es que los canales no están claramente separados, desde el punto de vista visual lo que tenemos son patrones rectangulares, que en ocasiones parecen agruparse generando ángulos que ascienden o descienden. La coloración es azulada y cuando se introduce algún ruido aparecen colores vivos, en la figura 6.6.13 vemos el efecto del ruido y en la figura 6.6.14 tres fotogramas seriados, donde se aprecian los ángulos.

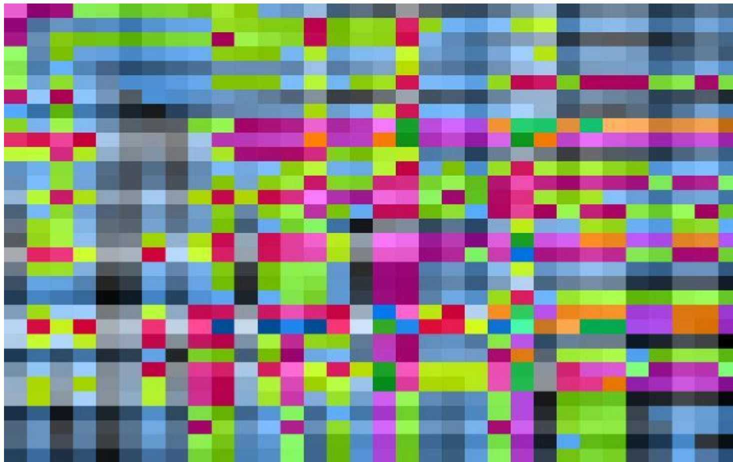


Figura 6.6.13. Aparición de ruido.

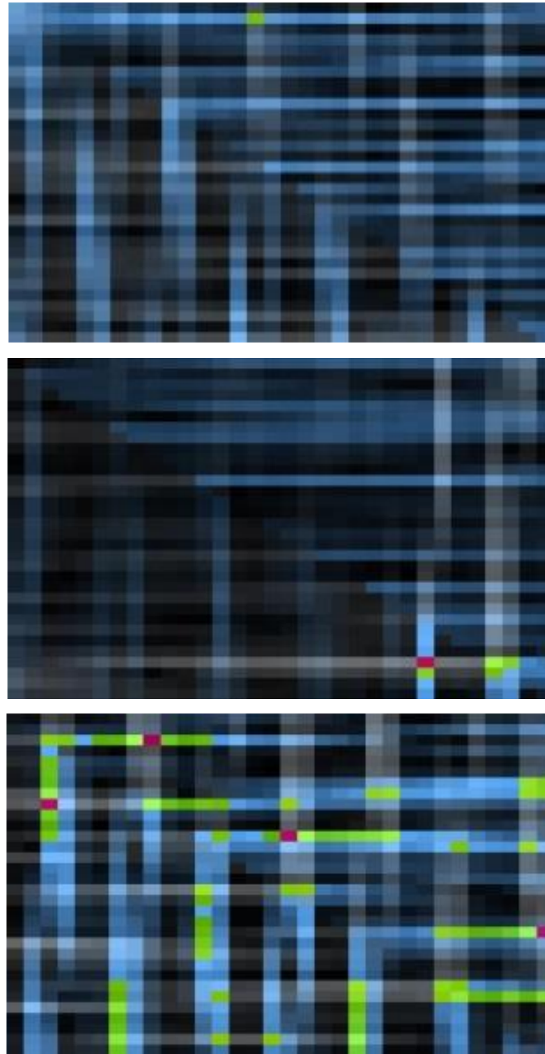


Figura 6.6.14. Fotogramas guitarra formando ángulos.

C2_d

Este caso sigue la tendencia de los anteriores desde el punto de vistas que se crean patrones rectangulares, al estar las dos líneas totalmente separadas en dos imágenes no se producen las pequeñas aberraciones de color que veíamos en el caso anterior. Como vemos

en la figura 6.6.15, cuando el escalado se realiza de forma suave aparecen unas formas más estilizadas que las puramente rectangulares. En general cuando usamos la interpolación suave las formas geométricas se diluyen y al suavizarse los contornos aparecen formas más definidas. El uso de escalados suaves supone una carga adicional de cálculo en el proceso de interpolación y eso repercute en una latencia mayor entre el sonido y la imagen, con el diseño de técnicas de *doble buffering* esto se podría evitar.

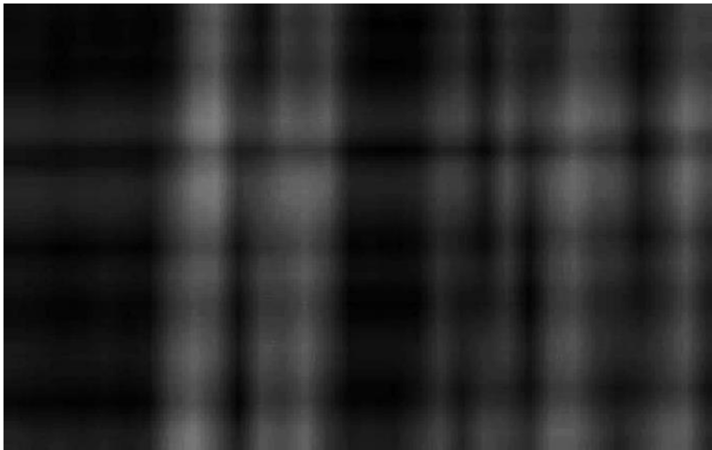


Figura 6.6.15. Formas rectangulares estilizadas.

6.6.3 Caso 3. Sintetizador

Descripción general. En objetivo principal de este caso es estudiar el comportamiento visual de sonidos generados mediante un sintetizador. Hemos realizado diversas pruebas buscando sobre todo casos interesantes desde el punto de vista visual. Como en el caso anterior se pretenden probar los algoritmos estudiados en el apartado 4.

Lugar. Se trata de una prueba de laboratorio, realizada en el estudio de PDP11.

Aspectos Técnicos. El material técnico usado es el general de todas las pruebas. Para el sonido de síntesis hemos usado un sintetizador de la marca Korg modelo microKorg XL. Se trata de un sintetizador donde se pueden programar los sonidos a través de su puerto USB, conectándolo a un programa de ordenador. Una de las salidas del sintetizador está conectada a un *looper* de un solo canal de la marca Vox modelo Dynamic Looper. Este *looper* permite aplicar efectos sobre el bucle almacenado.

Sonido. La gama de sonidos que puede generar el sintetizador es muy amplia desde sonidos muy cercanos a los instrumentos de percusión y por lo tanto cercanos al ruido, hasta sonidos muy afinados cercanos al piano. En todas las pruebas tenemos dos líneas del sintetizador, una de ellas está conectada con el *looper*. En general hemos aplicado una gran variedad de tipos de sonido: acordes, melodías consonantes y disonantes, sonidos graves y agudos, ritmos.

Imagen. Hemos realizado pruebas para las siguientes situaciones:

a) Las dos líneas de sonido van a la misma imagen. En una de las líneas aplicamos el algoritmo $idab$ sobre el canal azul y en la otra el algoritmo $abid$, sobre el canal verde. Sobre esta última línea aplicamos también el juego de la vida. Se usa el buffer principal y escalado duro.
b) En este caso hemos buscado un acabado en forma de cruz, usando 5 imágenes (ver figura 6.6.16). La imagen central se genera con una de las líneas de sonido donde aplicamos dos algoritmos diferentes sobre los mismos canales, $idab$ y $abid$. El resto de imágenes se generan con la otra línea se sonido, si la numeramos de izquierda a derecha y de arriba abajo, tenemos que la 1 y la 5 son iguales, mismos canales y mismo algoritmo y únicamente difieren en el buffer que usamos y la 2

y 4 tienen el algoritmo inverso de las otras dos y por lo demás igual. Los canales que usamos en todas las imágenes son el rojo y el verde y el azul que forman el cian, estas asignaciones van cambiando a lo largo del tiempo.

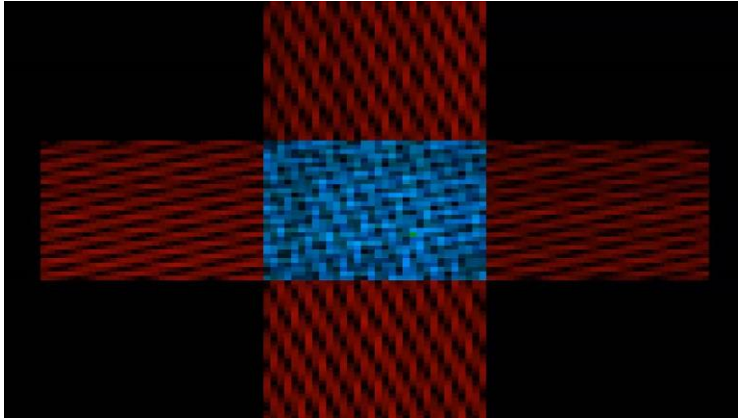


Figura 6.6.16. Visualización en forma de cruz.

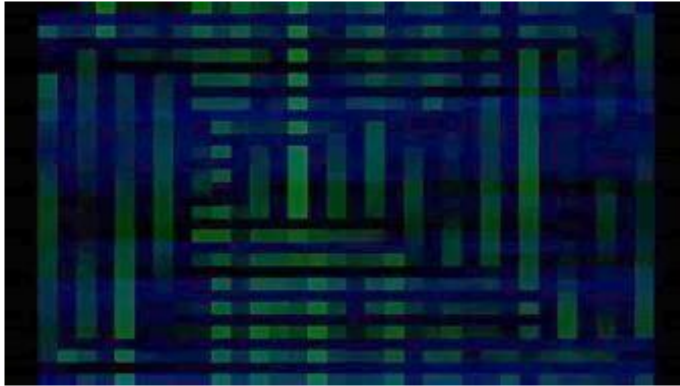
Grabación. La grabación visual de la sesión se ha realizado directamente sobre la pantalla usando el programa Camtasia Studio, y la sonora mediante el Reaper. Finalmente se han juntado en un único canal. Se han generado dos ficheros de video diferentes para cada uno de los dos casos propuestos $c3a$ y $c3b$.

Conclusiones. Como conclusiones finales podemos apreciar los siguientes resultados para cada de los esquemas:

C3_a

Cuando las dos líneas son simétricas al estar cada una presentada mediante un canal de video diferente y mediante una estrategia también diferente, genera una sensación visual de mayor cuerpo al

sonido, al visualizarse de varias formas (ver figura 6.6.17). Cuando los sonidos de los dos canales se separan, bien porque salen con diferentes retrasos o porque uno está grabado en el *looper* y el otro no, se genera las mismas sensaciones que ocurría con la guitarra separando mucho más los sonidos de cada una de las líneas. La combinación de un algoritmo sencillo como es $abid$, con el juego de la vida genera unas simetrías adicionales simulando la animación de una máscara. Como consecuencia de estas sensaciones visuales se plantea el introducir más mecanismos de ocultación de muestras.



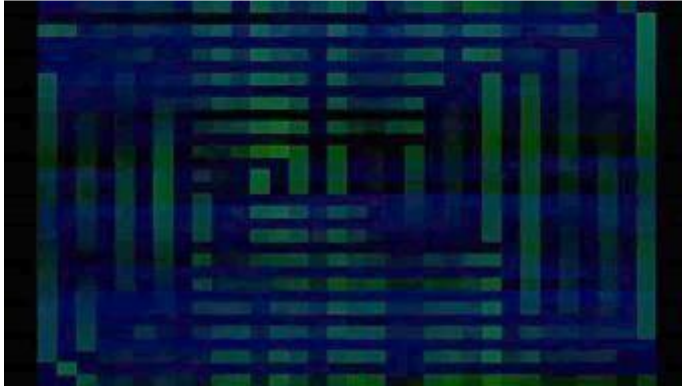


Figura 6.6. 17. Secuenciación de fotogramas caso 3a.

C3_b

Este caso representa una composición con más de dos imágenes y de nuevo la separación visual de las líneas de sonido hace que se perciban con mayor claridad. Con este caso relativamente sencillo se empiezan a apreciar las posibilidades creativas del método que proponemos. Aunque inicialmente parece algo limitado, conforme se empiezan a combinar los elementos del lenguaje se vislumbra el potencial creativo que tiene (ver figura 6.6.18).

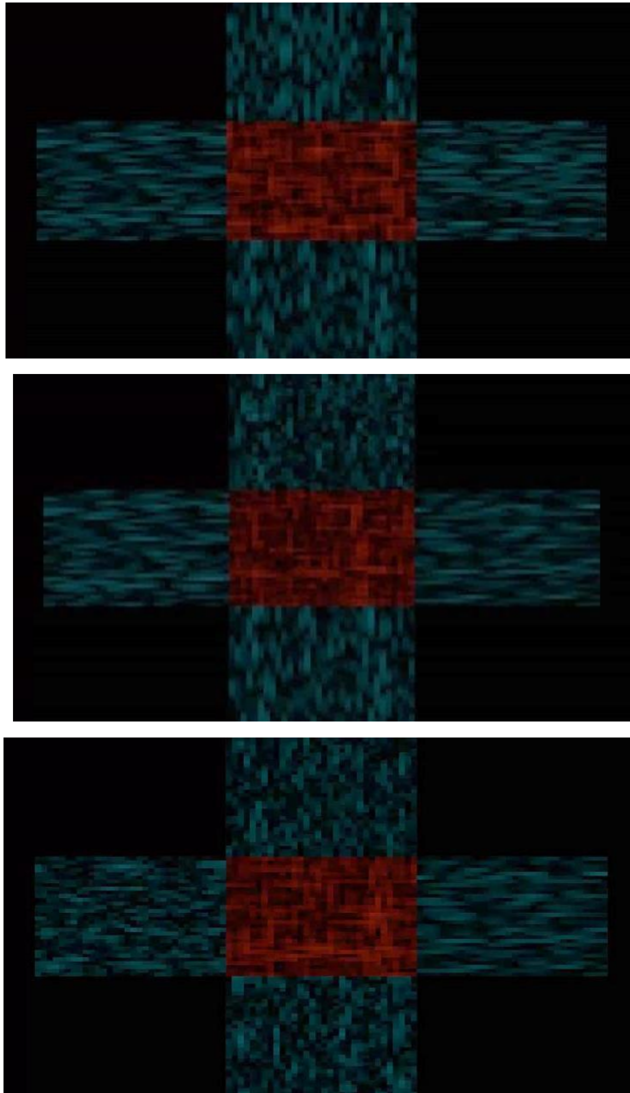


Figura 6.6.18. Secuenciación de fotogramas caso 3b.

6.6.4 Caso 4. Guitarra eléctrica y sintetizador

Descripción general. En objetivo principal de este caso es estudiar el comportamiento visual de sonidos generados mediante un sintetizador y mediante una guitarra eléctrica de forma simultánea.

Las pruebas han buscado principalmente la separación visual de los dos instrumentos.

Lugar. Se trata de una prueba de laboratorio, realizada en el estudio de PDP11.

Aspectos Técnicos. El material técnico usado es el general de todas las pruebas. Para el sonido de la guitarra hemos utilizado una Fender JazzMaster de 1973, conectada a un *delay* digital estéreo modelo DD-7 de la marca Boss y un *looper* estéreo de la marca Ditto. Para el sonido de síntesis hemos usado un sintetizador de la marca Korg modelo microKorg XL. Una de las salidas del sintetizador está conectada a un *looper* de un solo canal de la marca Vox modelo Dynamic Looper. Este *looper* permite aplicar efectos sobre el bucle almacenado.

Sonido. La gama de sonidos es muy amplia ya que además de los sonidos de guitarra tenemos los sonidos generados por el sintetizador. Para los sonidos de síntesis se han usado dos líneas y para los sonidos de guitarra otras dos. Durante el desarrollo de la improvisación en ocasiones el sonido de síntesis forma un colchón sonoro de fondo y la guitarra unas líneas melódicas, pero en otros momentos se han invertido los papeles. Lo que se ha buscado es una amplia riqueza de sonidos.

Imagen. Para la generación de la imagen hemos usado dos imágenes, una para las dos líneas de la guitarra y la otra para las dos líneas del sintetizador. Todas las líneas de sonido afectan al canal verde y azul. Para la imagen de la guitarra una de las líneas se visualiza usando el algoritmo de espiral de dentro hacia afuera y la otra línea lo hace al contrario de fuera hacia adentro. Las dos líneas

de síntesis generan la otra imagen usando un algoritmo de izquierda a derecha y de arriba abajo.

Grabación La grabación visual de la sesión se ha realizado directamente sobre la pantalla usando el programa Camtasia Studio, y la sonora mediante el Reaper. Finalmente se han juntado en un único canal. Se ha generado un único fichero de video $c4$.

Conclusiones. En todo momento visualmente los dos sonidos se diferencian claramente, esto supone que se aprecian mucho mejor sus características distintivas. Otro de los resultados que queremos resaltar es el uso de algoritmos contrarios sobre sonidos muy similares, en este caso sobre cada línea de la guitarra hemos aplicado una dirección diferentes de la espiral, también ayuda a remarcar mucho más el efecto sonoro del estéreo. En la figura 6.6.19 podemos ver algunos fotogramas generados para este caso.

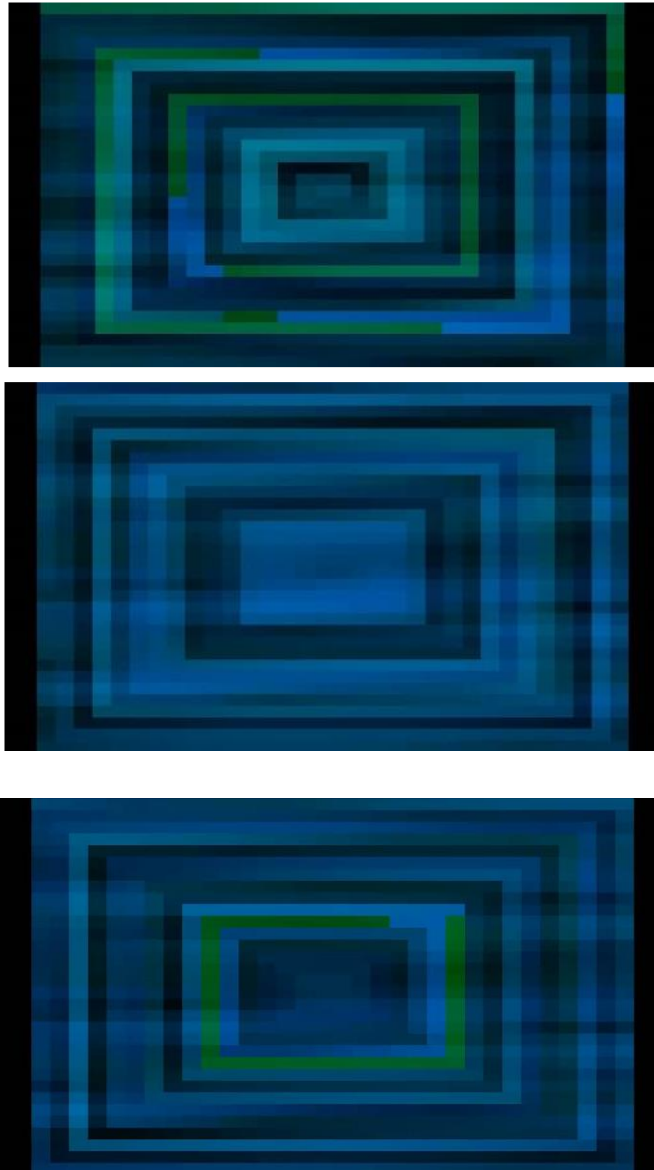


Figura 6.6.19. Secuenciación de fotogramas casos 4.

6.6.5 Caso 5. Improvisación PDP11.

Descripción general. En objetivo principal de este caso es visualizar un ensayo de PDP11 y estudiar el comportamiento de los algoritmos. La idea que hemos planteado es que cada músico tuviera una imagen con un algoritmo de ubicación sencillo, y vaya adaptando su improvisación en función de los resultados visuales. Ha participado Deco Nascimento a los sintetizadores.

Lugar. Se trata de una prueba de laboratorio, realizada en el estudio de PDP11.

Aspectos Técnicos. El material técnico usado es el general de todas las pruebas. Para el sonido de la guitarra hemos utilizado una Fender JazzMaster de 1973 (ver figura 6.6.9), conectada a un *delay* digital estéreo modelo DD-7 de la marca Boss y un *looper* estéreo de la marca Ditto. Para los sonidos de síntesis se ha usado un ordenador portátil MacBook, en el que está instalado el software Ableton Live y el Reason de Propellerhead, un controlador Trigger Finger de MAudio y sintetizador Korg XL (ver figura 6.6.19a).

Figura 6.6.19a. Deco Nascimento con parte de su equipo.

Sonido. La gama de sonidos es muy amplia ya que además de los sonidos de guitarra que ya hemos comentado en el caso2, en esta prueba se han combinado los sonidos generados por el sintetizador y por los programas Live y Reason. Para los sonidos de síntesis se han usado dos líneas y para los sonidos de guitarra otras dos. La idea de este caso ha sido realizar una improvisación libre sin apenas directrices, lo que ha ido surgiendo sobre la marcha. La única guía ha sido estar pendiente de la visualización y buscar correspondencias entre el sonido y las texturas obtenidas.

Imagen. Para la generación de la imagen hemos usado únicamente los algoritmos $idab$ o $abid$. Por un lado la elección ha sido mantener durante todo el tiempo el mismo concepto de ubicación de muestras tratando de buscar variaciones y por otro lado cuando se realizó la improvisación eran los únicos algoritmos que teníamos implementados. Cada una de las líneas de sonido genera una imagen diferente. La pantalla de visualización se ha dividido en 4 zonas para albergar cada una de las imágenes (ver figura 6.6.20)

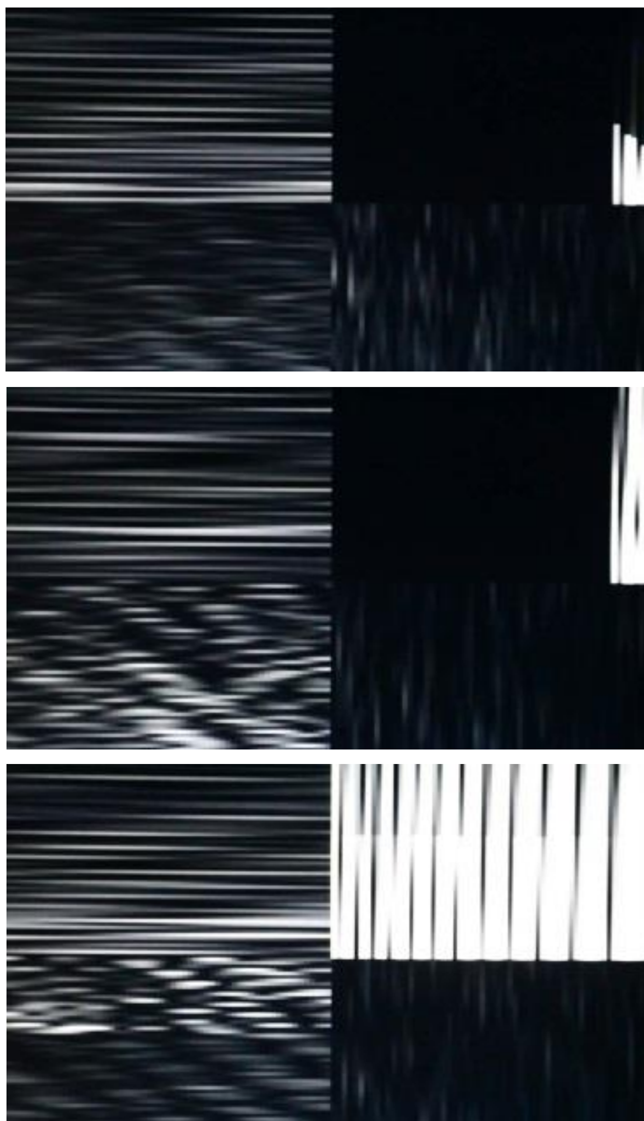


Figura 6.6.20. Visualización con 4 zonas.

Grabación. La grabación de la sesión se ha realizado directamente sobre la pantalla usando cámara Canon 550D. El sonido en postproducción se ha intentado limpiar un poco ya que su grabación

se había realizado directamente con el micrófono de la cámara. Se ha generado el fichero de video $c5$.

Conclusiones. La idea de realizar una improvisación en función de la visualización ha resultado ser muy interesante de cara al resultado visual y al sonoro, puesto que las variaciones sonoras se han visto directamente influenciadas por las variaciones visuales. En algunos momentos se ha profundizado en determinados esquemas sonoros por el simple hecho de ser visualmente atractivos. Un caso especial ha sido cuando con los sonidos del sintetizador ha aparecido la textura que podemos ver en el primer cuadrante de la figura 6.6.21, a partir de ese momento la improvisación ha girado en torno a ese sonido.

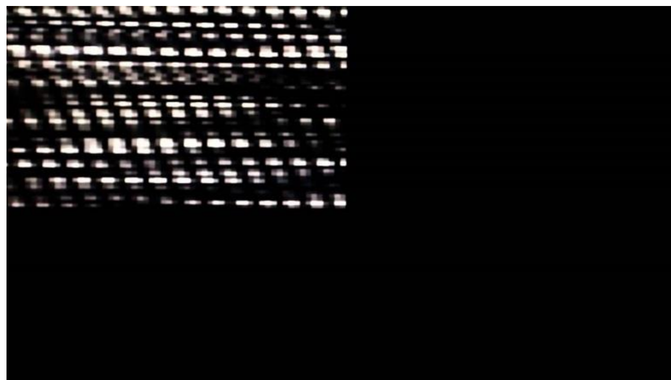


Figura 6.6.21. Textura guía de inprovisación.

Con todo esto podemos concluir que en esquemas de improvisación la visualización realiza las funciones de dirección o guía, debido a que la música se deja llevar por los aspectos visuales. Cuanto mayor sea el dispositivo de visualización más inmersivo resulta el proceso para el músico.

6.6.6 Caso 6. Anteproyecto *Lux Interior* PDP11

Descripción general. En objetivo principal es visualizar un ensayo de PDP11 para el proyecto Lux Interior, estudiando el comportamiento de los algoritmos de ubicación. *Lux Interior* es una performance sonora y visual que se presentará el día 26 de Enero de 2016 en el palacio Cerveró de Valencia. El concierto se enmarca dentro de la exposición *En clave de luz* que realiza el Laboratorio de Luz junto a la Real Sociedad Española de Física y de Óptica, con motivo del año internacional de la luz. Lo que presentamos a continuación es el primer ensayo que hemos realizado sobre el proyecto. Ha participado Paco Sanmartin al bajo acústico.

El principal concepto de la pieza es la luz, jugando con esa idea pensamos en *Lux Interior*[135] en relación al cantante del grupo de *psychobilly The Cramps* (ver figura 6.6.22). *Lux Interior* representa un icono de la historia del rock por su originalidad y sus aportaciones musicales. Revisando sus trabajos nos llamó la atención un concierto que realizó el grupo el 13 de Junio de 1978 para los pacientes de una clínica mental del estado de California, se trata de un concierto muy interesante sobre todo por la forma especial con la que se interactúa con los enfermos mentales.

[135] Lux Interior es el nombre artístico de Erick Lee Purkhiser, vocalista de The Cramps.

Figura 6.6.22. *Lux Interior The Cramps*.

A raíz de *Lux Interior*, encadenamos con la idea de cómo realiza la escucha un niño cuando todavía está en el vientre de su madre, estableciendo una relación directa entre la luz o vida en interior del útero y su forma de percibir el mundo que le rodea. Finalmente con esos conceptos en mente, *Lux Interior* como icono del rock y *Luz Interior* como niño en el interior del útero materno, realizamos una narrativa sencilla con el objetivo de estructurar la performance. La estructura narrativa de la performance se divide en 3 partes:

1. Johnny, un niño todavía en el vientre de su madre, está tranquilo escuchando sus latidos y los de su madre, los ruiditos típicos de la casa y sobre todo la voz de su madre, Mary.

2. Freddy, el padre del niño es enfermero de una clínica mental, llega a casa muy contento diciendo que *Poisson Ivy*[136] a vuelto a juntar a The Cramps para hacerle un último homenaje a *Lux Interior* en la clínica. Mary se pone muy contenta y empieza a rebuscar en el armario buscando sus viejos botines de gamuza dorada para rememorar viejos tiempos.

[136] *Poison Ivy* es el nombre artístico de Kristy Wallace, guitarrista de The Cramps.

3. En el concierto suena uno de los temas clásicos de The Cramps *Human Fly*.

A partir de estas ideas hemos estructurado el concierto, estableciendo las guías básicas de composición e improvisación. Cada una de las partes tiene una duración de aproximadamente 6 minutos.

Lugar. Se trata de un ensayo para el proyecto Lux Interior, realizado en el estudio de PDP11.

Aspectos Técnicos. El material técnico usado es el general de todas las pruebas. Para el sonido de la guitarra hemos utilizado una Fender JazzMaster de 1973, conectada a un *delay* digital estéreo modelo DD-7 de la marca Boss y un *looper* estéreo de la marca Ditto. Para el sonido de bajo se ha utilizado un bajo acústico Whashburn AB-10 procesado con Pure Data (ver figura 6.6.23). Para el sonido rítmico de síntesis se ha diseñado un *patch* de Pure Data.

Figura 6.6.23. Bajo acústico Whashburn AB-10.

Sonido. La primera parte se ha desarrollado mediante un *patch* de síntesis en PD, se trata de una base sintética simulando el latido del corazón de Mary, con estructura de rock and roll. Este latido esta modulado con el sonido del bajo acústico. Las aportaciones de la guitarra a esta parte representan el latido del niño, mucho más rápido que el ritmo básico de síntesis, y ruiditos esporádicos, representando los sonidos del ambiente. El sonido del bajo en esta parte sigue las mismas directrices que la guitarra. En la segunda parte, el latido sintético se modifica tímbricamente y la guitarra y el bajo además de apoyar con sonidos ambientales, generan tensiones conjuntas, emulando la alegría de Mary. En la tercera parte se presenta una versión desestructurada de la canción emblemática The Cramps, *Human Fly*. Durante prácticamente todo el concierto se escuchan los ecos de ese rock and roll.

Imagen. Para la generación de la imagen hemos usado todos los algoritmos simétricos. Hemos realizado 5 imágenes, dos para la guitarra, una para el bajo y otra para el sonido sintético de PD. La presentación se ha realizado en forma de cruz como podemos ver en la figura 6.6.24. Si hacemos un recorrido de izquierda a derecha y de arriba debajo sobre la cruz resultante, tendremos que el primer rectángulo se corresponde con el bajo y el canal de color usado es el rojo, el siguiente es una línea de la guitarra y el canal usado es el rojo también, el rectángulo central mezcla las dos líneas de la guitarra pero usando diferentes *buffers*, con lo que el instante visual es un poco anterior al sonoro, el siguiente rectángulo es la otra línea de la guitarra sobre canal verde, por último tenemos el rectángulo de la base sintética de PD, sobre el canal verde. El algoritmo de ubicación de muestras va variando a lo largo del tiempo.

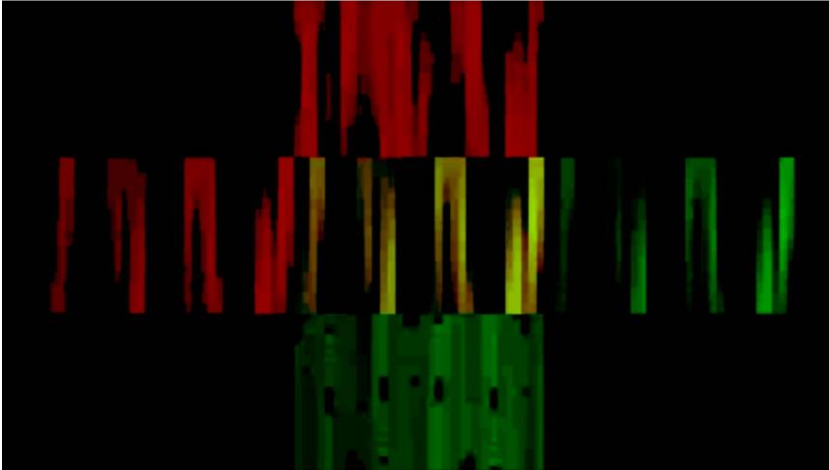


Figura 6.6.24. Visualización con 4 zonas en forma de cruz.

Grabación. La grabación de la imagen de la sesión se ha realizado directamente sobre la pantalla usando el programa Camtasia Studio, y la del sonido usando el programa de sonido Reaper. Posteriormente se ha reunificado la grabación sonora y la visual. Se ha generado el fichero de video $c6$.

Conclusiones. El caso que hemos propuesto es la primera fase en la elaboración de un concierto o performance sonora de PDP11. Debido a las fechas en las que hemos tenido que cerrar esta memoria, el concierto ni se ha realizado ni está todavía preparado del todo. No obstante nos ha parecido interesante establecer como último caso esta primera fase en la elaboración de un proyecto, que habitualmente se elude. El resultado visual nos sirve de base para realizar un análisis y tomar las decisiones con respecto a los algoritmos a usar en cada uno de las partes del concierto. Tras este ensayo pensamos que es una buena idea mantener las visualizaciones separadas de los diferentes instrumentos, es posible que el formato en cruz se descarte y se termine proyectando de forma

horizontal con dos proyectores. Todo esto dependerá de la puesta en escena final. El uso de los canales de color frente a un resultado en tonos de grises también es muy posible que se descarte, debido principalmente a que este concierto se realiza como PDP11 y como miembros del grupo de investigación Laboluz, y la estética habitual del Laboluz es en tonos de grises. En la figura 6.6. 25 vemos algunos fotogramas de este caso.

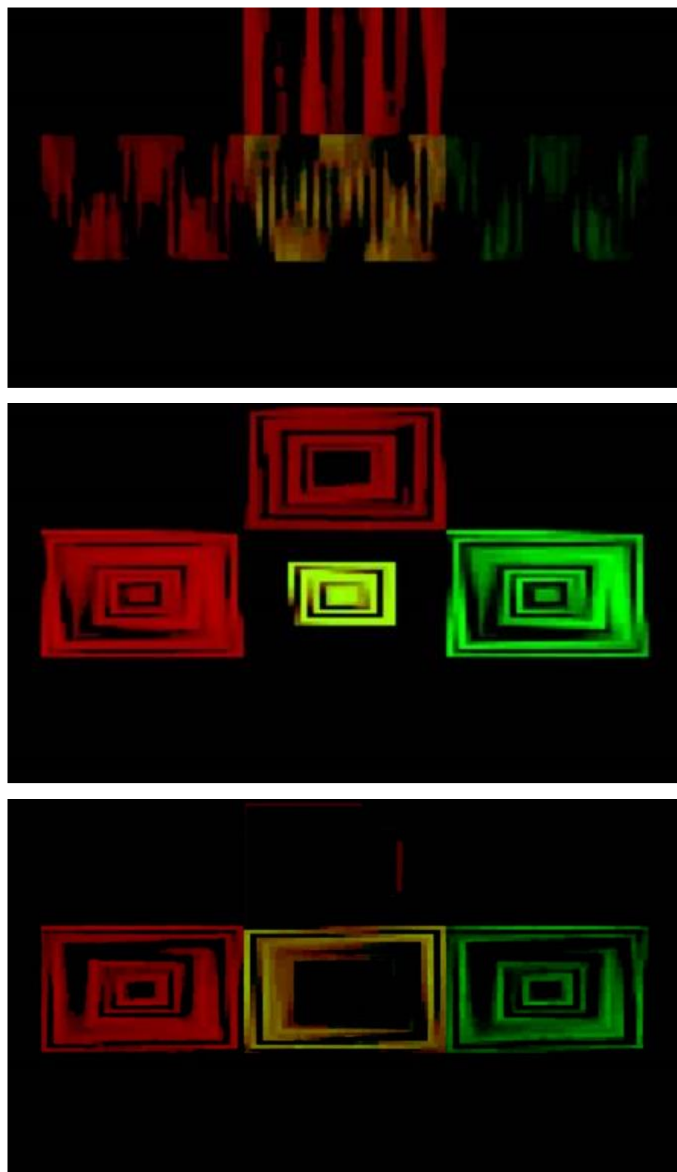


Figura 6.6.25. Fotogramas *Lux Interior*.

6.7 Videos en Vimeo

Todos los trabajos realizados para este apartado de casos prácticos están ubicados en Vimeo. Esta red social para la compartición, publicación, y almacenamiento de videos digitales se caracteriza por no admitir ni publicidad, ni demostraciones de videojuegos, ni pornografía. Es la red habitual de trabajos artísticos y experimentales. A fecha de cierre de este trabajo de investigación se pueden consultar todos los videos a los que hace referencia este apartado, el etiquetado de los videos es similar al etiquetado de los casos, con la finalidad que se puedan ver mientras se realiza la lectura del apartado. Por ejemplo para el $caso\ 1$, que tenemos varios subapartados, los videos están identificados como $c1a1, c1a2 \ldots$

Dentro de Vimeo los videos están dentro de un canal que hemos denominado Videos Cíclope[137], y asociado a cada uno de los videos hay una breve descripción de sus características. Desde el punto de vista técnico tenemos que:

Usuario Vimeo https://vimeo.com/user13711640

[137] Los videos también se encuentran en la página principal del usuario.

7 CONCLUSIONES

El método de visualización que proponemos se fundamenta en la idea que el lenguaje con el que almacenamos digitalmente el sonido y las imágenes es el mismo, es lenguaje de los números. A través de sucesiones numéricas etiquetadas en el tiempo y unas reglas sencillas de interpretación podemos representar sonidos e imágenes. Conceptualmente estos números representan intensidad sonora o intensidad visual. Si lo que pretendemos es ver el sonido, tendremos que ver su intensidad y tendremos que verla en el tiempo que se produce, de manera que el método que hemos propuesto consiste en ir ubicando espacialmente las muestras de sonido en una secuencia de imágenes generada con el devenir del tiempo. La idea principal del método radica en mantener en la representación visual los valores numéricos de la intensidad del sonido y la correlación temporal.

Para distribuir espacialmente las muestras de sonido hemos planteado muchas estrategias. Las que hemos denominado simétricas tienen unas reglas simples para distribuir las muestras de sonido en la imagen, como de izquierda a derecha y de arriba hacia abajo o desde dentro hacia afuera en forma de espiral. También hemos planteado algunas con reglas de comportamiento más sofisticadas como las basadas en el problema de la hormiga de Langton, o en el juego de la vida. Y por último hemos planteado como mejor estrategia, aquella que se obtienen como resultado de combinar varias de las estrategias básicas, pues se fundamenta en el principio fundamental del sonido que es el cambio y en el principio de simplicidad de los lenguajes de programación que con unas reglas muy sencillas se pueden resolver problemas complejos.

Para analizar los resultados visuales de esta propuesta hemos desarrollado una aplicación en el lenguaje de programación Java.

Para el desarrollo de la aplicación hemos seguido la metodología típica de ingeniería del software y hemos tratado de estructurar adecuadamente la aplicación para poder realizar futuras ampliaciones. Nos ha sido complicado resumir las características del programa realizado debido a que conforme se ha ido desarrollando la memoria, el propio programa ha ido cambiando. No obstante pensamos que las aplicaciones de autor son entidades vivas y están continuamente transformándose. Desde el principio hemos buscado que la aplicación se pueda utilizar para eventos de directo, como conciertos y performance.

A través de los casos de prueba hemos comprobado cómo las visualizaciones mantenían buena parte de las propiedades físicas y perceptuales del sonido, comprobando de esta forma la hipótesis que nos planteamos al inicio de la investigación. Los resultados visuales, con independencia de las estrategias de ubicación espacial usadas, presentan una estética propia del arte abstracto. Este resultado era el esperado debido a que justamente el sonido y la música son medios de naturaleza abstracta. Podemos decir que el método propuesto visualiza el sonido de una manera similar a cómo lo hacen las personas con capacidades sinestésicas.

La propiedad de intensidad sonora es la que más claramente se aprecia en la imagen ya que es la relación que establece el método de forma directa. No obstante el método llega a visualizar muchas más propiedades del sonido. Cuando el sonido es periódico, como es el caso de los sonidos producidos por instrumentos musicales, en la mayoría de las interpretaciones visuales se observa la aparición de simetrías que están directamente relacionadas con la frecuencia fundamental de la señal sonora y sus armónicos. Conforme el sonido se van volviendo más complejo, esto es el número de armónicos y parciales es mayor, las simetrías no son tan precisas, pero se siguen

manteniendo. La visualización de los ruidos presenta una estética aleatoria, se obtienen los mismos resultados al visualizar ruido que al visualizar sonidos muy periódicos con algoritmos muy aleatorios. Sin duda es en los sonidos afinados donde hemos encontrado mayor riqueza y diversidad visual. En composiciones sonoras rítmicas también las imágenes generadas establecen estructuras simétricas temporales. El efecto visual más importante se genera en la concordancia que se establece en el devenir del tiempo, por lo que para apreciar el proceso sinestésico que provoca el método propuesto se debe invertir un determinado tiempo a través de una mirada y una escucha atenta.

El trabajo pensamos que aporta nuevas ideas en el campo de la visualización del sonido. Se enmarca dentro de la corriente de abstracción que entendemos es el nicho natural de la estética del sonido. Y pese a que

Este trabajo de investigación propone una nueva vía de visualización los resultados no son tan llamativos como muchas de las propuestas actuales que hemos estado revisando en el apartado 2, si es honesto con la idea de visualización del sonido en toda su pureza. del sonido con fines artísticos, las muestras del sonido son colocadas en la imagen como si de un rompecabezas se tratara. Con este trabajo hemos querido dar los primeros pasos de este nuevo lenguaje de visualización. Fundamentalmente nos hemos basado en los algoritmos simétricos de colocación de muestras, y ya se aprecian los múltiples y variados resultados obtenidos. Al igual que los lenguajes formales y los lenguajes de programación que con un conjunto muy pequeño de reglas y palabras, se pueden expresar infinidad de cosas, nuestro método en base a unas reglas sencillas puede potencialmente crear un gran universo de visualizaciones.

El trabajo finalmente podemos decir que recupera las ideas de los primeros intentos de establecer relaciones entre el sonido y la imagen, pues establece un marco de conexión muy simple a través de los números y por otro lado se enmarca en las últimas tendencias de los sistemas informáticos, proponiendo un lenguaje que ante una apariencia simple puede evolucionar hasta convertirse en un verdadero referente de conexión entre la imagen y el sonido. Podemos decir que este trabajo se engloba perfectamente en las tendencias actuales de establecer conexiones entre diferentes áreas de conocimiento.

Con la finalidad de presentar el método de forma natural no hemos querido introducir elementos externos a las muestras de sonido, pero somos conscientes que esto enriquecería muchísimo más las posibilidades visuales.

Como futuro trabajo nos planteamos enriquecer el método con un mayor número de estrategias de visualización basadas en reglas de comportamiento. Además de realizar visualizaciones híbridas componiendo imágenes reales, imágenes sintéticas, e imágenes obtenidas de las muestras de sonido.

Otro trabajo futuro es desarrollar más la aplicación informática, Ciclope, con la finalidad que el usuario pueda introducir de forma más natural la evolución y los cambios de la visualización, también se plantea ampliar su funcionalidad para adaptarla al campo del *vjing*, de manera que el usuario tenga una interacción más directa con la aplicación. Somos conscientes que la aplicación se encuentra en una versión beta y nos gustaría que esta versión evolucionara hasta convertirse en una aplicación de uso más generalizado.

Otra de las posibles líneas futuras es presentar las muestras en un espacio 3D en vez de 2D. El universo 3D permitiría la existencia visual, de forma muy clara, de las muestras pasadas. Además la experiencia visual posiblemente sería mucho más envolvente e inmersiva.

Otra línea de trabajos futuros sería abordar siguiendo el mismo método la visualización de otros tipos de datos o la sonificación de la imagen. De la misma forma que podemos ver el sonido, también podemos escuchar la imagen. En general se trata de un problema de traducción de lenguajes.

Por último nos gustaría resaltar que este trabajo de investigación es una primera aproximación a las posibilidades que plantea el método propuesto y pese a su sencillez se empieza a vislumbrar el gran potencial que tiene para establecer conexiones entre el sonido y la imagen.

8 BIBLIOGRAFÍA

Allen, P. y Dannenberg, R. (1984). "Tracking musical beats in real time" en *Proceedings of the International Computer Music Conference. International Computer Music Association*. 140–143.

Alonso, F., Martínez, L. y Segovia, J. (2005). *Introducción a la Ingeniería del Software: Modelos de Desarrollo de Programas*. Delta Publicaciones.

Aucouturier J. J. y Sandler M. (2001). "Segmentation of Musical Signals using Hidden Markov Models" en *Proc. 110th Convention of the Audio Engineering Society*.

Bakedano, J. (1987). *Norman McLaren. Obra completa 1932-1985*. Museo de Bellas Artes de Bilbao.

Barton McLean (1992). "Composition with Sound and Light" en *Leonardo Music Journal*. vol. 2, p. 13-18.

Bayle, F. (1989). "Image-of-sound, or i-sound: Metaphor/metaform " en *Contemporary Music Review*. vol. 4, p. 165–70.

Benavides, Lidia (2010). "Cuatro propuestas sobre VideoArte y Cine Experimental" en *Arte, Individuo y Sociedad*. vol. 22 (1), p. 163-174.

Bishop, Bainbridge (1893). *A souvenir of the color organ, with some suggestions in regard to the soul of the rainbow and the harmony of light*. The de Vinne Press.

Brougher, K. et al (2005). *Visual Music: Synaesthesia in Art and Music since 1900*. Los Angeles: The Museum of Contemporary Art.

Burnham, Jack (1968). *Beyond Modern Sculpture: the effects of science and technology on the sculpture of this century*. New York: George Braziller.

Church K. y Helfman J. (1993). “Dotplot: A Program for exploring Self-Similarity in Millions of Lines of Text and Code” en *Journal of American Statistical Association.* vol. 2, issue 2, p.153–174.

Cleveland W. S. (1993). *Visualizing Data*. Hobart Press.

Clynes, M. (1977). *Sentics*. New York: Anchor Press.

Collopy, Fred. (2001). “Improvisational Lumia: Painting along with Musicians” en *Leonardo*, vol 34 issue 4, p. 353.

Collopy, F., Fuhrer, R. y Jamenson, D. (1999). “Visual Music in a Visual Programming Language” en *1999 IEEE Sysmposium on Visual Languajes*, Tokyo, Japan. 111-118.

Cooper, M. y Foote, J. (2002). “Automatic Music Summarization via Similarity Analysis” en *Proc. International Symposium on Music Information Retrieval*. Paris, Francia.

Cooper, M., Foote, J., Pampalk, E. y Tzanetakis, G. (2006). “Visualization in Audio-Based Music Information Retrieval” en *Computer Music Journal*, vol. 30, issue 2, p. 42-62.

Cytowie, Richard (1993). *The Man Who Tasted Shapes*. New York: G. P. Putnam's Sons.

Daniels D. et al (2009). *See this sound: Audiovisuology*. Verlag der Buchhandlung Walther Könic.

Daniels, D. (1997). “Art and Media in the XX. Century” en Joachimides Christos. *The Age of Modernism, Art in the 20th Century*. Norman Rosenthal yBrooks Adams. Gerd Hatje Verlag, Ostfildern. 553 – 564.

Devlin, Keith (2002). *El Lenguaje de las Matemáticas*, Editorial Ma Non Troppo.

DeWitt, T. (1987). “Visual music: Searching for an aesthetic” en *Leonardo*, vol. 20, p. 115-122.

Dixon, E. (2003). *Analysis of Musical Content in Digital Audio*. Austrian Research Institute for Artificial Intelligence.

Eisenstein, Sergei (1942). *The Film Sense*. New York: Harcourt, Brace & World.

Eisenstein, Sergei (1947). *The Film Form*. New York: Harcourt, Brace & Jovanovich.

Eisenstein, Sergei (2002). *Teoría y Técnica cinematográfica*. Madrid: Editorial Rialp.

Eckel, B. (2002). *Piensa en Java*. Editorial Prentice Hall.

Evans, Brian (2005). “Foundations of Visual Music” en *Computer Music Journal*, vol. 29, issue 4, p. 11-24.

Foote J. (1999). “Visualizing Music and Audio using Self-Similarity” en *Proceedings of ACM Multimedia '99*. ACM Press, 77-80.

Franssen, M. (1991). “The ocular harpsichord of Louis-Bertrand Castel. The science and aesthetics of an eighteenth-century cause célèbre” en *Tractrix. Yearbook for the History of Science, Medicine, Technology and Mathematics*, vol.3, p. 15-77.

Fry, Ben (2007). *Visualizing Data*. O`Really Media.

Furlong, Lucinda (1983). “Notes Toward a History of Image-Processed Video” en *Afterimage*, vol 11 p. 35-36.

Fuster, J. (2014). *Cerebro y Libertad. Los cimientos cerebrales de nuestra capacidad para elegir*. Editorial Planeta.

Galeyev, B.M. (1988). “The Fire of Prometheus Music-Kinetic Art Experiments in the USSR” en *Leonardo*, vol. 21, issue 4, p. 383-396.

Gardner, Martin (1970). “Mathematical games. The fantastic combinations of John Conway´s new solitaire game life” en *Scientific American*, vol. 223, p. 120-123.

Henke, Robert (2010). “Piezo”, en el Catálogo de la Exposición *Arte Sonoro*. La casa Encendida.

Hertz, Paul (1999). “Synesthetic Art, An Imaginary Number?” en Leonardo, vol. 32, num. 5.

Hill, Chris (1995). *Rewind. A Guide to Survey in the First Decade: Video Art and Alternative Media in the U.S., 1968-1980*. Ed. Video Data Bank.

Holtzman Stephen R. (1994). *Digital Mantras*. Cambridge, Massachusetts: The MIT Press.

Jacobson, I. et al (1992). *Object Oriented Software Engineering. A Use Case Driven Approach*. Addison Wesley.

Kaper, H. G., Wiebel, E. y Tipei, S. (1999). “Data Sonification and Sound Visualization” en *Computing in Science and Engineering*, vol. 1, issue 4, p. 48-58.

Kandinsky, Wassily (1996). *De lo espiritual en el arte*. Editorial Paidós.

Klein, A. B. (1927). *Colour-Music: The Art of Light*. London: Crosby Lockwood and Son.

Kock, W. (1971). *Seeing Sound*. New York: Wiley-Interscience.

Koening, W., Dunn, H. K. y Lacey, L. (1946). "The sound Spectrograph" en *Journal of Acoustical Society of America*. vol. 18, p.19-49.

Lage, Wedin (1972). "A Multidimensional Study of Perceptual-Emotional Qualities in Music" en *Scandinavian Journal of Psychology*, vol. 13, issue 1, p 241-257.

Langton, Christopher G. (1986). "Studying artificial life with cellular automata." Physica D: Nonlinear Phenomena 22(1-3): 120-149.

Lemi, E., Georgaki, A. (2007). "Reviewing the transformation of sound to image in new computer music software" en *Proceedings 4th Sound and Music Computing Conference*.

Levin, Golan (2000). *Painterly Interfaces for Audiovisual Performance*. M.S. Thesis. MIT Media Laboratory.

Lima, Manuel (2011). *Visual complexity: zapping patterns of information*. New York; Enfield: Princeton Architectural Press..

Lye, Len (1984). *Figures of Motion: Len Lye Selected Writings*. Edited Auckland University Press, p. 31–32.

Maeda, John (1999). *Design by Numbers*. Cambridge, Massachusetts: MIT Press.

Malinowski, S. *The Music Animation Machine.* <http://www.musanim.com/> [consulta: 10 julio 2013].

Masataka, Goto (2001). “An audio-based real-time beat tracking system for music with or without drum-sounds” en *Journal of New Music Research,* vol. 30, issue 2, p.159–171.

Mathews, M.V. (1969). *The Technology of Computer Music*. MIT Press, Cambridge, Massachusetts.

Martin Sanchez, G. (2010). *La música y la evolución de la narración audiovisual: la narración audiovisual en los videos musicales*. Editorial Abecdario.

Martinez, Julia (2011). *Kandinski y la abstracción: nuevas interpretaciones*. Tesis Doctoral. Departamento de Filosofía, Lógica y Estética. Universidad de Salamanca.

McDonnell, Maura (2007). “Visual Music” en *Visual Music Marathon Program Catalog*.

Moritz, M. (1996). “Mary Ellen Bute: Seeing Sound” en *Animation World,* vol. 1, issue 2, p. 49-62.

Moritz, William (1997). “The Dream of Color Music, And Machines That Made it Possible” en *Animation World Magazine*, vol. 2.

Negroponte, Nicholas (1995). *El mundo digital*. Barcelona: Ediciones B.

Ox, J. y Keefer, C. (2006). “On Curating Recent Digital Abstract Visual Music” en *New York Digital Salon`s Abstract Visual Music Project*.

Peacock, K. (1988). “Instruments to Perform Color-Music: Two Centuries of Technological Experimentation” en *Leonardo,* vol. 21, issue 3.

Pajares, G., et al (2001). *Visión por computador. Imágenes digitales y aplicaciones*. Editorial Ra-Ma..

Pajares, G., et al (2003). *Imágenes Digitales. Procesamiento práctico con Java*. Editorial Ra-Ma..

Palmer, S. E. et al. (2013). “Music-color associations are mediated by emotion” en *Proceedings of the National Academy of Sciences*.

Pérez Ornia, J.R. (1991). *El arte del video. Introducción a la historia del video experimental*. Ediciones del Serval.

Pfitzer, Gary (1991). “Music & Motion” en *Computer Graphics World*, p. 68-74.

Phoenix, Perry (Curator) (2002). Reline a video artist DVD compilation [DVD]. New York: Form Records.

Potter, P. et al (1947). *Visible Speech*. New York: Van Nostrand.

Pressman, R.S. (2001). *Ingeniería del Software: Un enfoque práctico*. Mc.Graw-Hill.

Puckette, M. (1991). “Something Digital” en *Computer Music Journal,* vol. 15, issue 4.

Quin, Iam (2015). “Extensible Markup Language (XML)” [texto on-line] <http://www.w3.org/XML/> [consulta: 10 julio 2013]

Ritter, Don (1993). “Interactive Video as a Way of Life” en *Musicworks*, vol 56, pp. 48-54.

Ritter, Don (2013). “A Flexible Approach for Synchronizing Video With Live Music” in *Leonardo Electronic Almanac*.

Restany, P. (1982). *Yves Klein*. New York: Harry A. Abrams.

Roads, C. (1996). *The Computer Music Tutorial*. MIT Press.

Rodriguez, Angel (1998). *La dimensión sonora del lenguaje audiovisual*. Barcelona: Editorial Paidós..

Roederer, J. (1997). *Acústica y Psicoacústica de la música*. Buenos Aires. Ricordi Americana S.A.E.C.

Rosseti, Laura (2011). *Videoarte: Del cine experimental al arte total*. México DF: UAM-X, CSH, Depto. de Educación y Comunicación.

Rumbaugh, J., et al (1995). *Modelado y diseño orientado a objetos*. Madrid. Prentice Hall, D.L.

Russett, Robert y Starr, Cecile (1988). *Experimental Animation: Origins of a New Art*. New York: Da Capo Press.

Sapp C. S. (2001). “Harmonic Visualizations of tonal Music” en *Proceedings of the 2001 International Computer Music Conference*. San Francisco, California: International Computer Music Association. p. 423-430.

Sánchez, J. A. et al (2009). *Historia, estética e iconografía del videoclip musical*. Málaga: Universidad de Málaga.

Sánchez Garreta, J. S. et al (2003). *Ingeniería de proyectos informáticos: actividades y procedimientos*. Universidad Jaume I, Servicio de Comunicación y Publicaciones.

Seashore, C. E. (1938). *Psychology of Music*. McGraw-Hill.

Sedes A., Courribet B. y Thiébaut J.B. (2004). "From the visualization of sound to real-time sonification: different prototypes in the Max/MSP/Jitter environment" en *Proceedings ICMC*.

Selva Ruiz, David (2012). "La visualización de la música en el videoclip" en Ámbitos, vol. 21-A, p. 101-115.

Scheirer, Eric D. (2000). *Music-Listening Systems*. Tesis Doctoral. Massachusetts Institute of Technology.

Sedes, A., Courribet, B., Thiebaut, J. (2004). "Visualization of Sound as a Control Interface" en Proc. of the 7th Int. Conference on Digital Audio Effects (DAFX-04), p. 1-6

Silleras Aguilar, Rocío (2015). *Sólido y Sonido: posibilidades creativas de la conjunción del sonido con los medios en estado sólido en la escultura sonora contemporánea*. Tesis doctoral. Universidad Politécnica de Valencia.

Slawson, Wayne (1989). "Sound Color" en *Music Perception: An Interdisciplinary Journal*, vol. 6, issue. 3, pp. 329-356.

Smalley, D. (1997). "Spectromorphology: Explaining Sound-Shapes" en Organised Sound. vol. 2, issue 2, p. 107–26.

Smith S. y Williams G. (1997). "A Visualization of Music" en Proceedings of Visualization, p. 499-502.

Spiegel, Laurie (1998). “Graphical Groove: Memorial for the Vampire, a Visual Music System” en *Organised Sound*, vol. 3, issue 3, p. 187-191.

Stober, S. y Nürnberger, A. (2013). “Adaptive music retrieval–a state of the art” en *Multimed Tools Appl,* vol. 65, p. 467–494.

Terhardt E. (1978). “Psychoacoustic evaluation of musical sounds” en Perception & Psychophysics, vol. 23, p. 483-492.

Tez (curator) (2009). *Various artists: optofonica.* [DVD]. Optofonica y Line.

Tzanetakis, G. y Cook, P. (2000). “*MARSYAS:* A framework for audio analysis” en *Organised Sound,* vol.4, issue 3.

Virilio, Paul (1989). *La máquina de la visión*. Editorial cátedra.

Wallace Rimington A. (1912). *Color Music: the art of Mobile Color*. London: Hutchinson & Company.

Walters, John L. (1997). “Sound, Code, Image” en *Eye,* vol. 26, p. 24-35.

Wattenberg, Martin (2002). “Arc Diagrams: Visualizing Structure in Strings” en *IEEE Symposium on Information Visualization*.

Wheeler, W. (2002). *Experimental Cinema: The Film Reader*. London: Wheeler Winston Dixon and Gwendolyn Audrey Foster.

Whitney, John (1980). *Digital harmony: On the Complementarity of Music and Visual Art*. McGraw-Hill.

Whitney, J. (1994). “To Paint on Water: The Audiovisual Duet of Complementarity” en *Computer Music Journal*, vol 18, issue 3, pp.44–52.

Yeo W. S., Berger J. y Lee Z. (2004). *SonART: A framework for data sonification, visualization and networked multimedia applications*. Ann Arbor, Michigan: MPublishing, University of Michigan Library.

Yeo W. S. y Berger J. (2006). *Raster Scanning: A New Approach to Image Sonification, Sound Visualization, Sound Analysis and Synthesis*. Ann Arbor, Michigan: MPublishing, University of Michigan Library.

Youngblood, Gene (1970). *Expanded Cinema.* New Cork. E. D Dutton and Co.

Zelkovitz, V., Shaw, A. y Gannon, J. (1979). *Principles of Software Engineering and Design*. Prentice Halls.

Zwicker, E. y Fastl, H.(1999). *Psychoacoustics: Facts and Models*. Springer.

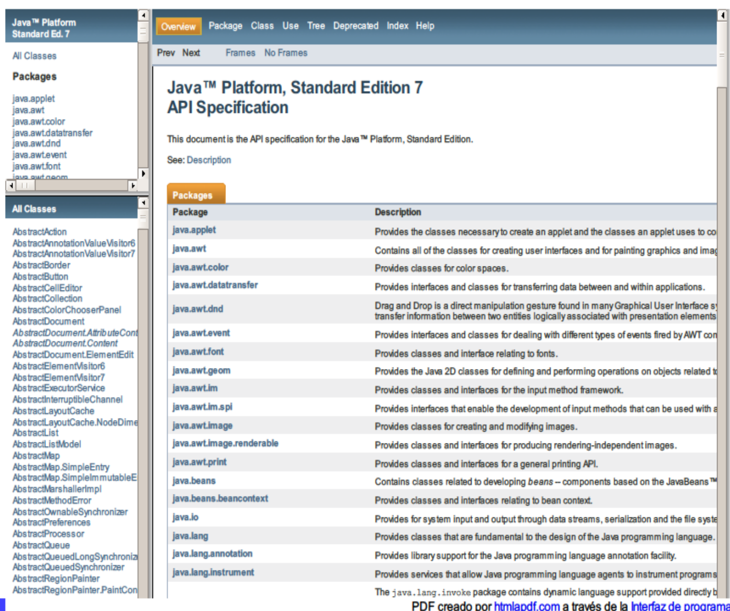
Anexo1. Documentación de Ciclope.

En este anexo vamos a presentar la documentación detallada por clases de la aplicación, que unido al contenido del capítulo 5 completan la documentación de la aplicación desarrollada.

El lenguaje de programación Java tiene prevista una herramienta para generación automática de documentación, que se denomina Javadoc, y es un estándar de la industria para documentar clases de Java. El esquema de documentación que propone es muy sencillo, conforme se va desarrollando la parte de código el programador a través de un lenguaje de etiquetas va añadiendo comentarios a las clases que va diseñando y a cada uno de sus métodos, sin preocuparse por la apariencia final que tendrá la documentación. Posteriormente a través de un pequeño programa que viene con la instalación del lenguaje Java, *javadoc*, se generará automáticamente un sitio web bien estructurado para la consulta de la documentación.

Para presentar la documentación en esta memoria lo que hemos realizado ha sido una conversión entre formatos, pasando de html a pdf y luego presentando los ficheros pdf en este documento. En la figura A1.1, vemos la página principal de la documentación de la librería estándar de *Java Standard Edition 7*. El diseño del sitio web está dividido en 3 marcos, siendo el que se encuentra a la derecha el principal y los dos de la izquierda secundarios para realizar las búsquedas por los paquetes y las clases.

Figura A1.1. Página principal de la documentación *de Java.*



En lo que resta de anexo, para presentar la documentación de las librerías y clases de Cíclope, vamos a realizar un aplanamiento del sitio web generado automáticamente mediante la herramienta *javadoc.* Únicamente mostraremos el desglose de clases junto con sus métodos obtenidos del marco principal que genera la herramienta. El proceso que hemos seguido para realizar el aplanamiento de la documentación ha sido transformar las diferentes páginas web que están en formato HTML[138], a formato PDF[139] para finalmente insertarlas en el texto. Para la presentación agruparemos las clases por librerías, y las presentaremos por orden alfabético.

[138] HTML, son las siglas de *Hyper Text Markup Languaje*, es un lenguaje basado en etiquetas para el diseño de páginas web.
[139] PDF, son las siglas de *Portable Document Format*, es un formato para almacenamiento de documentos digitales.

A.1.1. Librería de sonido

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

sonido

Class BufferLinea

java.lang.Object
    sonido.BufferLinea

```
public class BufferLinea
extends java.lang.Object
```

Esta clase representa un almacén de sonido simple, un buffer para una línea de sonido de entrada. Adicionalmente se etiqueta temporalmente dicho buffer para saber cuando se ha producido

Constructor Summary

Constructors

Constructor and Description
`BufferLinea(byte[] muestrasSonido, long etiqueta)` Constructor para la clase Sonido Buffer
`BufferLinea(java.nio.ByteBuffer muestrasSonido, long etiqueta)` Constructor para la clase Sonido Buffer.
`BufferLinea(float[] muestrasSonido, long etiqueta)` Constructor para la clase Sonido Buffer.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`esByte()` Método para comprobar si el metodo es de bytes o floats
`byte[]`	`getBufferByte()` Método para recuperar el vector de muestras
`java.nio.ByteBuffer`	`getBufferByteBuffer()` Método para recuperar el vector de muestras
`float[]`	`getBufferFloat()` Método para recuperar el vector de muestras
`long`	`getEtiquetaTemporal()` Método para recuperar la etiqueta temporal del vector de muestras

void	setBufferByte(byte[] muestrasSonido) Método para modificar el vector de muestras
void	setBufferByte(java.nio.ByteBuffer muestrasSonido) Método para modificar el vector de muestras
void	setBufferFloat(float[] muestrasSonido) Método para modificar el vector de muestras
void	setEtiquetaTemporal(long etiqueta) Método para modificar la etiqueta temporal del vector de muestras

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

BufferLinea

```
public BufferLinea(float[] muestrasSonido,
                   long etiqueta)
```

Constructor para la clase Sonido Buffer.

Parameters:

`muestras` - es el vector de muestras recogido del driver.

`etiqueta` - es la etiqueta temporal que identifica dicho buffer.

BufferLinea

```
public BufferLinea(java.nio.ByteBuffer muestrasSonido,
                   long etiqueta)
```

Constructor para la clase Sonido Buffer.

Parameters:

`muestras` - es el vector de muestras recogido del driver

`etiqueta` - es la etiqueta temporal que identifica dicho buffer.

BufferLinea

```
public BufferLinea(byte[] muestrasSonido,
                   long etiqueta)
```

Constructor para la clase Sonido Buffer

Parameters:

`muestras` - es el vector de muestras recogido del driver

etiqueta - es la etiqueta temporal que identifica dicho buffer

Method Detail

getBufferFloat

```
public float[] getBufferFloat()
```

Método para recuperar el vector de muestras

Returns:

vector de floats que representan las muestras.

getBufferByte

```
public byte[] getBufferByte()
```

Método para recuperar el vector de muestras

Returns:

vector de bytes que representan las muestras.

getBufferByteBuffer

```
public java.nio.ByteBuffer getBufferByteBuffer()
```

Método para recuperar el vector de muestras

Returns:

vector de bytes que representan las muestras.

getEtiquetaTemporal

```
public long getEtiquetaTemporal()
```

Método para recuperar la etiqueta temporal del vector de muestras

Returns:

la etiqueta temporal

setBufferFloat

```
public void setBufferFloat(float[] muestrasSonido)
```

Método para modificar el vector de muestras

Parameters:

muestrasSonido - un vector de floats que representan las muestras.

setBufferByte

```
public void setBufferByte(byte[] muestrasSonido)
```

Método para modificar el vector de muestras

Parameters:

`muestrasSonido` - un vector de bytes que representan las muestras.

setBufferByte

```
public void setBufferByte(java.nio.ByteBuffer muestrasSonido)
```

Método para modificar el vector de muestras

Parameters:

`muestrasSonido` - un objeto de la clase ByteBuffer

setEtiquetaTemporal

```
public void setEtiquetaTemporal(long etiqueta)
```

Método para modificar la etiqueta temporal del vector de muestras

Parameters:

`etiqueta` - la etiqueta temporal

esByte

```
public boolean esByte()
```

Método para comprobar si el metodo es de bytes o floats

Returns:

si el buffer es de bytes o no

sonido

Class ConfiguracionSonido

java.lang.Object
  sonido.ConfiguracionSonido

```
public class ConfiguracionSonido
extends java.lang.Object
```

Esta clase define los elementos y datos generales para la utilización del sonido en la aplicación, sus elementos son: - Sistema de sonido - driver asio - tamaño del buffer que manejará la aplicación, solo puede ser 1024 o 4096 - Número de buffers que vamos a mantener por línea - frecuencia de muestreo que solo podrá ser 44100 o 96000. - Array de valores lógicos indicando las líneas que están activas. - Vector de las líneas activas con todos sus elementos. Este es el objeto principal del paquete de sonido puesto que contendrá los buffers que vayamos capturando, que será usados posteriormente para realizar la visualización.

Constructor Summary

Constructors

Constructor and Description
`ConfiguracionSonido(AsioDriver driverAsio, int tamBuffer, int numBuffer, double frecuenciaMuestreo, boolean[] lineasActivas)` Constructor principal para la configuración general del sonido, cuando el sistema de sonido es ASIO.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`double`	`getFrecuencia()` Método para recuperar la frecuencia de muestreo actual
`LineaEntrada`	`getLineaEntrada(int indice)` Método para recuperar la línea de entrada a través de su índice
`boolean[]`	`getLineasActivas()` Método para recuperar el vector de líneas activas
`LineaEntrada[]`	`getLineasEntrada()` Método para recuperar el vector de canales de entrada
`int`	`getNumBuffer()`

	Método para recuperar el número de buffer
int	getNumLineasActivas() Método para recuperar el número de entradas activas
java.lang.String	getSistemaSonido() Método para recuperar el sistema de sonido
int	getTamBuffer() Método para recuperar el tamaño del buffer en bytes
AsioDriver	getTarjeta() Método para recuperar el driver asio actual
void	setFrecuencia(double muestreo) Método para modificar la frecuencia de muestreo actual
void	setSonidoEntrada(LineaEntrada linea, int posicion) Método para añadir una línea de entrad, en una determinada posición
void	setTamBuffer(int buffer) Método para modificar el tamaño del buffer en bytes
void	setTarjeta(AsioDriver asio) Método para modificar el driver de la tarjeta de sonido

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ConfiguracionSonido

```
public ConfiguracionSonido(AsioDriver driverAsio,
                           int tamBuffer,
                           int numBuffer,
                           double frecuenciaMuestreo,
                           boolean[] lineasActivas)
```

Constructor principal para la configuración general del sonido, cuando el sistema de sonido es ASIO. Este constructor es invocado por la configuración XML y tiene como parámetros todos los datos que obtenemos de dicho fichero, menos el driverAsio que se lo pedimos al usuario

Parameters:

driverAsio - el driver asio

tamBuffer - es el tamaño del buffer en muestras

numBuffer - es el número máximo de buffers que vamos a manejar

frecuenciaMuestreo - es la frecuencia de muestreo

lineasActivas - es un vector que me indica para cada una de las líneas de entrada

si están o no activas

Method Detail

getLineaEntrada

```
public LineaEntrada getLineaEntrada(int indice)
```

Método para recuperar la línea de entrada a través de su índice

Returns:

la línea de entrada

getFrecuencia

```
public double getFrecuencia()
```

Método para recuperar la frecuencia de muestreo actual

Returns:

el frecuencia de muestreo actual

getSistemaSonido

```
public java.lang.String getSistemaSonido()
```

Método para recuperar el sistema de sonido

Returns:

el sistema de sonido

getNumBuffer

```
public int getNumBuffer()
```

Método para recuperar el número de buffer

Returns:

el número de buffer

getTamBuffer

```
public int getTamBuffer()
```

Método para recuperar el tamaño del buffer en bytes

Returns:

el tamaño de buffer actual

getNumLineasActivas

```
public int getNumLineasActivas()
```

Método para recuperar el número de entradas activas

Returns:

`el número de líneas de entrada`

getLineasActivas

```
public boolean[] getLineasActivas()
```

Método para recuperar el vector de líneas activas

Returns:

`el vector de líneas activas`

getLineasEntrada

```
public LineaEntrada[] getLineasEntrada()
```

Método para recuperar el vector de canales de entrada

Returns:

`el vector de canales de entrada`

setFrecuencia

```
public void setFrecuencia(double muestreo)
```

Método para modificar la frecuencia de muestreo actual

Parameters:

`la - frecuencia de muestreo actual`

setTamBuffer

```
public void setTamBuffer(int buffer)
```

Método para modificar el tamaño del buffer en bytes

Parameters:

`el - tamaño de buffer actual`

setSonidoEntrada

```
public void setSonidoEntrada(LineaEntrada linea,
```

```
int posicion)
```

Método para añadir una línea de entrad, en una determinada posición

Parameters:

`línea` - representa una línea de entrada

`posicion` - de la línea de entrada

getTarjeta

```
public AsioDriver getTarjeta()
```

Método para recuperar el driver asio actual

Returns:

la tajeta de sonido asio actualmente seleccionada

setTarjeta

```
public void setTarjeta(AsioDriver asio)
```

Método para modificar el driver de la tarjeta de sonido

Parameters:

`asio` - el driver

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

sonido

Class ConfiguracionSonidoXML

java.lang.Object
    sonido.ConfiguracionSonidoXML

```
public class ConfiguracionSonidoXML
extends java.lang.Object
```

Esta clase representa el acceso al fichero XML con el fin de leer sus datos y escribirlos. El atributo principal es el fichero XML que debe manipular. Un ejemplo de dicho fichero es el siguiente:

< ?xml version="1.0" encoding="UTF-8"?>
<configuracionSonido>
<driver tipo=asio>OPTA-CAPTURE</driver>
<buffer>1024</buffer>
<numBuffer>4</numBuffer>
<frecuenciaMuestreo>96000</frecuenciaMuestreo>
<lineaEntrada>1</lineaEntrada>
<lineaEntrada>2</lineaEntrada>
<lineaEntrada>3</lineaEntrada>
<lineaEntrada>4</lineaEntrada>
<lineaEntrada>5</lineaEntrada>
<lineaEntrada>6</lineaEntrada>
</configuracionSonido>

Constructor Summary

Constructors

Constructor and Description
`ConfiguracionSonidoXML(java.io.File fich)` Constructor para la clase ConfiguracionSonidoXML unicamente obtenemos el nombre del fichero XML
`ConfiguracionSonidoXML(java.lang.String nomFich)` Segundo constructor para la clase SecuenciacionXML únicamente obtenemos el nombre del fichero XML

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`escribirXML(ConfiguracionSonido sonidoConfiguracion)`

	Este método dada una configuración de sonido construye un fichero XML y lo guarda en memoria <?
`java.io.File`	`getFicheroXML()` Este método recupera el fichero de secuenciación XML
`ConfiguracionSonido`	`leerXML()` Este método lee el contenido del fichero de configuración de sonido XML generando un objeto de la clase ConfiguracionSonido
`void`	`setFicheroXML(java.io.File ficheroXML)` Este método para modificar el fichero de secuenciación XML

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ConfiguracionSonidoXML

```
public ConfiguracionSonidoXML(java.io.File fich)
```

Constructor para la clase ConfiguracionSonidoXML unicamente obtenemos el nombre del fichero XML

Parameters:

`fich` - el fichero xml de secuenciacion

ConfiguracionSonidoXML

```
public ConfiguracionSonidoXML(java.lang.String nomFich)
```

Segundo constructor para la clase SecuenciacionXML únicamente obtenemos el nombre del fichero XML

Parameters:

`nomFich` - el fichero xml de secuenciacion

Method Detail

getFicheroXML

```
public java.io.File getFicheroXML()
```

Este método recupera el fichero de secuenciación XML

Returns:

`ficheroXML el fichero xml`

setFicheroXML

```
public void setFicheroXML(java.io.File ficheroXML)
```

Este método para modificar el fichero de secuenciación XML

Parameters:

`ficheroXML` - el fichero xml

leerXML

```
public ConfiguracionSonido leerXML()
```

Este método lee el contenido del fichero de configuración de sonido XML generando un objeto de la clase ConfiguracionSonido

Returns:

`configuracionSonidoXML la nueva configuración de sonido`

escribirXML

```
public void escribirXML(ConfiguracionSonido sonidoConfiguracion)
```

Este método dada una configuración de sonido construye un fichero XML y lo guarda en memoria

```
<?xml version="1.0" encoding="UTF-8"?>
<configuracionSonido>
<driver tipo=asio>OPTA-CAPTURE </driver>
<buffer> 1024 </buffer>
<numBuffer> 4 </numBuffer>
<frecuenciaMuestreo> 96000 </frecuenciaMuestreo>
<lineaEntrada> 1 </lineaEntrada>
<lineaEntrada> 2 </lineaEntrada>
</configuracionSonido>
```

Parameters:

`sonidoConfiguracion` - la configuración de sonido

sonido

Class LineaEntrada

java.lang.Object
    sonido.LineaEntrada

All Implemented Interfaces:

AsioDriverListener

```
public class LineaEntrada
extends java.lang.Object
implements AsioDriverListener
```

Esta clase representa una línea de sonido donde hemos ido almacenando todas las muestras que llegan por dicha línea, el atributo principal es el LinkedList de buffers que respresentan las muestras. Todas las muestras estan entre -1 y 1.

Constructor Summary

Constructors

Constructor and Description
`LineaEntrada(java.lang.String identificador, int index, int limite, int tamBuffer)` Constructor para la línea de entrada

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`bufferSizeChanged(int bufferSize)` The driver has a new preferred buffer size.
`void`	`bufferSwitch(long tiempoMuestra, long posicionMuestra, java.util.Set<AsioChannel> listaLineas)` Método principal para realizar la captura de las muestras de sonido, que vienen de la tarjeta de sonido.
`BufferLinea`	`getBufferActual()` Método para recuperar el buffer actual
`BufferLinea`	`getBufferIndex(int index)` Método para recuperar el buffer que indica el índice
`java.lang.String`	`getIdentificador()` Método para recuperar el identificador
`int`	`getIndex()`

	Método para recuperar el número de la línea
java.util.LinkedList<BufferLinea>	getListaBuffer() Método para recuperar el vector de buffers
void	latenciesChanged(int inputLatency, int outputLatency) The input or output latencies have changed.
void	resetRequest() The driver requests a reset in the case of an unexpected failure or a device reconfiguration.
void	resyncRequest() The driver detected audio buffer underruns and requires a resynchronization.
void	sampleRateDidChange(double sampleRate) The sample rate has changed.
void	setBufferActual(BufferLinea almacen) Método para añadir un buffer al vector
void	setIdentificador(java.lang.String id) Método para modificar el identificador de la linea de entrada
void	setListaBuffer(java.util.LinkedList<BufferLinea> almacen) Método para modificar la lista de buffers
void	setSonidoBuffer(BufferLinea almacen) Método para añadir un buffer al vector

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LineaEntrada

```
public LineaEntrada(java.lang.String identificador,
                    int index,
                    int limite,
                    int tamBuffer)
```

Constructor para la línea de entrada

Parameters:

identificador - es el identificador de la línea

acotada - si la línea de entrada tiene acotados el numeros de buffers que se guardan

límite - en caso de estar acotado el numero de buffers históricos

Method Detail

bufferSwitch

```
public void bufferSwitch(long tiempoMuestra,
                         long posicionMuestra,
                         java.util.Set<AsioChannel> listaLineas)
```

Método principal para realizar la captura de las muestras de sonido, que vienen de la tarjeta de sonido. Este método lo invoca el driver en función del tamaño del buffer que hayamos colocado en la configuración. Aunque este método tiene como parámetro la lista entera de todas las líneas de sonido. Sólo recuperaremos el buffer de la que se corresponde con este objetto

Specified by:

`bufferSwitch` in interface `AsioDriverListener`

Parameters:

`tiempoMuestra` - el tiempo de la priemra línea del la lista de las muestras

`posicionMuestra` - la posición que desde inicio de la captura ocupa el primer elemento de cada lista de las líneas

`listaLineas` - es la lista de líneas de sonido que estan activas en la tarjeta de sonido

sampleRateDidChange

```
public void sampleRateDidChange(double sampleRate)
```

Description copied from interface: `AsioDriverListener`

The sample rate has changed. This may be due to a user initiated change, or a change in input/output source.

Specified by:

`sampleRateDidChange` in interface `AsioDriverListener`

Parameters:

`sampleRate` - The new sample rate.

resetRequest

```
public void resetRequest()
```

Description copied from interface: `AsioDriverListener`

The driver requests a reset in the case of an unexpected failure or a device reconfiguration. As this request is being made in a callback, the driver should only be reset after this callback method has returned. The recommended way to reset the driver is:

```
 public void resetRequest() {
   new Thread() {
```

Specified by:

`resetRequest` in interface `AsioDriverListener`

resyncRequest

```
public void resyncRequest()
```

Description copied from interface: AsioDriverListener
The driver detected audio buffer underruns and requires a resynchronization.

Specified by:
resyncRequest in interface AsioDriverListener

bufferSizeChanged

```
public void bufferSizeChanged(int bufferSize)
```

Description copied from interface: AsioDriverListener
The driver has a new preferred buffer size. The host should make an effort to accommodate the driver by returning to the INITIALIZED state and calling AsioDriver.createBuffers().

Specified by:
bufferSizeChanged in interface AsioDriverListener

Parameters:
bufferSize - The new preferred buffer size.

latenciesChanged

```
public void latenciesChanged(int inputLatency,
                             int outputLatency)
```

Description copied from interface: AsioDriverListener
The input or output latencies have changed. The host is updated with the new values.

Specified by:
latenciesChanged in interface AsioDriverListener

Parameters:
inputLatency - The new input latency in milliseconds.

outputLatency - The new output latency in milliseconds.

getListaBuffer

```
public java.util.LinkedList<BufferLinea> getListaBuffer()
```

Método para recuperar el vector de buffers

Returns:
el vector de buffers

setListaBuffer

```
public void setListaBuffer(java.util.LinkedList<BufferLinea> almacen)
```

Método para modificar la lista de buffers

Parameters:
almacen - el vector de buffers

setSonidoBuffer

```
public void setSonidoBuffer(BufferLinea almacen)
```

Método para añadir un buffer al vector

Parameters:
`almacen` - el buffer

getIdentificador

```
public java.lang.String getIdentificador()
```

Método para recuperar el identificador

Returns:
el identificador de la linea

getIndex

```
public int getIndex()
```

Método para recuperar el número de la línea

Returns:
el identificador de la linea

getBufferIndex

```
public BufferLinea getBufferIndex(int index)
```

Método para recuperar el buffer que indica el índice

Parameters:
`index` - el indice del buffer

Returns:
el buffer

getBufferActual

```
public BufferLinea getBufferActual()
```

Método para recuperar el buffer actual

Returns:
el buffer

setIdentificador

```
public void setIdentificador(java.lang.String id)
```

Método para modificar el identificador de la linea de entrada

Parameters:

id - el identificador

setBufferActual

```
public void setBufferActual(BufferLinea almacen)
```

Método para añadir un buffer al vector

Parameters:

almacen - el buffer

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

sonido

Class SonidoException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        sonido.SonidoException

All Implemented Interfaces:

`java.io.Serializable`

```
public class SonidoException
extends java.lang.RuntimeException
```

SonidoException, es una excepción que se lanza cuando algo no funciona bien con el sistema de sonido

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
`SonidoException(java.lang.String mensaje)`

Method Summary

Methods inherited from class java.lang.Throwable

`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

SonidoException

```
public SonidoException(java.lang.String mensaje)
```

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

A.1.2. Librería de interfaz de usuario

interfazUsuario

Class ConfiguracionIU

java.lang.Object
    interfazUsuario.ConfiguracionIU

```
public class ConfiguracionIU
extends java.lang.Object
```

En esta clase tendremos todos los elementos princiaples de la igu, que serán accesibles vía atributos estáticos para el resto de la aplicación.

Field Summary

Fields

Modifier and Type	Field and Description
IUPrincipal	iguPrincipal
IULienzo	lienzo
IUMarcoLienzo	marcoLienzo

Constructor Summary

Constructors

Constructor and Description
ConfiguracionIU(IUPrincipal iguPrincipal) Constructor de la clase le pasamos como argumento la igu principal, es el único elemento que gestionamos desde aquí.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
IUPrincipal	getIguPrincipal() Metodo para recuperar el marco principal de la aplicación
void	setIguPrincipal(IUPrincipal marco) Metodo para modificar el marco principal de la aplicación.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

iguPrincipal

```
public IUPrincipal iguPrincipal
```

marcoLienzo

```
public IUMarcoLienzo marcoLienzo
```

lienzo

```
public IULienzo lienzo
```

Constructor Detail

ConfiguracionIU

```
public ConfiguracionIU(IUPrincipal iguPrincipal)
```

Constructor de la clase le pasamos como argumento la igu principal, es el único elemento que gestionamos desde aquí.

Parameters:

`iguPrincipal` - la interfaz gráfica principal.

Method Detail

getIguPrincipal

```
public IUPrincipal getIguPrincipal()
```

Metodo para recuperar el marco principal de la aplicación

Returns:

el JFrame principal

setIguPrincipal

```
public void setIguPrincipal(IUPrincipal marco)
```

Metodo para modificar el marco principal de la aplicación.

Parameters:

`marco - el marco principal.`

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

interfazUsuario

Class InterfazUsuarioException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        interfazUsuario.InterfazUsuarioException

All Implemented Interfaces:

`java.io.Serializable`

```
public class InterfazUsuarioException
extends java.lang.RuntimeException
```

Esta clase represeta una excepción que se lanza cuando algo no funciona bien en la intefaz de usuario.

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
`InterfazUsuarioException(java.lang.String mensaje)`

Method Summary

Methods inherited from class java.lang.Throwable
`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

InterfazUsuarioException

InterfazUsuarioException

```
public InterfazUsuarioException(java.lang.String mensaje)
```

interfazUsuario

Class IUBoton

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.AbstractButton
          javax.swing.JButton
            interfazUsuario.IUBoton

All Implemented Interfaces:

`java.awt.image.ImageObserver, java.awt.ItemSelectable, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.SwingConstants`

```
public class IUBoton
extends javax.swing.JButton
```

Esta clase define un botón con una determinada imagen que le pasamos en el constructor, las imágenes están almacenadas en la carpeta recursos de este mismo paquete. Es la clase que usamos para crear cada uno de los 6 botones de la IGU de la aplicación, todos son iguales salvo como es lógico su programación y su imagen.

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent

`javax.swing.JComponent.AccessibleJComponent`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.AbstractButton

`BORDER_PAINTED_CHANGED_PROPERTY, CONTENT_AREA_FILLED_CHANGED_PROPERTY, DISABLED_ICON_CHANGED_PROPERTY, DISABLED_SELECTED_ICON_CHANGED_PROPERTY, FOCUS_PAINTED_CHANGED_PROPERTY, HORIZONTAL_ALIGNMENT_CHANGED_PROPERTY, HORIZONTAL_TEXT_POSITION_CHANGED_PROPERTY, ICON_CHANGED_PROPERTY, MARGIN_CHANGED_PROPERTY, MNEMONIC_CHANGED_PROPERTY, MODEL_CHANGED_PROPERTY,`

PRESSED_ICON_CHANGED_PROPERTY, ROLLOVER_ENABLED_CHANGED_PROPERTY, ROLLOVER_ICON_CHANGED_PROPERTY, ROLLOVER_SELECTED_ICON_CHANGED_PROPERTY, SELECTED_ICON_CHANGED_PROPERTY, TEXT_CHANGED_PROPERTY, VERTICAL_ALIGNMENT_CHANGED_PROPERTY, VERTICAL_TEXT_POSITION_CHANGED_PROPERTY

Fields inherited from class javax.swing.JComponent

TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.SwingConstants

BOTTOM, CENTER, EAST, HORIZONTAL, LEADING, LEFT, NEXT, NORTH, NORTH_EAST, NORTH_WEST, PREVIOUS, RIGHT, SOUTH, SOUTH_EAST, SOUTH_WEST, TOP, TRAILING, VERTICAL, WEST

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Constructors

Constructor and Description
IUBoton(javax.swing.ImageIcon imagen, javax.swing.ImageIcon imagenP, java.awt.Rectangle bound) Constructor para la clase IGUBoton simplemente le pasamos la imagen que representa el botón, la posición y dimensiones que debe tener.

Method Summary

Methods inherited from class javax.swing.JButton

getAccessibleContext, getUIClassID, isDefaultButton, isDefaultCapable, removeNotify, setDefaultCapable, updateUI

Methods inherited from class javax.swing.AbstractButton

addActionListener, addChangeListener, addItemListener, doClick, doClick, getAction, getActionCommand, getActionListeners, getChangeListeners, getDisabledIcon, getDisabledSelectedIcon, getDisplayedMnemonicIndex, getHideActionText, getHorizontalAlignment, getHorizontalTextPosition, getIcon,

getIconTextGap, getItemListeners, getLabel, getMargin, getMnemonic, getModel, getMultiClickThreshhold, getPressedIcon, getRolloverIcon, getRolloverSelectedIcon, getSelectedIcon, getSelectedObjects, getText, getUI, getVerticalAlignment, getVerticalTextPosition, imageUpdate, isBorderPainted, isContentAreaFilled, isFocusPainted, isRolloverEnabled, isSelected, removeActionListener, removeChangeListener, removeItemListener, setAction, setActionCommand, setBorderPainted, setContentAreaFilled, setDisabledIcon, setDisabledSelectedIcon, setDisplayedMnemonicIndex, setEnabled, setFocusPainted, setHideActionText, setHorizontalAlignment, setHorizontalTextPosition, setIcon, setIconTextGap, setLabel, setLayout, setMargin, setMnemonic, setMnemonic, setModel, setMultiClickThreshhold, setPressedIcon, setRolloverEnabled, setRolloverIcon, setRolloverSelectedIcon, setSelected, setSelectedIcon, setText, setUI, setVerticalAlignment, setVerticalTextPosition

Methods inherited from class javax.swing.JComponent

addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, hide, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list,

locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

IUBoton

```
public IUBoton(javax.swing.ImageIcon imagen,
               javax.swing.ImageIcon imagenP,
               java.awt.Rectangle bound)
```

Constructor para la clase IGUBoton simplemente le pasamos la imagen que representa el botón, la posición y dimensiones que debe tener.

Parameters:

`imagen` - la imagen que representa el botón

`bound` - la posición y dimensiones dentro del contenedor al que pertenece

`imagenP` - que se ve cuando se pulse el botón

interfazUsuario

Class IULienzo

java.lang.Object
 java.awt.Component
  java.awt.Container
   javax.swing.JComponent
    javax.swing.JPanel
     interfazUsuario.IULienzo

All Implemented Interfaces:

`java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible`

```
public class IULienzo
extends javax.swing.JPanel
```

Esta clase representa un lienzo donde dibujaremos las imágenes generadas desde el sonido

See Also:

`Serialized Form`

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent

`javax.swing.JComponent.AccessibleJComponent`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.JComponent

`TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW`

Fields inherited from class java.awt.Component

`BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT`

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Constructors

Constructor and Description
IULienzo() Esta clase es el constructor del lienzo

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	paintComponent(java.awt.Graphics g) El método principal para realizar el dibujo de las diferentes imágenes

Methods inherited from class javax.swing.JPanel

getAccessibleContext, getUI, getUIClassID, setUI, updateUI

Methods inherited from class javax.swing.JComponent

addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, hide, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled,

setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

IULienzo

```
public IULienzo()
```

Esta clase es el constructor del lienzo

Method Detail

paintComponent

```
public void paintComponent(java.awt.Graphics g)
```

El método principal para realizar el dibujo de las diferentes imágenes

Overrides:

`paintComponent in class javax.swing.JComponent`

interfazUsuario

Class IUMarcoLienzo

java.lang.Object
 java.awt.Component
  java.awt.Container
   java.awt.Window
    java.awt.Dialog
     javax.swing.JDialog
      interfazUsuario.IUMarcoLienzo

All Implemented Interfaces:

`java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants`

```
public class IUMarcoLienzo
extends javax.swing.JDialog
```

Esta clase representa el lienzo en su versión visual, se trata de un JDialog que luego albergará el como tal que es una componente genérica JPanel. En cualquier caso aquí es importante tener en cuenta las dimensiones del lienzo, asi como su posición en x e y dentro del dispositivo de visualización

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class java.awt.Dialog

`java.awt.Dialog.ModalExclusionType, java.awt.Dialog.ModalityType`

Nested classes/interfaces inherited from class java.awt.Window

`java.awt.Window.Type`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class java.awt.Dialog

DEFAULT_MODALITY_TYPE

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Constructors

Constructor and Description
IUMarcoLienzo()

Method Summary

Methods inherited from class javax.swing.JDialog

getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setTransferHandler, update

Methods inherited from class java.awt.Dialog

addNotify, getModalityType, getTitle, hide, isModal, isResizable, isUndecorated, setBackground, setModal, setModalityType, setOpacity, setResizable, setShape, setTitle, setUndecorated, setVisible, show, toBack

Methods inherited from class java.awt.Window

addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBackground, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner, getOpacity, getOwnedWindows, getOwner, getOwnerlessWindows, getShape, getToolkit, getType, getWarningString,

getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isAutoRequestFocus, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isOpaque, isShowing, isValidateRoot, pack, paint, postEvent, removeNotify, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setAutoRequestFocus, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImage, setIconImages, setLocation, setLocation, setLocationByPlatform, setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize, setSize, setType, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, print, printComponents, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusTraversalKeys, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, dispatchEvent, enable, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBaseline, getBaselineResizeBehavior, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphicsConfiguration, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener,

removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, requestFocusInWindow, resize, resize, revalidate, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setMaximumSize, setName, setPreferredSize, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

IUMarcoLienzo

```
public IUMarcoLienzo()
```

interfazUsuario

Class IUPrincipal

java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
                        interfazUsuario.IUPrincipal

All Implemented Interfaces:

`java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants`

```
public class IUPrincipal
extends javax.swing.JFrame
```

Esta clase representa la Interfaz gráfica principal de la aplicación. Se trata de una especialización de la clase JFrame, que es el marco principal de cualquier aplicación. En esta clase están definidos todos los botones de la aplicación y además sus correspondientes oyentes, para su tratamiento.

See Also:

`Serialized Form`

Nested Class Summary

Nested classes/interfaces inherited from class java.awt.Window

`java.awt.Window.Type`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.JFrame

`EXIT_ON_CLOSE`

Fields inherited from class java.awt.Frame

CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Constructors

Constructor and Description
IUPrincipal() Este es el constructor principal de la clase y el único método, debido a que se muestran todos los elementos de la interfaz y se activan todos los oyentes de dichos elementos.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
javax.swing.JButton	getBotonConfigurar() Método para recuperar el botón de configuracion de la aplicación
javax.swing.JButton	getBotonCrear() Método para recuperar el botón de creación de la aplicación
javax.swing.JButton	getBotonEjecutar() Método para recuperar el botón de ejecución de la aplicación
javax.swing.JButton	getBotonParar() Método para recuperar el botón de parar de la aplicación
javax.swing.JButton	getBotonPausar() Método para recuperar el botón de pausar de la aplicación
javax.swing.JButton	getBotonPreparar() Método para recuperar el botón de preparación de la aplicación

java.lang.String	getDriverAsio() Metodo para recuperar el nombre del driver asio actual
javax.swing.JLabel	getEtiquetaDriver() Método para recuperar la etiqueta del combo box
javax.swing.JLabel	getEtiquetaErrores() Método para recuperar la etiqueta de errores
javax.swing.JLabel	getEtiquetaTiempo() Método para recuperar la etiqueta del tiempo
javax.swing.JComboBox	getListaDriver() Método para recuperar la etiqueta del tiempo
void	setDriverAsio(java.lang.String asio) método para modificar el nombre del driver de la tarjeta de sonido

Methods inherited from class javax.swing.JFrame

getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout, setTransferHandler, update

Methods inherited from class java.awt.Frame

addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setBackground, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setOpacity, setResizable, setShape, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window

addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBackground, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner, getOpacity, getOwnedWindows, getOwner, getOwnerlessWindows, getShape, getToolkit, getType, getWarningString, getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, hide, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isAutoRequestFocus, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isOpaque, isShowing, isValidateRoot, pack, paint, postEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setAutoRequestFocus, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImages, setLocation, setLocation, setLocationByPlatform, setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize, setSize,

setType, setVisible, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, print, printComponents, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusTraversalKeys, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, dispatchEvent, enable, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBaseline, getBaselineResizeBehavior, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphicsConfiguration, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, requestFocusInWindow, resize, resize, revalidate, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setMaximumSize, setName, setPreferredSize, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Methods inherited from interface java.awt.MenuContainer

`getFont, postEvent`

Constructor Detail

IUPrincipal

```
public IUPrincipal()
```

Este es el constructor principal de la clase y el único método, debido a que se muestran todos los elementos de la interfaz y se activan todos los oyentes de dichos elementos.

Method Detail

getBotonCrear

```
public javax.swing.JButton getBotonCrear()
```

Método para recuperar el botón de creación de la aplicación

Returns:

`ojo que es el botón`

getBotonConfigurar

```
public javax.swing.JButton getBotonConfigurar()
```

Método para recuperar el botón de configuracion de la aplicación

Returns:

`ojo que es el botón`

getBotonPreparar

```
public javax.swing.JButton getBotonPreparar()
```

Método para recuperar el botón de preparación de la aplicación

Returns:

`ok que es el botón`

getBotonEjecutar

```
public javax.swing.JButton getBotonEjecutar()
```

Método para recuperar el botón de ejecución de la aplicación

Returns:

`play que es el botón`

getBotonPausar

```
public javax.swing.JButton getBotonPausar()
```

Método para recuperar el botón de pausar de la aplicación

Returns:

`pause que es el botón`

getBotonParar

```
public javax.swing.JButton getBotonParar()
```

Método para recuperar el botón de parar de la aplicación

Returns:

`pause que es el botón`

getEtiquetaDriver

```
public javax.swing.JLabel getEtiquetaDriver()
```

Método para recuperar la etiqueta del combo box

Returns:

`textoComboBox que es el texto del combo box`

getEtiquetaTiempo

```
public javax.swing.JLabel getEtiquetaTiempo()
```

Método para recuperar la etiqueta del tiempo

Returns:

`tiempo que es la etiqueta del tiempo`

getEtiquetaErrores

```
public javax.swing.JLabel getEtiquetaErrores()
```

Método para recuperar la etiqueta de errores

Returns:

errores la etiqueta de errores

getListaDriver

```
public javax.swing.JComboBox getListaDriver()
```

Método para recuperar la etiqueta del tiempo

Returns:

tiempo que es la etiqueta del tiempo

getDriverAsio

```
public java.lang.String getDriverAsio()
```

Metodo para recuperar el nombre del driver asio actual

Returns:

el nombre del driver asio actualmente seleccionada

setDriverAsio

```
public void setDriverAsio(java.lang.String asio)
```

método para modificar el nombre del driver de la tarjeta de sonido

Parameters:

asio - el nombre del driver

A.1.3. Librería de imagen

imagen

Class ConfiguracionImagen

java.lang.Object
    imagen.ConfiguracionImagen

```
public class ConfiguracionImagen
extends java.lang.Object
```

Esta clase contiene todos los elementos de configuración relacionados con las imágenes. Los dos elementos mas importantes que tiene esta clase son: la lista de imágenes que vamos a gestionar en la visualización y el lienzo donde vamos a realizar dicha visualización.

Constructor Summary

Constructors

Constructor and Description
`ConfiguracionImagen(Lienzo lienzo, java.util.LinkedList<Imagen> listaSecuenciacionImagenes)` El constructor de la clase tendrá como parámetros el lienzo y la lista de las imágenes a secuenciar obtenidos del fichero configuracionImagen.xml.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`Lienzo`	`getLienzo()` Método recupera el lienzo donde veremos las imágenes.
`java.util.LinkedList<Imagen>`	`getListaImagenes()` Método recupera la lista de secuenciación de las imagenes.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ConfiguracionImagen

```
public ConfiguracionImagen(Lienzo lienzo,

java.util.LinkedList<Imagen> listaSecuenciacionImagenes)
```

El constructor de la clase tendrá como parámetros el lienzo y la lista de las imágenes a secuenciar obtenidos del fichero configuracionImagen.xml.

Parameters:

`lienzo` - el lienzo donde vamos a dibujar las imágenes.

`listaSecuenciacionImagenes` - la lista de imágenes que vamos a usar para la visualización.

Method Detail

getListaImagenes

```
public java.util.LinkedList<Imagen> getListaImagenes()
```

Método recupera la lista de secuenciación de las imagenes.

Returns:

la lista de secuenciacion de las imagenes

getLienzo

```
public Lienzo getLienzo()
```

Método recupera el lienzo donde veremos las imágenes.

Returns:

el lienzo de la secuenciacion

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

imagen

Class ConfiguracionImagenXML

java.lang.Object
    imagen.ConfiguracionImagenXML

```
public class ConfiguracionImagenXML
extends java.lang.Object
```

Esta clase representa el acceso al fichero XML, donde se encuentran los datos de configuración del lienzo y de las imágenes.

Constructor Summary

Constructors

Constructor and Description
`ConfiguracionImagenXML(java.io.File fich)` Constructor para la clase SecuenciacionXML.
`ConfiguracionImagenXML(java.lang.String nomFich)` Constructor para la clase SecuenciacionXML.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`escribirXML(ConfiguracionImagen secuenciacion)`
`java.io.File`	`getFicheroXML()` Método recuperar el fichero de secuenciacion XML.
`ConfiguracionImagen`	`leerXML()` Este método lee el contenido del fichero de secuenciación XML, generando la listas de imágenes correspondientes, donde están todos los elementos necesarios para realizar la secuenciación.
`void`	`setFicheroXML(java.io.File ficheroXML)` Método para modificar el fichero de secuenciación XML.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ConfiguracionImagenXML

```
public ConfiguracionImagenXML(java.io.File fich)
```

Constructor para la clase SecuenciacionXML.

Parameters:

`fich` - el fichero xml de secuenciación.

ConfiguracionImagenXML

```
public ConfiguracionImagenXML(java.lang.String nomFich)
```

Constructor para la clase SecuenciacionXML.

Parameters:

`nomFich` - el fichero xml de secuenciación.

Method Detail

leerXML

```
public ConfiguracionImagen leerXML()
```

Este método lee el contenido del fichero de secuenciación XML, generando la listas de imágenes correspondientes, donde están todos los elementos necesarios para realizar la secuenciación.

Returns:

la configuración principal de la imagen.

getFicheroXML

```
public java.io.File getFicheroXML()
```

Método recuperar el fichero de secuenciacion XML.

Returns:

ficheroXML el fichero xml.

setFicheroXML

```
public void setFicheroXML(java.io.File ficheroXML)
```

Método para modificar el fichero de secuenciación XML.

Parameters:

`ficheroXML` - el fichero de secuenciación xml.

escribirXML

```
public void escribirXML(ConfiguracionImagen secuenciacion)
```

imagen

Class Generar

java.lang.Object
  java.util.TimerTask
    imagen.Generar

All Implemented Interfaces:

`java.lang.Runnable`

```
public class Generar
extends java.util.TimerTask
```

Esta clase representa la tarea básica de renderización de las imágenes, lo que hace es ir accediendo a los elementos actuales de los nodos temporales de cada uan de las líneas de sonido y realizar la actualización de la imagen pixel por pixel, a través de los algoritmos de visualización.

Constructor Summary

Constructors

Constructor and Description
`Generar(Imagen imagen)` Constructor para la clase Generar, se el pasa la imagen obtenida del fichero xml .

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`ABID(boolean[] canalesColor, float muestra)`
void	`ABIDALEATORIO(boolean[] canalesColor, float muestra)`
void	`ALEATORIO(boolean[] canalesColor, float muestra)`
void	`ALEATORIOGRIS(boolean[] canalesColor, float muestra)`
void	`colorFondo(java.awt.Color color)` Este método recorrer toda la imagen y pone el color de fondo que le pasamos como parámetro.
void	`HORMIGA(boolean[] canalesColor, float muestra)`
void	`HORMIGAESTATICA(boolean[] canalesColor, float muestra)`
void	`IDAB(boolean[] canalesColor, float muestra)`

	el método IDAB, lo que hace es dada la posición actual del buffer que la tenemos en el atributo calcula segun el algoritmo en el que estemos su posición espacial
void	`iniciar()` Este método inicia la generación de la imagen.
void	`renderizarImagen()`
void	`run()` Este método representa la tarea principal de esta TimerTask que consiste en generar los pixeles de la imagen en funcion del algoritmo que este activo en cada momento.

Methods inherited from class java.util.TimerTask

`cancel, scheduledExecutionTime`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Generar

```
public Generar(Imagen imagen)
```

Constructor para la clase Generar, se el pasa la imagen obtenida del fichero xml .

Parameters:

`imagen` - `la imagen.`

Method Detail

iniciar

```
public void iniciar()
```

Este método inicia la generación de la imagen.

colorFondo

```
public void colorFondo(java.awt.Color color)
```

Este método recorrer toda la imagen y pone el color de fondo que le pasamos como parámetro. El modelo de color con el que trabajamos es float y el número máximo es el 1 y el mínimo el 0.

Parameters:

`color` - el color de fondo.

run

```
public void run()
```

Este método representa la tarea principal de esta TimerTask que consiste en generar los pixeles de la imagen en funcion del algoritmo que este activo en cada momento.

Specified by:

`run` in interface `java.lang.Runnable`

Specified by:

`run` in class `java.util.TimerTask`

renderizarImagen

```
public void renderizarImagen()
```

IDAB

```
public void IDAB(boolean[] canalesColor,
                 float muestra)
```

el método IDAB, lo que hace es dada la posición actual del buffer que la tenemos en el atributo calcula segun el algoritmo en el que estemos su posición espacial

ABID

```
public void ABID(boolean[] canalesColor,
                 float muestra)
```

ABIDALEATORIO

```
public void ABIDALEATORIO(boolean[] canalesColor,
                          float muestra)
```

ALEATORIO

```
public void ALEATORIO(boolean[] canalesColor,
                      float muestra)
```

ALEATORIOGRIS

```
public void ALEATORIOGRIS(boolean[] canalesColor,
                          float muestra)
```

HORMIGAESTATICA

```
public void HORMIGAESTATICA(boolean[] canalesColor,
                            float muestra)
```

HORMIGA

```
public void HORMIGA(boolean[] canalesColor,
                    float muestra)
```

imagen

Class Imagen

java.lang.Object
java.awt.Image
java.awt.image.BufferedImage
imagen.Imagen

All Implemented Interfaces:

java.awt.image.RenderedImage, java.awt.image.WritableRenderedImage, java.awt.Transparency

---

```
public class Imagen
extends java.awt.image.BufferedImage
```

Esta clase representa una imagen, es una extensión de la clase BufferedImage.

Field Summary

Fields inherited from class java.awt.image.BufferedImage

TYPE_3BYTE_BGR, TYPE_4BYTE_ABGR, TYPE_4BYTE_ABGR_PRE, TYPE_BYTE_BINARY, TYPE_BYTE_GRAY, TYPE_BYTE_INDEXED, TYPE_CUSTOM, TYPE_INT_ARGB, TYPE_INT_ARGB_PRE, TYPE_INT_BGR, TYPE_INT_RGB, TYPE_USHORT_555_RGB, TYPE_USHORT_565_RGB, TYPE_USHORT_GRAY

Fields inherited from class java.awt.Image

SCALE_AREA_AVERAGING, SCALE_DEFAULT, SCALE_FAST, SCALE_REPLICATE, SCALE_SMOOTH, UndefinedProperty

Fields inherited from interface java.awt.Transparency

BITMASK, OPAQUE, TRANSLUCENT

Constructor Summary

Constructors

Constructor and Description
Imagen(java.lang.String id, java.awt.Dimension dimension, java.awt.image.ComponentColorModel colorModel, java.awt.image.WritableRaster raster, boolean b) Primer constructor muy sencillo de la imagen simplemente le pasamos la dimensión de la imagen y un nombre
Imagen(java.lang.String id, java.awt.Dimension dimension, java.util.LinkedList<LineaSonido> listaLineas, TransformadasImagen transformada, java.awt.image.ComponentColorModel colorModel, java.awt.image.WritableRaster raster, boolean b) Constructor para la clase SecuenciacionSecuenciarImagen simplemente le pasamos la lista de

secuenciación obtenida del fichero xml para una imagen

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
float	getAlfa() Método para recuperar la constante por la que multiplicaremos el canal alfa de la imagen.
ControlAccesoImagen	getControlAcceso() Método para recuperar el control de acceso sincronizado de la imagen.
java.awt.Dimension	getDimension() Método para recuperar la dimension de la imagen.
Generar	getGenerar() Método para recuperar el objeto que se encarga de realizar la renderización de la imagen.
java.lang.String	getIdentificador() Método para recuperar el identificador de la imagen.
int	getInterpolacion() Método para recuperar el tipo de interpolación de la imagen.
java.util.LinkedList<LineaSonido>	getListaLineas() Método para recuperar la lista de líneas de la imagen.
boolean	getRender() Método para saber si la imagen está o no incluída en la cola de render .
Secuenciar	getSecuenciar() Método para recuperar el objeto que se encarga de gestionar la lista de acciones para el renderizado.
TransformadasImagen	getTransformada() Método para recuperar la transformada de la imagen.
void	setAlfa(float alfa) Método para modificar la constante por la que multiplicaremos el canal alfa de la imagen.
void	setControlAcceso(ControlAccesoImagen accesoImagen) Método para cambiar el control de acceso sincronizado de la imagen.
void	setDimension(java.awt.Dimension dim) Método para modificar la dimension de la imagen.
void	setIdentificador(java.lang.String id) Método para modificar el identificador de la imagen.
void	setInterpolacion(int interpolacion) Método para modificar el tipo de interpolación de la imagen.
void	setLinea(LineaSonido linea)
void	setListaLineas(java.util.LinkedList<LineaSonido> listaLineas)

	Método para modificar la lista de líneas de la imagen.
void	setRender(boolean render) Método para modificar la incluisón de la imagen o no en la cola de render .
void	setTransformada(TransformadasImagen transformada) Método para modificar las transformadas básicas de la imagen.

Methods inherited from class java.awt.image.BufferedImage

addTileObserver, coerceData, copyData, createGraphics, getAlphaRaster, getColorModel, getData, getData, getGraphics, getHeight, getHeight, getMinTileX, getMinTileY, getMinX, getMinY, getNumXTiles, getNumYTiles, getProperty, getProperty, getPropertyNames, getRaster, getRGB, getRGB, getSampleModel, getSource, getSources, getSubimage, getTile, getTileGridXOffset, getTileGridYOffset, getTileHeight, getTileWidth, getTransparency, getType, getWidth, getWidth, getWritableTile, getWritableTileIndices, hasTileWriters, isAlphaPremultiplied, isTileWritable, releaseWritableTile, removeTileObserver, setData, setRGB, setRGB, toString

Methods inherited from class java.awt.Image

flush, getAccelerationPriority, getCapabilities, getScaledInstance, setAccelerationPriority

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Imagen

```
public Imagen(java.lang.String id,
              java.awt.Dimension dimension,
              java.awt.image.ComponentColorModel colorModel,
              java.awt.image.WritableRaster raster,
              boolean b)
```

Primer constructor muy sencillo de la imagen simplemente le pasamos la dimensión de la imagen y un nombre

Parameters:

nombre - identificador de la imagen

dimension - tiene el ancho y el alto de la imagen

colorModel - el modelo de color

raster - el raster de la imagen

b - si el canal alfa esta premultiplicado o no

Imagen

```
public Imagen(java.lang.String id,
              java.awt.Dimension dimension,
              java.util.LinkedList<LineaSonido> listaLineas,
```

```
               TransformadasImagen transformada,
               java.awt.image.ComponentColorModel colorModel,
               java.awt.image.WritableRaster raster,
               boolean b)
```

Constructor para la clase SecuenciacionSecuenciarImagen simplemente le pasamos la lista de secuenciación obtenida del fichero xml para una imagen

Parameters:

`id - la idedentificacion de la imagen`

`ancho - el ancho de la imagen`

`alto - el alto de la imagen`

`la - lista de secuenciación de líneas de una imagen`

Method Detail

getListaLineas

```
public java.util.LinkedList<LineaSonido> getListaLineas()
```

Método para recuperar la lista de líneas de la imagen.

Returns:

`la lista de Lineas;`

getIdentificador

```
public java.lang.String getIdentificador()
```

Método para recuperar el identificador de la imagen.

Returns:

`el identificador de la imagen.`

getTransformada

```
public TransformadasImagen getTransformada()
```

Método para recuperar la transformada de la imagen.

Returns:

`la transformada de la imagen.`

getInterpolacion

```
public int getInterpolacion()
```

Método para recuperar el tipo de interpolación de la imagen. El 1 equivale a una interpolacion dura, VALUE_STROKE_PURE de la clase RenderingHints. El 2 equivale a una interpolacion dura, VALUE_INTERPOLATION_BILINEAR de la clase RenderingHints.

Returns:

`la interpolacion de la imagen.`

getAlfa

```
public float getAlfa()
```

Método para recuperar la constante por la que multiplicaremos el canal alfa de la imagen.

getControlAcceso

```
public ControlAccesoImagen getControlAcceso()
```

Método para recuperar el control de acceso sincronizado de la imagen.

Returns:
`el control de acceso de la imagen.`

getDimension

```
public java.awt.Dimension getDimension()
```

Método para recuperar la dimension de la imagen.

Returns:
`la dimension de la imagen.`

getGenerar

```
public Generar getGenerar()
```

Método para recuperar el objeto que se encarga de realizar la renderización de la imagen.

Returns:
`objeto de renderización.`

getSecuenciar

```
public Secuenciar getSecuenciar()
```

Método para recuperar el objeto que se encarga de gestionar la lista de acciones para el renderizado.

Returns:
`objeto de secuenciación.`

getRender

```
public boolean getRender()
```

Método para saber si la imagen está o no incluída en la cola de render .

Returns:
`si renderizamos la imagen o no.`

setListaLineas

```
public void setListaLineas(java.util.LinkedList<LineaSonido> listaLineas)
```

Método para modificar la lista de líneas de la imagen.

Parameters:

`la` - lista de líneas.

setIdentificador

```
public void setIdentificador(java.lang.String id)
```

Método para modificar el identificador de la imagen.

Parameters:

`id` - el nuevo identificador de la imagen.

setDimension

```
public void setDimension(java.awt.Dimension dim)
```

Método para modificar la dimension de la imagen.

Parameters:

`dim` - la nueva dimensión de la imagen.

setTransformada

```
public void setTransformada(TransformadasImagen transformada)
```

Método para modificar las transformadas básicas de la imagen.

Parameters:

`transformada` - las nuevas transformadas de la imagen.

setLinea

```
public void setLinea(LineaSonido linea)
```

setInterpolacion

```
public void setInterpolacion(int interpolacion)
```

Método para modificar el tipo de interpolación de la imagen. El 1 equivale a una interpolacion dura, VALUE_STROKE_PURE de la clase RenderingHints. El 2 equivale a una interpolacion suave, VALUE_INTERPOLATION_BILINEAR de la clase RenderingHints.

Parameters:

`interpolacion` - la interpolacion de la imagen.

setRender

```
public void setRender(boolean render)
```

Método para modificar la incluisón de la imagen o no en la cola de render .

Parameters:

`render` - si renderizamos la imagen o no.

setAlfa

```
public void setAlfa(float alfa)
```

Método para modificar la constante por la que multiplicaremos el canal alfa de la imagen.

Parameters:

`alfa` - el factor para el canal alfa.

setControlAcceso

```
public void setControlAcceso(ControlAccesoImagen accesoImagen)
```

Método para cambiar el control de acceso sincronizado de la imagen.

Parameters:

`accesoImagen` - el control de acceso de la imagen.

imagen

Class Lienzo

java.lang.Object
  imagen.Lienzo

```
public class Lienzo
extends java.lang.Object
```

Esta clase representa el lienzo básico que vamos a usar para la visualización, pero a este nivel es simplemente un concepto que encapsula la dimensión del dispositivo de visualización.

Constructor Summary

Constructors

Constructor and Description
`Lienzo(java.awt.Dimension dim, java.lang.String identificador)` Constructor para la clase Lienzo, le pasamos como parámetros un objeto de la clase Dimension.
`Lienzo(int width, int height, java.lang.String identificador)` Constructor para la clase Lienzo, le pasamos como parámetros las dimensiones del lienzo.
`Lienzo(java.awt.geom.Point2D.Float posicion, java.awt.Dimension dim, java.lang.String identificador)`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`java.awt.Dimension`	`getDimLienzo()` Método para obtener la dimensión del lienzo.
`java.lang.String`	`getIdentificador()` Este método recupera el identificador del lienzo.
`java.awt.geom.Point2D.Float`	`getPosicion()` Método para obtener la posición del lienzo en el dispositivo de visualización.
`void`	`setDimLienzo(int width, int height)` Método para modificar la dimensión del lienzo.

void	setPosicion(int x, int y) Método para modificar la posición del lienzo.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Lienzo

```
public Lienzo(int width,
              int height,
              java.lang.String identificador)
```

Constructor para la clase Lienzo, le pasamos como parámetros las dimensiones del lienzo.

Parameters:

width - ancho del lienzo.

height - alto del lienzo.

identificador - identificador del lienzo.

Lienzo

```
public Lienzo(java.awt.Dimension dim,
              java.lang.String identificador)
```

Constructor para la clase Lienzo, le pasamos como parámetros un objeto de la clase Dimension.

Parameters:

dim - dimension del lienzo

identificador - identificador del lienzo.

Lienzo

```
public Lienzo(java.awt.geom.Point2D.Float posicion,
              java.awt.Dimension dim,
              java.lang.String identificador)
```

Method Detail

getIdentificador

```
public java.lang.String getIdentificador()
```

Este método recupera el identificador del lienzo.

Returns:

`identificador del lienzo`

getDimLienzo

```
public java.awt.Dimension getDimLienzo()
```

Método para obtener la dimensión del lienzo.

Returns:

`dimensión del lienzo.`

setDimLienzo

```
public void setDimLienzo(int width,
                         int height)
```

Método para modificar la dimensión del lienzo.

Parameters:

`width` - ancho del lienzo.

`height` - alto del lienzo.

getPosicion

```
public java.awt.geom.Point2D.Float getPosicion()
```

Método para obtener la posición del lienzo en el dispositivo de visualización.

Returns:

`posición del lienzo.`

setPosicion

```
public void setPosicion(int x,
                        int y)
```

Método para modificar la posición del lienzo.

Parameters:

`x` - posición x del lienzo.

`y` - posición y del lienzo.

imagen

Class LineaSonidoGenerar

java.lang.Object
    imagen.LineaSonidoGenerar

```
public class LineaSonidoGenerar
extends java.lang.Object
```

Esta clase representa una extensión de una linea de sonido en la que se han incluido las variaciones que se van a producir a lo largo del tiempo de su visualización. Su cometido es gestionar la lista de nodos temporales, de forma que en cada momento se mantenga información actualizada de su estado.

Constructor Summary

Constructors

Constructor and Description
LineaSonidoGenerar(java.util.LinkedList<NodoTemporal> listaNodos, int indiceLinea, int numBuffer) Constructor para la clase LineaSonidoGenerar.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
boolean	getFinSecuenciacion() Método para preguntar si hemos llegado al final de la secuenciación.
int	getIndiceLinea() Método para obtener el identificador de la línea de sonido de esta secuenciación.
LineaEntrada	getlineaEntrada() Método para obtener la referencia a la línea de entrada donde están las muestras.
java.util.LinkedList<NodoTemporal>	getListaNodos() Método para obtener la lista de nodos de la línea de sonido.
NodoTemporal	getNodoActual() Método para obtener el nodo actual de la secuenciación de línea de sonido.
int	getNumBuffer() Método para obtener el número de buffer de la línea de sonido.
java.util.ListIterator	getPosicionActual() Método para obtener la posición actual de la secuenciación de línea

	de sonido.
void	**setFinSecuenciacion**(boolean fin) Método para colocar la secuenciacion en estado de finalización o inicio.
void	**setIndiceLinea**(int identificadorLinea) Método para modificar el identificador de la línea de sonido.
void	**setListaNodos**(java.util.LinkedList<NodoTemporal> listaNodos) Método para modificar la lista de los nodos de la secuenciación.
void	**setNodoActual**(NodoTemporal actual) Método para modificar el nodo actual de la secuenciación.
void	**setNumBuffer**(int numBuffer) Método para modificar el número de buffer de la línea.
void	**setPosicionActual**(java.util.ListIterator posicionActual) Método para modificar la posición actual de la secuenciación.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LineaSonidoGenerar

```
public LineaSonidoGenerar(java.util.LinkedList<NodoTemporal> listaNodos,
                          int indiceLinea,
                          int numBuffer)
```

Constructor para la clase LineaSonidoGenerar. Los elementos que le pasamos como parámetros los hemos obtenido del xml de secuenciación.

Parameters:

listaNodos - lista de nodos lo que le pasamos es la lista de nodos.

indiceLinea - identificador de la línea de sonido.

numBuffer - el número de buffer de la línea de sonido.

Method Detail

getFinSecuenciacion

```
public boolean getFinSecuenciacion()
```

Método para preguntar si hemos llegado al final de la secuenciación.

Returns:

un valor lógico que indica si hemos llegado al final la lista.

getNumBuffer

```
public int getNumBuffer()
```

Método para obtener el número de buffer de la línea de sonido.

Returns:
el número de buffer.

getIndiceLinea

```
public int getIndiceLinea()
```

Método para obtener el identificador de la línea de sonido de esta secuenciación.

Returns:
el identificador de la línea de sonido.

getListaNodos

```
public java.util.LinkedList<NodoTemporal> getListaNodos()
```

Método para obtener la lista de nodos de la línea de sonido.

Returns:
la lista de nodos.

getNodoActual

```
public NodoTemporal getNodoActual()
```

Método para obtener el nodo actual de la secuenciación de línea de sonido.

Returns:
el nodo actual.

getPosicionActual

```
public java.util.ListIterator getPosicionActual()
```

Método para obtener la posición actual de la secuenciación de línea de sonido.

Returns:
la posición actual.

getlineaEntrada

```
public LineaEntrada getlineaEntrada()
```

Método para obtener la referencia a la línea de entrada donde están las muestras.

Returns:
la línea de entrada.

setFinSecuenciacion

```
public void setFinSecuenciacion(boolean fin)
```

Método para colocar la secuenciacion en estado de finalización o inicio.

Parameters:

`fin` - el nuevo estado.

setNumBuffer

```
public void setNumBuffer(int numBuffer)
```

Método para modificar el número de buffer de la línea.

Parameters:

`numBuffer` - el número de buffer.

setIndiceLinea

```
public void setIndiceLinea(int identificadorLinea)
```

Método para modificar el identificador de la línea de sonido.

Parameters:

`identificadorLinea` - el identificador de la línea de sonido.

setListaNodos

```
public void setListaNodos(java.util.LinkedList<NodoTemporal> listaNodos)
```

Método para modificar la lista de los nodos de la secuenciación.

Parameters:

`listaNodos` - la lista de los nodos.

setNodoActual

```
public void setNodoActual(NodoTemporal actual)
```

Método para modificar el nodo actual de la secuenciación.

Parameters:

`actual` - el nodo actual.

setPosicionActual

```
public void setPosicionActual(java.util.ListIterator posicionActual)
```

Método para modificar la posición actual de la secuenciación.

Parameters:

`posicionActual` - la posición actual.

LineaSonidoGenerar

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

imagen

Class NodoTemporal

java.lang.Object
    imagen.NodoTemporal

```
public class NodoTemporal
extends java.lang.Object
```

Esta clase representa un elemento de la secuenciación, dada una línea de sonido y un buffer de dicha línea, cada elemento de la secuenciación tendrá como datos: el inicio en milisengundos de la secuencia, la duración, el algoritmo que se debe usar para visualizar, el número de buffer y el número de canales a los que afecta. Toda esta información es lo que representa esta clase.

Constructor Summary

Constructors

Constructor and Description
`NodoTemporal(long inicio, long duracion, int algoritmo, int metodoF, boolean[] canales)` Constructor para la clase SecuenciaNodo, le damos los valores a los atributos de la clase que serán los elementos de una línea de secuenciación.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`int`	`getAlgoritmo()` Método para recuperar el algoritmo del nodo de secuenciación.
`boolean[]`	`getCanales()` Método para recuperar los canales de color que afectan al nodo de secuenciación.
`long`	`getDuracion()` Método para recuperar la duración de tiempo en milisengundos del nodo de secuenciación.
`long`	`getInicio()` Método para recuperar el instante de tiempo en milisengundos del nodo de secuenciación.
`int`	`getMetodoFusion()`

	Método para recuperar el método de fusión empleado en el nodo de secuenciación.
void	setAlgoritmo(int algoritmo) Método para modificar el algoritmo del nodo de secuenciación.
void	setCanales(boolean[] canales) Método para modificar los canales de color del nodo de secuenciación.
void	setDuracion(long duracion) Método para modificar la duración del nodo de secuenciación.
void	setInicio(long inicio) Método para modificar el instante de tiempo de inicio en milisengundos del nodo de secuenciación.
void	setMetodoFusion(int metodoF) Método para modificar el método de fusión del nodo de secuenciación.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

NodoTemporal

```
public NodoTemporal(long inicio,
                    long duracion,
                    int algoritmo,
                    int metodoF,
                    boolean[] canales)
```

Constructor para la clase SecuenciaNodo, le damos los valores a los atributos de la clase que serán los elementos de una línea de secuenciación.

Parameters:

inicio - es el instante de inicio en milisegundos de dicho nodo.

duracion - es la duración en milisegundos del nodo.

algoritmo - es el tipo del algoritmo que tendremos que usar.

canales - es un vector de 4 elementos que indica a que canales afecta el nodo.

Method Detail

getInicio

```
public long getInicio()
```

Método para recuperar el instante de tiempo en milisengundos del nodo de secuenciación.

Returns:

`el valor de inicio.`

getDuracion

```
public long getDuracion()
```

Método para recuperar la duración de tiempo en milisengundos del nodo de secuenciación.

Returns:

`la duración.`

getAlgoritmo

```
public int getAlgoritmo()
```

Método para recuperar el algoritmo del nodo de secuenciación.

Returns:

`el algoritmo.`

getCanales

```
public boolean[] getCanales()
```

Método para recuperar los canales de color que afectan al nodo de secuenciación.

Returns:

`los canales de color.`

getMetodoFusion

```
public int getMetodoFusion()
```

Método para recuperar el método de fusión empleado en el nodo de secuenciación.

Returns:

`el metodo de fusión.`

setInicio

```
public void setInicio(long inicio)
```

Método para modificar el instante de tiempo de inicio en milisengundos del nodo de secuenciación.

Parameters:

`inicio` - el nuevo valor de inicio.

setDuracion

```
public void setDuracion(long duracion)
```

Método para modificar la duración del nodo de secuenciación.

Parameters:
`duracion` - el nuevo valor de duración.

setAlgoritmo

```
public void setAlgoritmo(int algoritmo)
```

Método para modificar el algoritmo del nodo de secuenciación.

Parameters:
`algoritmo` - el nuevo algoritmo.

setCanales

```
public void setCanales(boolean[] canales)
```

Método para modificar los canales de color del nodo de secuenciación.

Parameters:
`canales` - los nuevos canales.

setMetodoFusion

```
public void setMetodoFusion(int metodoF)
```

Método para modificar el método de fusión del nodo de secuenciación.

Parameters:
`metodoF` - el nuevo método de fusión.

imagen

Class Secuenciar

java.lang.Object
  java.util.TimerTask
    imagen.Secuenciar

All Implemented Interfaces:

java.lang.Runnable

```
public class Secuenciar
extends java.util.TimerTask
```

Esta clase representa la tarea básica de la secuenciación de las imágenes. Lo que hace es ir actualizando los nodos temporales conforme va pasando el tiempo, para que la tarea de renderización tenga los valores correctos al ir a generar las imágenes.

Constructor Summary

Constructors

Constructor and Description
`Secuenciar(java.util.LinkedList<LineaSonidoGenerar> listaLineas)` Constructor para la clase Secuenciar, se le pasa la lista de secuenciación obtenida del fichero xml.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`iniciar()` Este método inicia la secuenciación, lo que hace es recorrer la lista de líneas y colocar los elementos distinguidos en la primera posición, preparando la lista para realizar el recorrido.
void	`run()` Este método representa la tarea principal de esta TimerTask, que consiste en ir recorriendo las líneas de la lista de secuenciacion de la imagen y comprobar si hay cambios.

Methods inherited from class java.util.TimerTask

`cancel, scheduledExecutionTime`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Secuenciar

```
public Secuenciar(java.util.LinkedList<LineaSonidoGenerar> listaLineas)
```

Constructor para la clase Secuenciar, se le pasa la lista de secuenciación obtenida del fichero xml.

Parameters:

`listalineas` - la lista de secuenciación de líneas de una imagen

Method Detail

iniciar

```
public void iniciar()
             throws SecuenciarException
```

Este método inicia la secuenciación, lo que hace es recorrer la lista de líneas y colocar los elementos distinguidos en la primera posición, preparando la lista para realizar el recorrido.

Throws:

`SecuenciarException`

run

```
public void run()
```

Este método representa la tarea principal de esta TimerTask, que consiste en ir recorriendo las líneas de la lista de secuenciacion de la imagen y comprobar si hay cambios.

Specified by:

`run` in interface `java.lang.Runnable`

Specified by:

`run` in class `java.util.TimerTask`

imagen

Class SecuenciarException

java.lang.Object
 java.lang.Throwable
 java.lang.Exception
 java.lang.RuntimeException
 imagen.SecuenciarException

All Implemented Interfaces:

`java.io.Serializable`

```
public class SecuenciarException
extends java.lang.RuntimeException
```

Esta clase es una excepción, que se lanza cuando algo no funciona bien con la secuenciación.

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
`SecuenciarException(java.lang.String mensaje)`

Method Summary

Methods inherited from class java.lang.Throwable

`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

SecuenciarException

```
public SecuenciarException(java.lang.String mensaje)
```

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

imagen

Class TransformadasImagen

java.lang.Object
imagen.TransformadasImagen

```
public class TransformadasImagen
extends java.lang.Object
```

/** Esta clase representa una transformada bidimiensional de la imagen. Punto de anclaje de la imagen, posición dentro del lienzo, escala y rotación con respecto al punto de anclaje.

Constructor Summary

Constructors

Constructor and Description
`TransformadasImagen()` Constructor básico que crea una transformada sin producir ninguna alteración.
`TransformadasImagen(java.awt.geom.Point2D.Double puntoAnclaje, java.awt.geom.Point2D.Double posicion, java.awt.Dimension escala, int rotacion)` Constructor para la clase TransformadasImagen, crea un elemento de dicha clase, una transformada 2D.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`java.awt.Dimension`	`getEscala()` Método para recuperar la escala.
`java.awt.geom.Point2D.Double`	`getPosicion()` Método para recuperar la posición.
`java.awt.geom.Point2D.Double`	`getPuntoAnclaje()` Método para recuperar el punto de anclaje.
`int`	`getRotacion()` Método para recuperar la rotación.
`void`	`setEscala(java.awt.Dimension escala)` Método para modificar la escala.
`void`	`setPosicion(java.awt.geom.Point2D.Double posicion)`

	Método para modificar la posición.
void	`setPuntoAnclaje(java.awt.geom.Point2D.Double puntoAnclaje)` Método para modificar el punto de anclaje.
void	`setRotacion(int rotacion)` Método para modificar la rotación.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

TransformadasImagen

```
public TransformadasImagen(java.awt.geom.Point2D.Double puntoAnclaje,
                           java.awt.geom.Point2D.Double posicion,
                           java.awt.Dimension escala,
                           int rotacion)
```

Constructor para la clase TransformadasImagen, crea un elemento de dicha clase, una transformada 2D.

Parameters:

`puntoAnclaje` - punto anclaje

`posicion` - posición de la imagen

`escala` - porcentaje de x e y en la escala

`rotacion` - con respecto al punto de anclaje

TransformadasImagen

```
public TransformadasImagen()
```

Constructor básico que crea una transformada sin producir ninguna alteración.

Method Detail

getPuntoAnclaje

```
public java.awt.geom.Point2D.Double getPuntoAnclaje()
```

Método para recuperar el punto de anclaje.

Returns:

`la posición del punto de anclaje.`

getPosicion

```
public java.awt.geom.Point2D.Double getPosicion()
```

Método para recuperar la posición.

Returns:
`la posición de la imagen.`

getEscala

```
public java.awt.Dimension getEscala()
```

Método para recuperar la escala.

Returns:
`la escala de la imagen.`

getRotacion

```
public int getRotacion()
```

Método para recuperar la rotación.

Returns:
`la rotación de la imagen.`

setPuntoAnclaje

```
public void setPuntoAnclaje(java.awt.geom.Point2D.Double puntoAnclaje)
```

Método para modificar el punto de anclaje.

Parameters:
`puntoAnclaje` - `el punto de anclaje.`

setPosicion

```
public void setPosicion(java.awt.geom.Point2D.Double posicion)
```

Método para modificar la posición.

Parameters:
`posicion` - `la posición.`

setEscala

```
public void setEscala(java.awt.Dimension escala)
```

Método para modificar la escala.

Parameters:
`escala` - la escala de la imagen.

setRotacion

```
public void setRotacion(int rotacion)
```

Método para modificar la rotación.

Parameters:
`rotacion` - la rotación de la imagen.

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

A.1.4. Librería de controlAplicacion

controlAplicacion

Class ConfiguracionControl

java.lang.Object
  controlAplicacion.ConfiguracionControl

```
public class ConfiguracionControl
extends java.lang.Object
```

Esta clase define los elementos necesarios de configuración generales del de la aplicación.

Constructor Summary

Constructors

Constructor and Description
ConfiguracionControl()

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method and Description
static ConfiguracionIU	getConfiguracionIGU() Método para recuperar la igu de la aplicación.
static ConfiguracionImagen	getConfiguracionImagen() Método para recuperar la configuración de la imagen.
static ConfiguracionSonido	getConfiguracionSonido() Método para recuperar el driver asio.
static Reloj	getContador() Método para recuperar el contador de la aplicación
static ControlProcesos	getControlProcesos() Método para recuperar el control de procesos.
static void	setConfiguracionIGU(ConfiguracionIU igu) Método para modificar la igu de la aplicación.
static void	setConfiguracionImagen(ConfiguracionImagen a) Método para cambiar la configuración de la imagen.
static void	setConfiguracionSonido(ConfiguracionSonido a)

	Método para modificar la configuración de sonido.
static void	`setContador(Reloj cont)` Método para modificar el contador de la aplicación.
static void	`setControlProcesos(ControlProcesos control)` Método para modificar el control de procesos.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ConfiguracionControl

```
public ConfiguracionControl()
```

Method Detail

getControlProcesos

```
public static ControlProcesos getControlProcesos()
```

Método para recuperar el control de procesos.

Returns:

`el control de procesos de la aplicación.`

setControlProcesos

```
public static void setControlProcesos(ControlProcesos control)
```

Método para modificar el control de procesos.

Parameters:

`control - el control de procesos de la aplicación.`

setContador

```
public static void setContador(Reloj cont)
```

Método para modificar el contador de la aplicación.

Parameters:

`cont - contador de la aplicación.`

getContador

`public static Reloj getContador()`

Método para recuperar el contador de la aplicación

Returns:

`el contador`

getConfiguracionIGU

`public static ConfiguracionIU getConfiguracionIGU()`

Método para recuperar la igu de la aplicación.

Returns:

`iguPrincipal, la igu de la aplicación.`

setConfiguracionIGU

`public static void setConfiguracionIGU(ConfiguracionIU igu)`

Método para modificar la igu de la aplicación.

Parameters:

`igu` - `la igu de la aplicación.`

getConfiguracionSonido

`public static ConfiguracionSonido getConfiguracionSonido()`

Método para recuperar el driver asio.

Returns:

`el driver asio actual.`

getConfiguracionImagen

`public static ConfiguracionImagen getConfiguracionImagen()`

Método para recuperar la configuración de la imagen.

Returns:

`la configuración de la imagen.`

setConfiguracionSonido

`public static void setConfiguracionSonido(ConfiguracionSonido a)`

Método para modificar la configuración de sonido.

Parameters:

`la - configuración de sonido.`

setConfiguracionImagen

```
public static void setConfiguracionImagen(ConfiguracionImagen a)
```

Método para cambiar la configuración de la imagen.

Parameters:

`a - la nueva configuración de la audio imagen`

controlAplicacion

Interface Control

All Known Implementing Classes:

ControlCapturar, ControlContar, ControlGenerar, ControlIGU, ControlPintar, ControlProcesos, ControlSecuenciar

---

```
public interface Control
```

Esta interfaz representa la tarea básica de control de la aplicación, contiene los métodos básicos para realizar los cambios de estado. Todos los controles de los procesos de la aplicación deben implementar esta interfaz.

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
void	configurar() Método para pasar del estado INICIADO al estado CONFIGURADO.
void	crear() Método para pasar del estado INICIAL al estado INICIADO.
void	ejecutar() Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN
void	finalizar() Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL
void	parar() Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO
void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO

Method Detail

crear

```
void crear()
```

Método para pasar del estado INICIAL al estado INICIADO.

configurar

```
void configurar()
```

Método para pasar del estado INICIADO al estado CONFIGURADO.

preparar

```
void preparar()
```

Método para pasar del estado CONFIGURADO al estado PREPARADO

ejecutar

```
void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN

pausar

```
void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

parar

```
void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

finalizar

```
void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

controlAplicacion

Class ControlAccesoImagen

java.lang.Object
  controlAplicacion.ControlAccesoImagen

```
public class ControlAccesoImagen
extends java.lang.Object
```

Esta clase representa el mecanismo de sincronización de acceso a una imagen. Se plantean dos modalidades: el acceso para modificarlas a través de las clases y del modulo de y el acceso para realizar la lectura con la clase del módulo de . El mecanismo de sincronización se basa con ligeras modificaciones en el clásico de lectores y escritores. En nuestro caso solamente tendremos dos escritores que son el hilo activo de la generación de la imagen y el de secuenciación y el lector que será el hilo principal de renderización.

Constructor Summary

Constructors

Constructor and Description
`ControlAccesoImagen()`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`ControlAccesoImagen()` Constructor para la clase de ControlAccesoImagen, básicamente lo que realiza es dar los valores iniciales de los parámetros internos.
void	`finEscritura()` Método para finalizar la modificación de la imagen.
void	`finLectura()` Método para finalizar el acceso a la imagen para lectura.
void	`inicioEscritura()` Método para iniciar el acceso a la imagen para modificarla.
void	`inicioLectura()` Método para iniciar el acceso a la imagen para lectura, esto es para la renderización de la imagen.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ControlAccesoImagen

```
public ControlAccesoImagen()
```

Method Detail

ControlAccesoImagen

```
public void ControlAccesoImagen()
```

Constructor para la clase de ControlAccesoImagen, básicamente lo que realiza es dar los valores iniciales de los parámetros internos.

inicioLectura

```
public void inicioLectura()
```

Método para iniciar el acceso a la imagen para lectura, esto es para la renderización de la imagen. Este hilo se debe bloquear cuando tengamos un hilo generando la imagen. Una vez desbloqueado accede a la imagen y detiene a cualquier otro proceso de escritura hasta que termine. En el problema de sincronización de lectores y escritores puede haber mas de un lector accediendo a la imagen pero solo uno modificando los buffers.

finLectura

```
public void finLectura()
```

Método para finalizar el acceso a la imagen para lectura. Una vez realizada la lectura de la imagen indicamos que hay un lector menos, y desbloqueamos a los procesos escritores que estén esperando.

inicioEscritura

```
public void inicioEscritura()
```

Método para iniciar el acceso a la imagen para modificarla. Hay que tener en cuenta que no debe haber ningún lector recogiendo datos de la imagen, ni ninguún escritor dándole valores

finEscritura

```
public void finEscritura()
```

Método para finalizar la modificación de la imagen.

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

controlAplicacion

Class ControlAplicacionException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        controlAplicacion.ControlAplicacionException

All Implemented Interfaces:

`java.io.Serializable`

```
public class ControlAplicacionException
extends java.lang.RuntimeException
```

Esta clase es una excepción que se lanza cuando algo no funciona bien en el módulo de control de la aplicación.

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
`ControlAplicacionException(java.lang.String mensaje)`

Method Summary

Methods inherited from class java.lang.Throwable

`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ControlAplicacionException

```
public ControlAplicacionException(java.lang.String mensaje)
```

controlAplicacion

Class ControlCapturar

java.lang.Object
    controlAplicacion.ControlCapturar

All Implemented Interfaces:

`Control`

```
public class ControlCapturar
extends java.lang.Object
implements Control
```

Esta clase representa el control del proceso de captura de sonido, es la encargada de realizar todas las acciones relacionadas con el cambio de estado del proceso capturar.

Constructor Summary

Constructors

Constructor and Description
`ControlCapturar()`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`configurar()` Método para pasar del estado INICIADO al estado CONFIGURADO
`void`	`crear()` Método para pasar del estado INICIAL al etado INICIADO
`void`	`ejecutar()` Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN
`void`	`finalizar()` Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL
`void`	`parar()` Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ControlCapturar

```
public ControlCapturar()
```

Method Detail

crear

```
public void crear()
```

Método para pasar del estado INICIAL al etado INICIADO

Specified by:

crear in interface Control

configurar

```
public void configurar()
```

Método para pasar del estado INICIADO al estado CONFIGURADO

Specified by:

configurar in interface Control

preparar

```
public void preparar()
```

Método para pasar del estado CONFIGURADO al estado PREPARADO

Specified by:

preparar in interface Control

ejecutar

```
public void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN

Specified by:
`ejecutar` in interface `Control`

pausar

```
public void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

Specified by:
`pausar` in interface `Control`

parar

```
public void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

Specified by:
`parar` in interface `Control`

finalizar

```
public void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

Specified by:
`finalizar` in interface `Control`

controlAplicacion

Class ControlContar

java.lang.Object
  java.util.Timer
    controlAplicacion.ControlContar

All Implemented Interfaces:

Control

```
public class ControlContar
extends java.util.Timer
implements Control
```

Esta clase representa el control del proceso de contar, es la encargada de realizar todas las acciones relacionadas con el paso del tiempo.

Constructor Summary

Constructors

Constructor and Description
ControlContar()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	configurar() Método para pasar del estado INICIADO al estado CONFIGURADO
void	crear() Método para pasar del estado INICIAL al etado INICIADO
void	ejecutar() Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN
void	finalizar() Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL
void	parar() Método para pasar del estado de EJECUCIÓN o PAUSADO al estado

	de PREPARADO
void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO Mediante este método es cuando realmente creamos la clase contar y lo dejamos todo listo para que empiece la cuenta

Methods inherited from class java.util.Timer

`cancel, purge, schedule, schedule, schedule, schedule, scheduleAtFixedRate, scheduleAtFixedRate`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ControlContar

```
public ControlContar()
```

Method Detail

crear

```
public void crear()
```

Método para pasar del estado INICIAL al etado INICIADO

Specified by:

`crear` in interface `Control`

configurar

```
public void configurar()
```

Método para pasar del estado INICIADO al estado CONFIGURADO

Specified by:

`configurar` in interface `Control`

preparar

```
public void preparar()
```

Método para pasar del estado CONFIGURADO al estado PREPARADO Mediante este método es cuando realmente creamos la clase contar y lo dejamos todo listo para que empiece la cuenta

Specified by:

`preparar` in interface `Control`

ejecutar

```
public void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN

Specified by:

`ejecutar` in interface `Control`

pausar

```
public void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

Specified by:

`pausar` in interface `Control`

parar

```
public void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

Specified by:

`parar` in interface `Control`

finalizar

```
public void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

Specified by:

`finalizar` in interface `Control`

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

controlAplicacion

Class ControlGenerar

java.lang.Object
  java.util.Timer
    controlAplicacion.ControlGenerar

All Implemented Interfaces:

`Control`

```
public class ControlGenerar
extends java.util.Timer
implements Control
```

Esta clase representa el control del proceso generar, es la encargada de realizar todas las acciones relacionadas con la generación de las imágenes.

Constructor Summary

Constructors

Constructor and Description
`ControlGenerar()`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`configurar()` Método para pasar del estado INICIADO al estado CONFIGURADO
`void`	`crear()` Método para pasar del estado INICIAL al etado INICIADO
`void`	`ejecutar()` Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN
`void`	`finalizar()` Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL
`void`	`parar()` Método para pasar del estado de EJECUCIÓN o PAUSADO al estado

	de PREPARADO
void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO

Methods inherited from class java.util.Timer

cancel, purge, schedule, schedule, schedule, schedule, scheduleAtFixedRate, scheduleAtFixedRate

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ControlGenerar

```
public ControlGenerar()
```

Method Detail

crear

```
public void crear()
```

Método para pasar del estado INICIAL al etado INICIADO

Specified by:

crear in interface Control

configurar

```
public void configurar()
```

Método para pasar del estado INICIADO al estado CONFIGURADO

Specified by:

configurar in interface Control

preparar

```
public void preparar()
```

Método para pasar del estado CONFIGURADO al estado PREPARADO

Specified by:
`preparar` in interface `Control`

ejecutar

```
public void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN

Specified by:
`ejecutar` in interface `Control`

pausar

```
public void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

Specified by:
`pausar` in interface `Control`

parar

```
public void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

Specified by:
`parar` in interface `Control`

finalizar

```
public void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

Specified by:
`finalizar` in interface `Control`

controlAplicacion

Class ControlIGU

java.lang.Object
    controlAplicacion.ControlIGU

All Implemented Interfaces:

Control

```
public class ControlIGU
extends java.lang.Object
implements Control
```

Esta clase representa el control de la interfaz gráfica, es la encargada de realizar todas las acciones relacionadas con el cambio de estado de la interfaz gráfica.

Constructor Summary

Constructors

Constructor and Description
ControlIGU()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	configurar() Método para pasar del estado INICIADO al estado CONFIGURADO
void	crear() Método para pasar del estado INICIAL al etado INICIADO
void	ejecutar() Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN
void	finalizar() Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL
void	parar() Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ControlIGU

`public ControlIGU()`

Method Detail

crear

`public void crear()`

Método para pasar del estado INICIAL al etado INICIADO

Specified by:
`crear in interface Control`

configurar

`public void configurar()`

Método para pasar del estado INICIADO al estado CONFIGURADO

Specified by:
`configurar in interface Control`

preparar

`public void preparar()`

Método para pasar del estado CONFIGURADO al estado PREPARADO

Specified by:
`preparar in interface Control`

ejecutar

```
public void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN

Specified by:

`ejecutar in interface Control`

pausar

```
public void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

Specified by:

`pausar in interface Control`

parar

```
public void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

Specified by:

`parar in interface Control`

finalizar

```
public void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

Specified by:

`finalizar in interface Control`

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

controlAplicacion

Class ControlPintar

java.lang.Object
 controlAplicacion.ControlPintar

All Implemented Interfaces:

`Control`

```
public class ControlPintar
extends java.lang.Object
implements Control
```

Esta clase representa el control del proceso pintar, es la encargada de realizar todas las acciones relacionadas con la renderización final de las imágenes en el lienzo.

Constructor Summary

Constructors

Constructor and Description
`ControlPintar()`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`configurar()` Método para pasar del estado INICIADO al estado CONFIGURADO
void	`crear()` Método para pasar del estado INICIAL al etado INICIADO
void	`ejecutar()` Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN
void	`finalizar()` Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL
void	`parar()` Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ControlPintar

```
public ControlPintar()
```

Method Detail

crear

```
public void crear()
```

Método para pasar del estado INICIAL al etado INICIADO

Specified by:

`crear in interface Control`

configurar

```
public void configurar()
```

Método para pasar del estado INICIADO al estado CONFIGURADO

Specified by:

`configurar in interface Control`

preparar

```
public void preparar()
```

Método para pasar del estado CONFIGURADO al estado PREPARADO

Specified by:

`preparar in interface Control`

ejecutar

```
public void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN

Specified by:

`ejecutar` in interface `Control`

pausar

```
public void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

Specified by:

`pausar` in interface `Control`

parar

```
public void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

Specified by:

`parar` in interface `Control`

finalizar

```
public void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

Specified by:

`finalizar` in interface `Control`

controlAplicacion

Class ControlProcesos

java.lang.Object
    controlAplicacion.ControlProcesos

All Implemented Interfaces:

Control

```
public class ControlProcesos
extends java.lang.Object
implements Control
```

Esta clase representa el control general de todos los procesos de la aplicación.

Constructor Summary

Constructors

Constructor and Description
ControlProcesos()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	configurar() Método para pasar del estado INICIADO al estado CONFIGURADO
void	crear() Método para pasar del estado INICIAL al etado INICIADO
void	ejecutar() Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN
void	finalizar() Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL
Estados	getEstadoAplicacion() Metodo para recuperar el estado de la aplicación
void	parar()

	Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO
void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ControlProcesos

```
public ControlProcesos()
```

Method Detail

crear

```
public void crear()
```

Método para pasar del estado INICIAL al etado INICIADO

Specified by:

`crear in interface Control`

configurar

```
public void configurar()
```

Método para pasar del estado INICIADO al estado CONFIGURADO

Specified by:

`configurar in interface Control`

preparar

```
public void preparar()
```

Método para pasar del estado CONFIGURADO al estado PREPARADO

Specified by:

`preparar in interface Control`

ejecutar

```
public void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN

Specified by:

`ejecutar in interface Control`

pausar

```
public void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

Specified by:

`pausar in interface Control`

parar

```
public void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

Specified by:

`parar in interface Control`

finalizar

```
public void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

Specified by:

`finalizar in interface Control`

getEstadoAplicacion

```
public Estados getEstadoAplicacion()
```

Metodo para recuperar el estado de la aplicación

Returns:

`el estado de la aplicación`

controlAplicacion

Class ControlSecuenciar

java.lang.Object
  java.util.Timer
    controlAplicacion.ControlSecuenciar

All Implemented Interfaces:

Control

```
public class ControlSecuenciar
extends java.util.Timer
implements Control
```

Esta clase representa el control del proceso de secuenciar, es la encargada de realizar todas las acciones relacionadas con las variaciones de la generación de las imágenes.

Constructor Summary

Constructors

Constructor and Description
ControlSecuenciar()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	configurar() Método para pasar del estado INICIADO al estado CONFIGURADO
void	crear() Método para pasar del estado INICIAL al etado INICIADO.
void	ejecutar() Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN como la tarea ya esta programada lo único es cambiar de estado ya que en la propia tarea tendremos contemplado que se debe o no hacer en cada uno de los casos
void	finalizar() Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

void	parar() Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO
void	pausar() Método para pasar del estado de EJECUCIÓN al estado PAUSADO
void	preparar() Método para pasar del estado CONFIGURADO al estado PREPARADO

Methods inherited from class java.util.Timer

`cancel, purge, schedule, schedule, schedule, schedule, scheduleAtFixedRate, scheduleAtFixedRate`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ControlSecuenciar

`public ControlSecuenciar()`

Method Detail

crear

`public void crear()`

Método para pasar del estado INICIAL al etado INICIADO. Realmente no hacemos nada.

Specified by:

`crear in interface Control`

configurar

`public void configurar()`

Método para pasar del estado INICIADO al estado CONFIGURADO

Specified by:

`configurar in interface Control`

preparar

```
public void preparar()
```

Método para pasar del estado CONFIGURADO al estado PREPARADO

Specified by:

`preparar in interface Control`

ejecutar

```
public void ejecutar()
```

Método para pasar del estado PREPARADO o PAUSADO al estado de EJECUCIÓN como la tarea ya esta programada lo único es cambiar de estado ya que en la propia tarea tendremos contemplado que se debe o no hacer en cada uno de los casos

Specified by:

`ejecutar in interface Control`

pausar

```
public void pausar()
```

Método para pasar del estado de EJECUCIÓN al estado PAUSADO

Specified by:

`pausar in interface Control`

parar

```
public void parar()
```

Método para pasar del estado de EJECUCIÓN o PAUSADO al estado de PREPARADO

Specified by:

`parar in interface Control`

finalizar

```
public void finalizar()
```

Método para pasar de cualquier estado salvo el INCIAL al estado INICIAL

Specified by:

`finalizar in interface Control`

controlAplicacion

Enum Estados

java.lang.Object
    java.lang.Enum<Estados>
        controlAplicacion.Estados

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<Estados>

```
public enum Estados
extends java.lang.Enum<Estados>
```

Este enumerado define los diferentes estados por los que pasan diferentes procesos de la aplicación. Todos ellos obedecen al mismo conjunto de estados que son los estados de la aplicación. Las aplicación consta de los siguientes procesos: - Capturar. Es el proceso que se encarga de recoger los datos de las diferentes líneas de la tarjeta de sonido. Este proceso es manejado principalmente por el driver. Nosotros a través de un oyente accedemos a los buffers para recoger las muestras. Actualmente esta implementado usando el protocolo ASIO y la librería jasiohost. - Secuenciar. Es el proceso que maneja la lista de imágenes que tendremos que renderizar y dibujar. Para cada una de las imágenes de la secuenciación tendremos nodos temporales asociados a las líneas de sonido, que indican las variaciones de visualización de las muestras. - Contar. Este proceso es el mas simple de todos y solo contabiliza el tiempo que transcurre en segundos. - Generar. Es el proceso que se encarga de ir generando las imágenes en función de las líneas de sonido involucradas en cada momento y del algoritmo de visualización. - Pintar. Es el proceso que se encarga de refrescar el lienzo y pintar las imágenes ya generadas. Lo controla la IGU de la aplicación. El ciclo de vida de cada uno de estos procesos y el de la aplicación sigue el siguiente modelo de estados: - Inicial. Proceso inicial no ha pasado nada. - Inicializado. Para llegar a este estado se ha de crear las instancias de las clases de los procesos y la IGU principal. - Configurado. Para llegar a este estado se deben configurar todos los procesos, por ejemplo recuperando los datos de los ficheros XML en el caso de la secuenciación. - Preparado. Los datos que hemos configurado se deben preparar, como sería inicializar el recorrido de todas las listas. Para procesos simples el número de estados puede que sea mas simple. No obstante en vistas a generalizar lo máximo posible vamos a incluir todos los posibles estados que nos encontramos. - Los siguientes estados ya obedecen a un ciclo clásico de ejecución, pausado y vuelta a preparado. En cualquiera de los estados que estemos siempre se puede acabar la aplicación y tendremos que regresar al estado inicial, parando adecuadamente todos los hilos de ejecución que hayamos iniciado.

Enum Constant Summary

Enum Constants

Enum Constant and Description
CONFIGURADO
EJECUCION

INICIADO
INICIAL
PAUSADO
PREPARADO

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static Estados	valueOf(java.lang.String name) Returns the enum constant of this type with the specified name.
static Estados[]	values() Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum

compareTo, equals, getDeclaringClass, hashCode, name, ordinal, toString, valueOf

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Enum Constant Detail

INICIAL

```
public static final Estados INICIAL
```

INICIADO

```
public static final Estados INICIADO
```

CONFIGURADO

```
public static final Estados CONFIGURADO
```

PREPARADO

```
public static final Estados PREPARADO
```

EJECUCION

```
public static final Estados EJECUCION
```

PAUSADO

```
public static final Estados PAUSADO
```

Method Detail

values

```
public static Estados[] values()
```

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (Estados c : Estados.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

valueOf

```
public static Estados valueOf(java.lang.String name)
```

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

name - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

java.lang.IllegalArgumentException - if this enum type has no constant with the specified name

java.lang.NullPointerException - if the argument is null

controlAplicacion

Class LanzarAplicacion

java.lang.Object
    controlAplicacion.LanzarAplicacion

```
public class LanzarAplicacion
extends java.lang.Object
```

En esta es la clase principal que lanza la aplicación, es donde encontramos el método main que lanza la interfaz gráfica principal.

Constructor Summary

Constructors

Constructor and Description
`LanzarAplicacion()`

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`static void`	`main(java.lang.String[] args)`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

LanzarAplicacion

```
public LanzarAplicacion()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

controlAplicacion

Class Reloj

java.lang.Object
  java.util.TimerTask
    controlAplicacion.Reloj

All Implemented Interfaces:

java.lang.Runnable

```
public class Reloj
extends java.util.TimerTask
```

Esta clase representa un reloj, un contador de tiempo. En el momento que se pone en marcha empieza a contar el número de segundos transcurridos y su valor lo va almacenando en el atributo segundos. El reloj se pondrá en marcha cuando la aplicación pase a estado de ejecución, y se parará cuando pase a estado de parar o a estado de preparado, que en este caso se pondrá a cero. Por lo tanto tiene un funcionamiento muy sencillo.

Constructor Summary

Constructors

Constructor and Description
`Reloj()` Constructor para la clase Reloj.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`long`	`getSegundos()` Método para obtener el número de segundos.
`void`	`iniciar()` El método iniciar, inicializa los segundos y recoge el tiempo actual.
`void`	`run()` El método run, se ejecutará cada segundo.

Methods inherited from class java.util.TimerTask

`cancel, scheduledExecutionTime`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Reloj

```
public Reloj()
```

Constructor para la clase Reloj. No tiene parámetros, lo hace es inicializar el tiempo, ponemos los segundos a cero y recogemos la hora actual.

Method Detail

iniciar

```
public void iniciar()
```

El método iniciar, inicializa los segundos y recoge el tiempo actual. hace exactamente lo mismo que el constructor de la clase

run

```
public void run()
```

El método run, se ejecutará cada segundo. Su programación se realiza a través de la clase ControlContar, que será un Timer encargado entre otras cosas de lanzar esta tarea

Specified by:

`run in interface java.lang.Runnable`

Specified by:

`run in class java.util.TimerTask`

getSegundos

```
public long getSegundos()
```

Método para obtener el número de segundos.