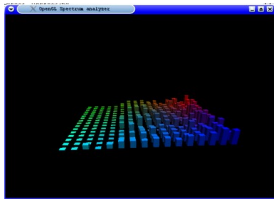
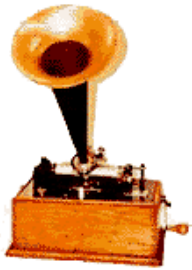


Arquitecturas y Entornos de desarrollo para Videoconsolas



Audio

101010
101010



Grado de Ingeniero en Informática
Escola Tècnica Superior d'Enginyeria Informàtica
Curso 2020/2021

Objetivos

- Conocer y comprender los fundamentos del sonido
- Conocer y comprender los fundamentos de la representación de audio en digital
- Comparar los diferentes formatos de almacenamiento de señales de audio
- Aplicar los conocimientos a la plataforma de estudio

Índice

- *Introducción*
 - *Sonido y percepción*
 - *Características del sonido*
- *Adquisición de señales de audio*
 - *Digitalización*
 - *Concepto de bitrate*
- *Representación del audio*
 - *Forma de ondas y MIDI*
 - *Edición y modelado del audio*
- *Formatos de almacenamiento*
 - *Compresión de la información multimedia*
 - *Compresión con pérdidas en el audio*

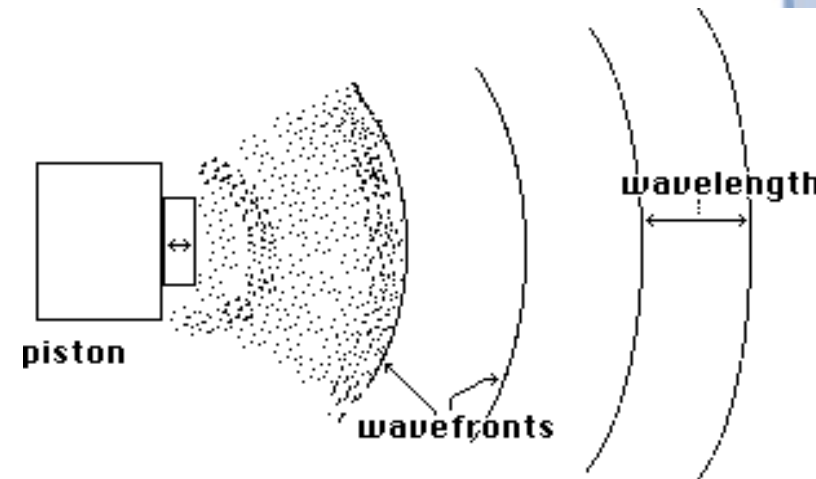
• ...

Índice

- ...
- *Caso de estudio: plataforma NDS*
 - *Hardware*
 - *API libnds + maxmod*
 - *Práctica asociada*
- *Caso de estudio: plataforma 3DS*
 - *Hardware*
 - *API libctru*
 - *Práctica asociada*
- **Bibliografía**

Sonido y percepción

- Naturaleza del sonido
- Onda de presión
 - Vibración de las moléculas en un medio (p. ej., aire).
 - Produce una sensación “auditiva” al perturbar el estado de reposo del oído.
- Intensidad
 - Potencia recibida por unidad de área (W/m^2)
 - En intensidades de sonido se realizan definiciones relativas al *mínimo nivel de intensidad audible* (10^{-12} W/m^2)
 - Bel (B), Decibel (dB): $1\text{B} = 10\text{dB}$
 - Rango dinámico



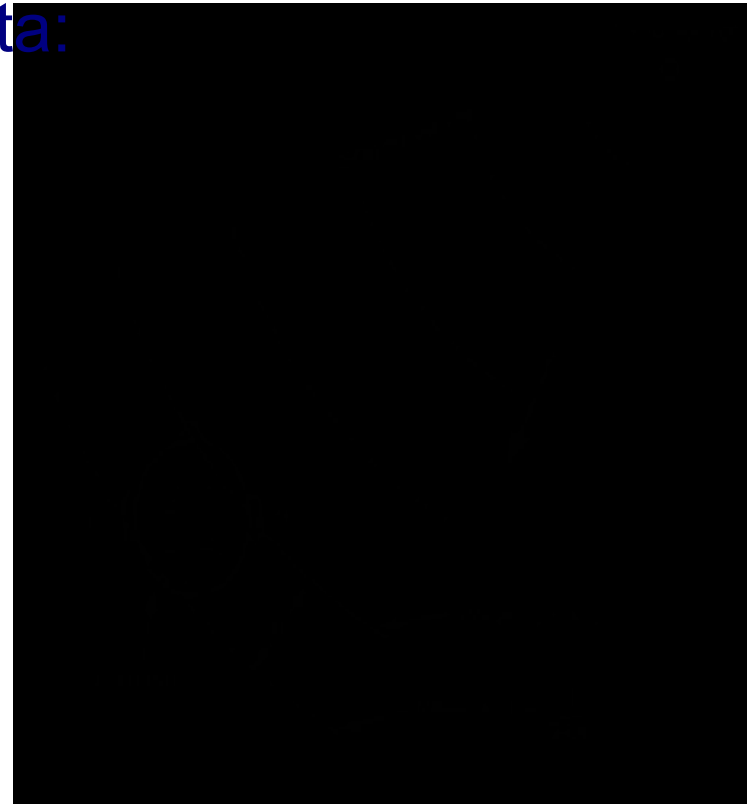
Sonido y percepción (y II)

- Ejemplos de sonidos y su cuantificación en dB

<i>Fuente de sonido</i>	<i>Intensidad (W/m²)</i>	<i>Intensidad relativa</i>	<i>Reacción del que escucha</i>	<i>Intensidad en decibeles (dB)</i>
	10^4	10^{16}		160
Turbina de jet a 10m	10^3	10^{15}	Daño inmediato	150
	10^2	10^{14}	Dolor de oído	140
	10	10^{13}		130
	1	10^{12}	Desagrado	120
Trueno	10^{-1}	10^{11}		110
Cataratas del Niagara	10^{-2}	10^{10}		100
	10^{-3}	10^9		90
Fábrica	10^{-4}	10^8		80
Tráfico de ciudad a 15m.	10^{-5}	10^7		70
Conversación normal a 1m.	10^{-6}	10^6		60
Residencia suburbana	10^{-7}	10^5		50
Biblioteca	10^{-8}	10^4		40
	10^{-9}	10^3		30
Estudio de grabación	10^{-10}	10^2		20
Respiración	10^{-11}	10		10
	10^{-12}	1	Límite audible	0

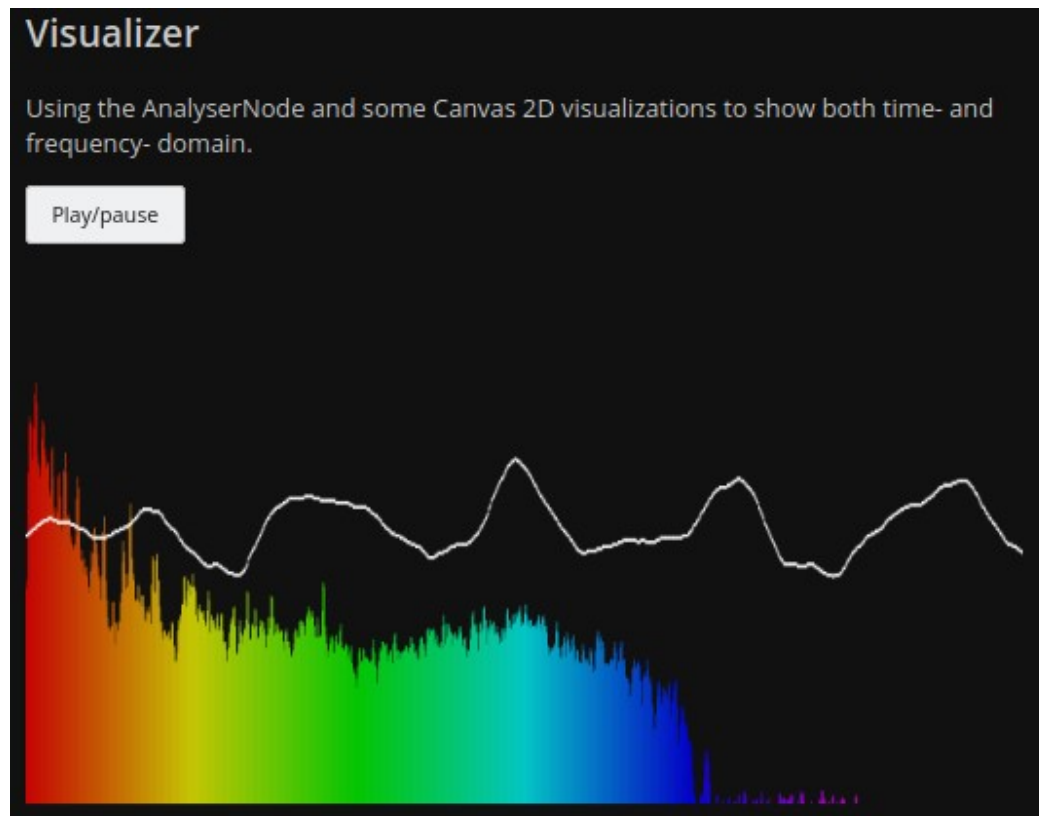
Sonido y Acústica

- Sonido vs Audio
 - Impresión fisiológica producida en el oído por las vibraciones elásticas de los cuerpos.
 - Perceptible por el oído humano
- Y el sistema de percepción interpreta:
 - Hz: 20 .. 20k,
 - edad, intensidad, duración.
 - dB (umbral audible)
 - Relativa a 0dB
 - Escala logarítmica
dB: 20 .. 135
 - Dirección del sonido
- Características del sonido
 - Tiempo vs frecuencia



Sonido y Acústica

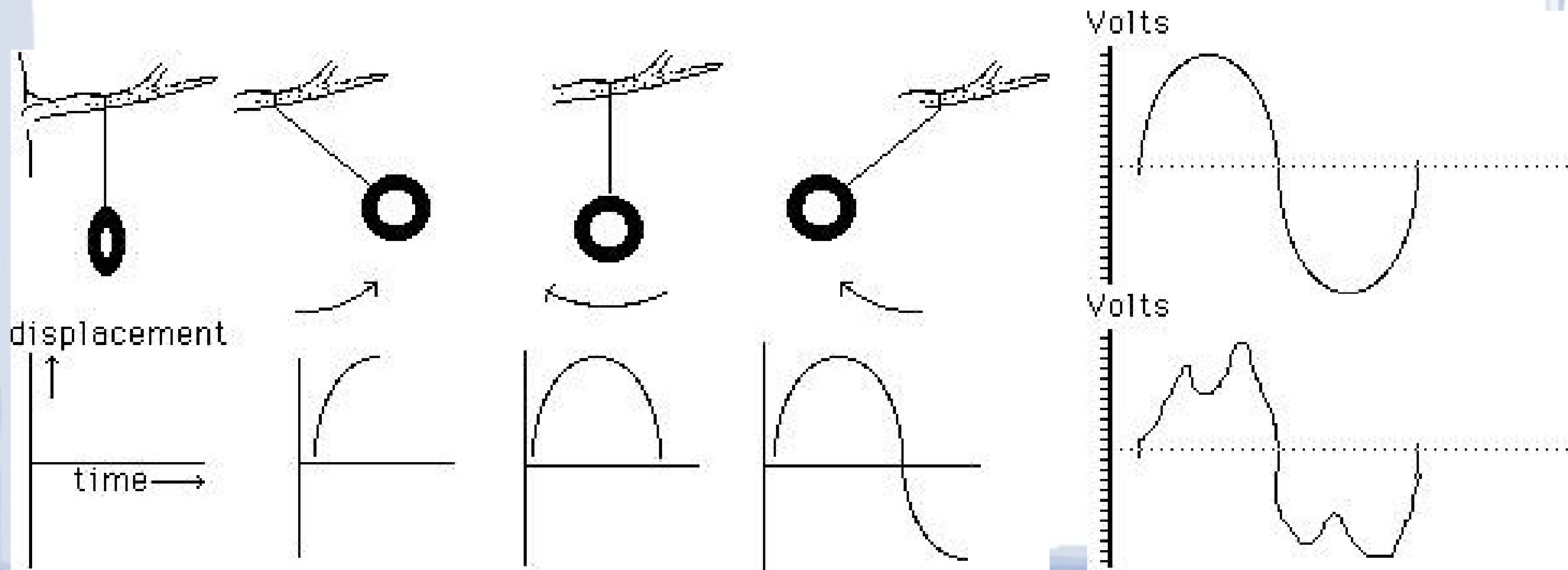
- Sonido vs Audio
- Y el sistema de percepción interpreta
- Características del sonido
 - Tiempo vs frecuencia



Características del sonido

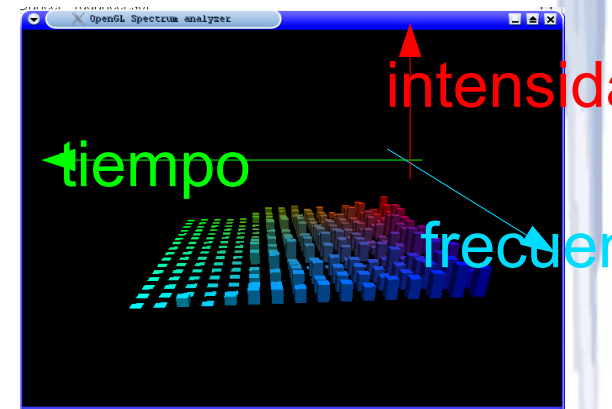
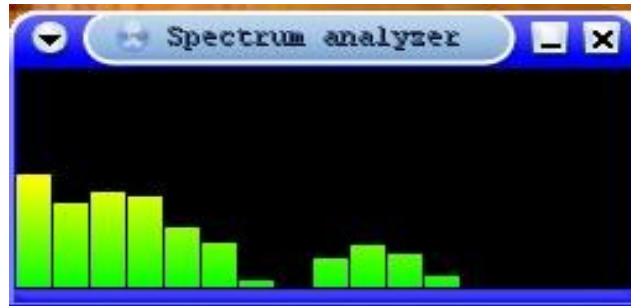
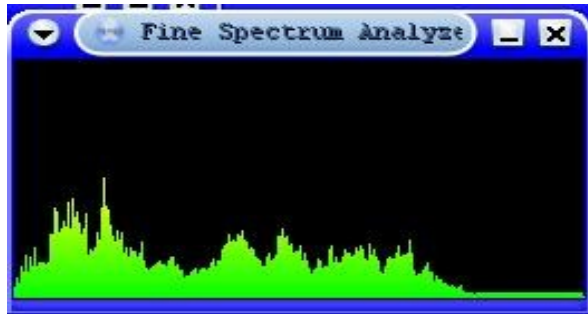
- Modelado en el dominio del tiempo

- **Sonoridad** (*loudness*): es la intensidad subjetiva, depende principalmente de las **amplitudes** de las comp. del sonido.
- **Altura** (*pitch*): **frecuencia** percibida
- **Timbre** : frec. fundamental y armónicos (calidad del sonido).
 - Pueden tener la misma amplitud y frec. fundamental pero suenan diferente



Características del sonido (y 2)

- Modelado del sonido en frecuencia
 - Espectro de frecuencias



- Ecualizador



Adquisición y reproducción

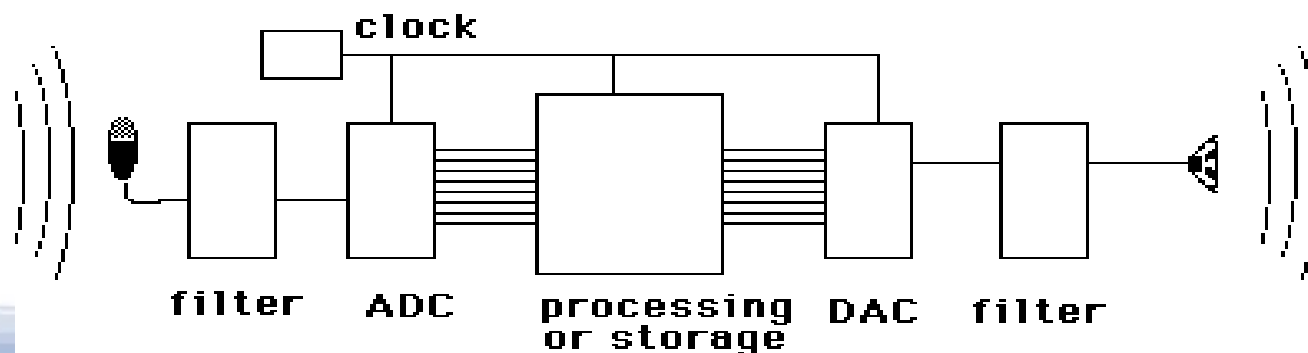
- Adquisición

- ★ Un micrófono vibra de acuerdo con la presión ejercida por las ondas sonoras sobre su membrana.

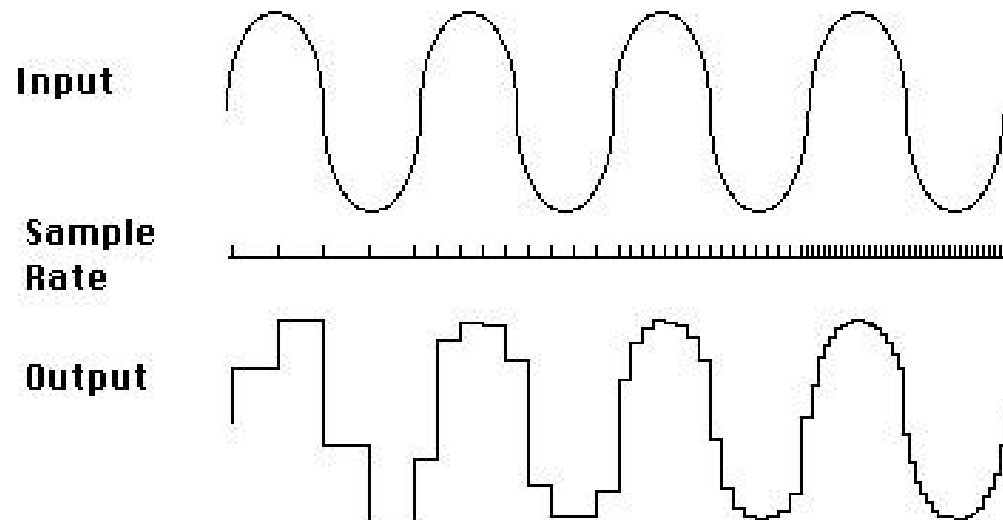
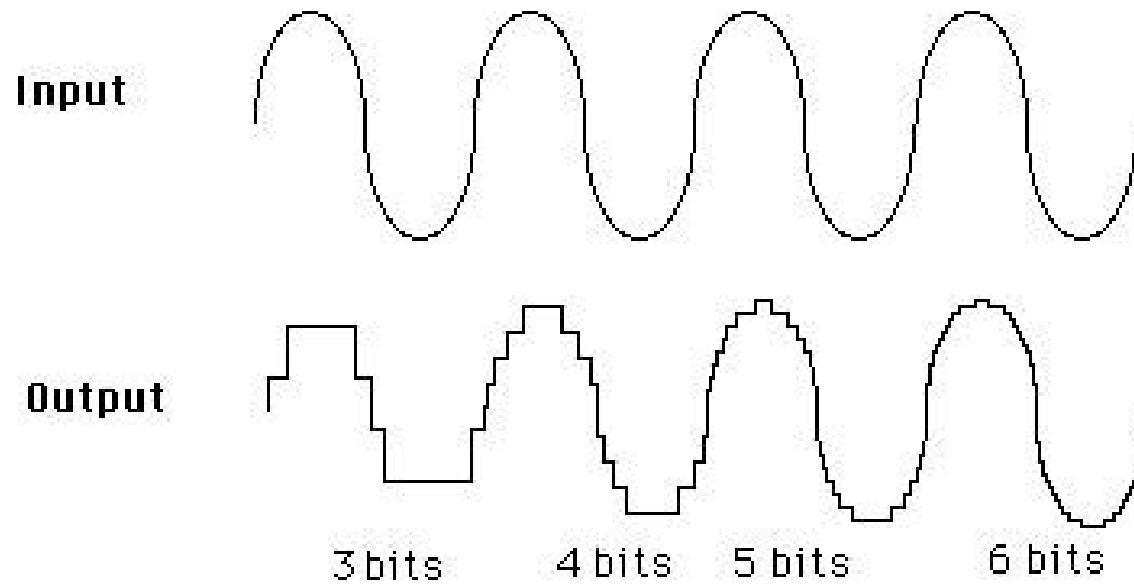
- Se transforma esta energía mecánica en energía eléctrica.

- Reproducción

- ★ Un altavoz transforma la energía eléctrica en ondas sonoras



Digitalización



Digitalización

- Digitalización de 60 seg. de audio

<i>Velocidad de muestreo</i>	<i>Resolución en bits</i>	<i>Canales Mono ó Estéreo</i>	<i>Tamaño del archivo resultante (Bytes)</i>	<i>Características y posibilidades de la grabación</i>
11 KHz	8	1	660 KB	Esta es la capacidad más baja que se puede emplear en la práctica para obtener resultados útiles. Sonido poco claro y apagado.
11 KHz	8	2	1.3 MB	No es frecuente utilizar 2 canales a esta velocidad de muestreo.
11 KHz	16	1	1.3 MB	Sonido monoaural con calidad radiofónica.
11 KHz	16	2	2.6 MB	El estéreo se nota y mejora ligeramente.
22.05 KHz	8	1	1.3 MB	Sonido monoaural con calidad equiparable a la de los televisores clásicos con sonido analógico. Buena elección para grabaciones de diálogos.
22.05 KHz	8	2	2.6 MB	Elección popular para grabaciones estéreo con calidad aceptable.
22.05 KHz	16	1	2.6 MB	No es una mala elección para voz, pero si reducimos a 8 bits, ahorraremos tamaño y sacrificaremos un poquito de fidelidad.
22.05 KHz	16	2	5.25 MB	Sonido de calidad comparable a la de una buena cinta de sonido.
44.1 KHz	8	1	2.6 MB	Es una elección adecuada para grabaciones monoaurales como diálogos y narraciones con buena calidad.
44.1 KHz	8	2	5.25 MB	La mejor calidad que se podía conseguir en las primeras tarjetas de sonido.
44.1 KHz	16	1	5.25 MB	La elección ideal para narraciones.
44.1 KHz	16	2	10.5 MB	Calidad de grabación de un CD-Audio.

Concepto de *bitrate*

- Tasa de transferencia de bits
 - Cantidad de información digital / tiempo
 - Generalmente se expresa en kbits/segundo
kbits/s, kb/s o kbps.
- Se suelen utilizar múltiplos decimales (SI)
 - $1000 \text{ bit/s} = 1 \text{ kbit/s}$ (un kilobit o mil bits por segundo)
 - $1000 \text{ kbit/s} = 1 \text{ Mbit/s}$ (un megabit o un millón de bits por segundo)
 - $1000 \text{ Mbit/s} = 1 \text{ Gbit/s}$ (un gigabit o mil millones de bits por segundo)
- Suele ser un indicador de la calidad de la información MM

Representaciones del audio digital

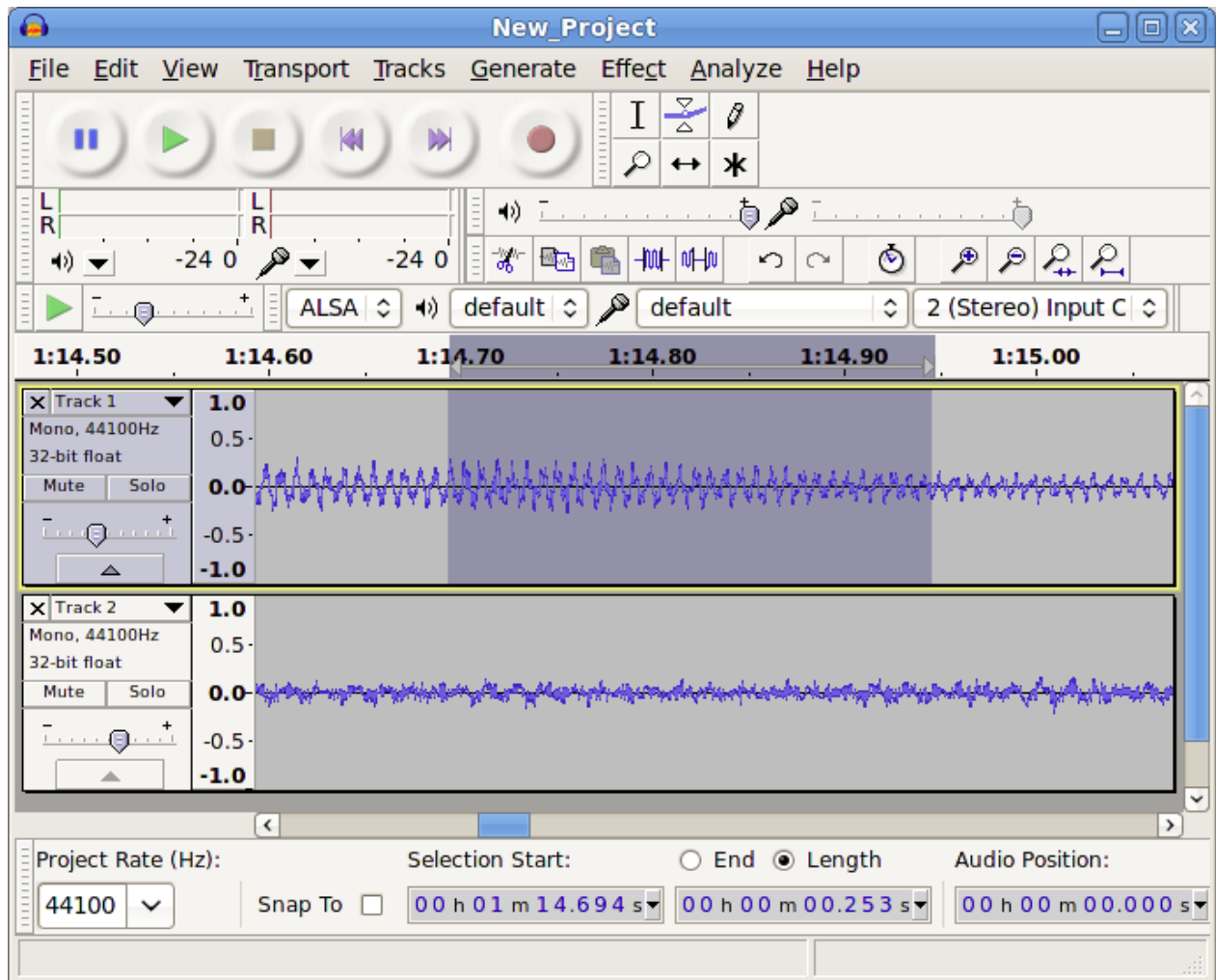
- **Forma de onda (*waveform*)**
 - La señal de audio original es digitalizada (p. ej, WAVE)
 - ***Compact Disc Digital Audio (CD-DA)***
 - Sonido digital de calidad CD
 - **DVD-Audio**
 - Frecuencias de muestreo de 96 kHz en 5.1 (192 kHz en estéreo) y 24-bits de ancho de palabra
 - **Audio continuo (“*streaming audio*”)**
 - Permite difundir audio en vivo, simultáneo y uno a muchos.
- **Audio estructurado (MIDI)**
 - No “graba” el sonido generado por los instrumentos sino cómo se genera.

Edición y modelado de sonido

- Edición
 - Forma de ondas (*Waveform*)
vs Secuenciadores (*Trackers*)
- Efectos básicos
- Síntesis

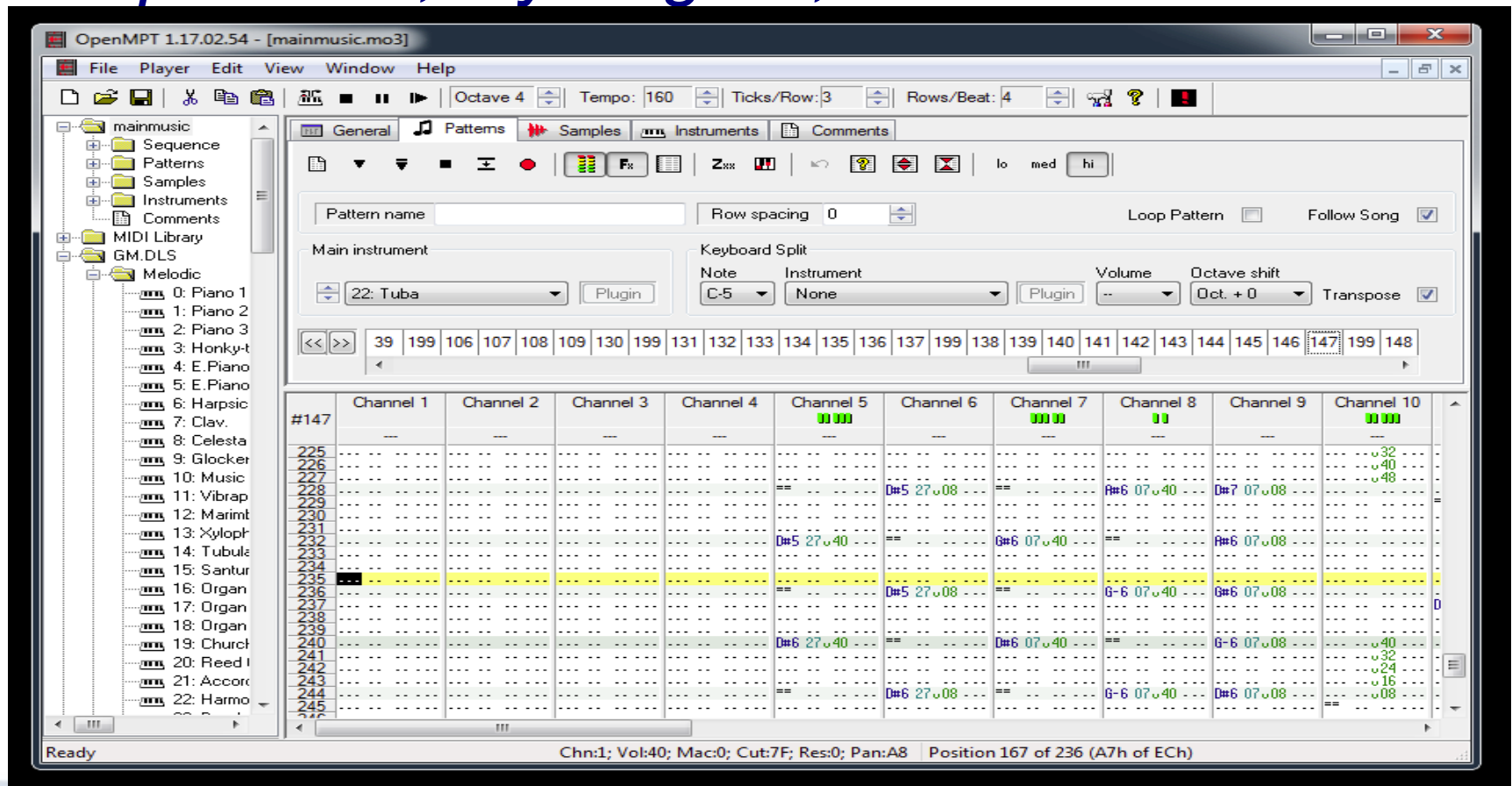
Edición y modelado de sonido (II)

- Edición en forma de ondas
 - Audacity



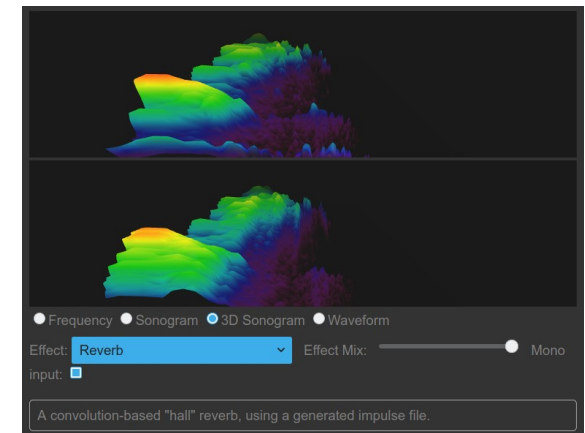
Edición y modelado del sonido (III)

- Edición de pistas
 - Secuenciadores (*Trackers*) y *Module files*
 - *OpenMPT*, *Hydrogren*, ...

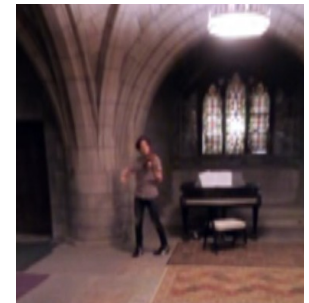
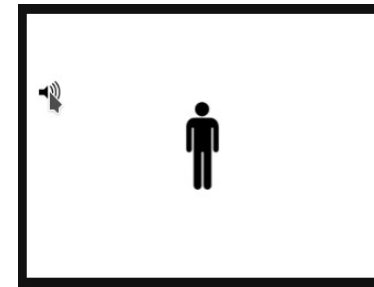


Edición y modelado del sonido (IV)

- Edición
- Efectos básicos
 - Retraso (*delay*).
 - Coros (*chorus*)
 - Eco (*echo*)
y reverberación (*reverb*)
 - Ecualizar.
 - Filtros (*FX*): doppler, ...
 - Ambientación espacial: audio 3D

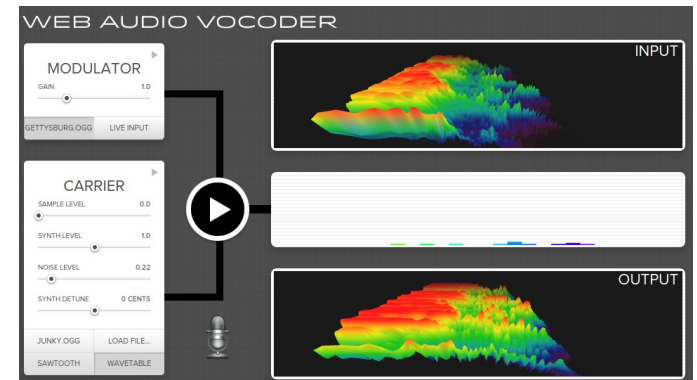


- Síntesis



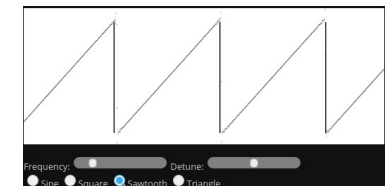
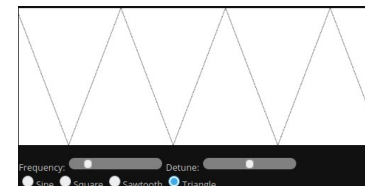
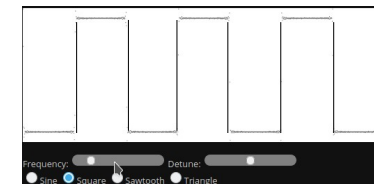
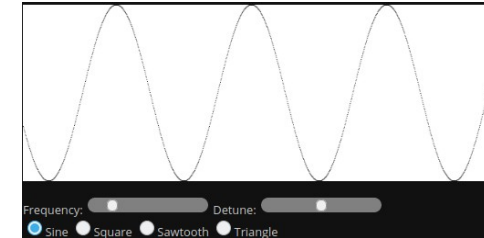
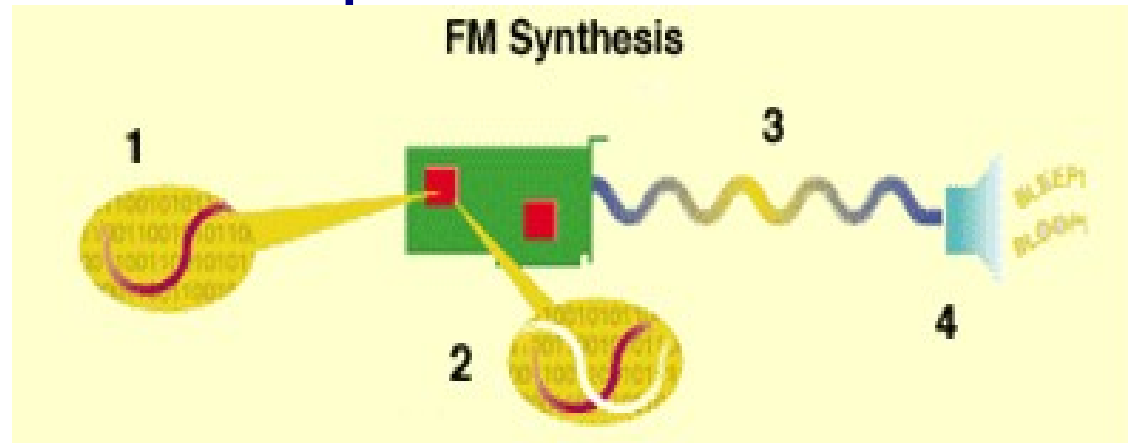
Edición y modelado del sonido (V)

- Edición
- Efectos básicos
- Síntesis
 - Voz
 - Vocoder
 - TTS / SR
 - Sonidos
 - FM: modulación de señal (envolvente)
 - Tabla de ondas
 - Modelado por síntesis aditiva, sustractiva, modular, ...



Edición y modelado del sonido (VI)

- Síntesis por modulación de señal (FM)

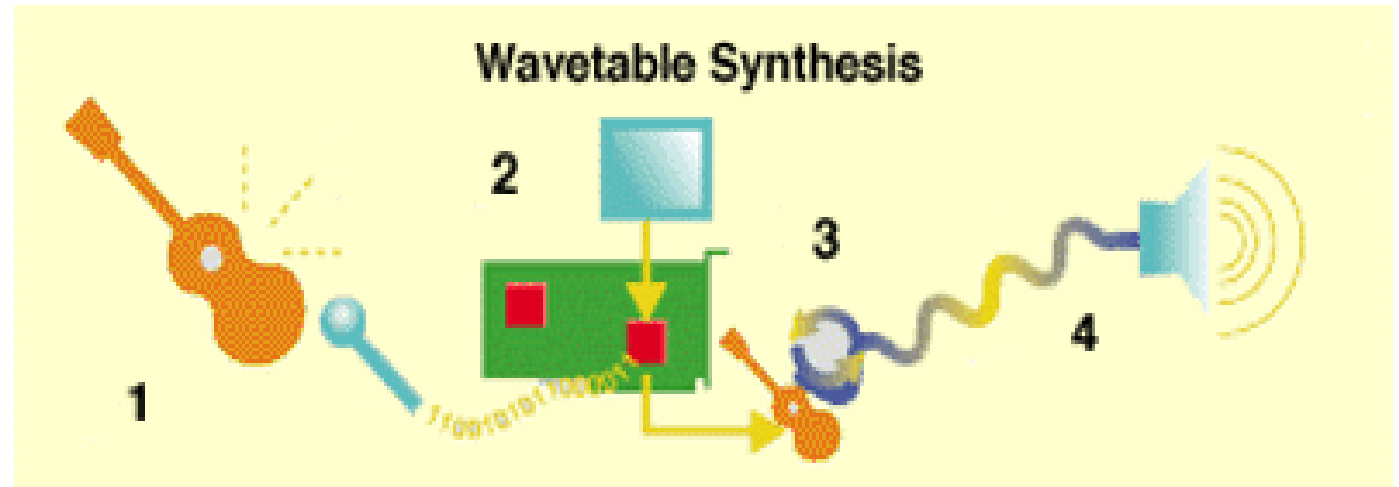


1. El chip de síntesis genera ondas senoidales/cuadrangulares/...
2. Dos o más ondas se combinan para formar señales más complejas... (modulación)
3. ... que son convertidas a señales analógicas

Los sonidos generados por modulación de frecuencia tienden a parecer artificiales.

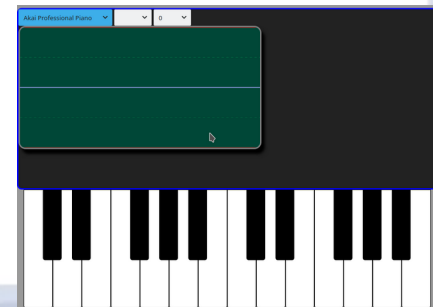
Edición y modelado del sonido (VII)

- Síntesis por tabla de ondas



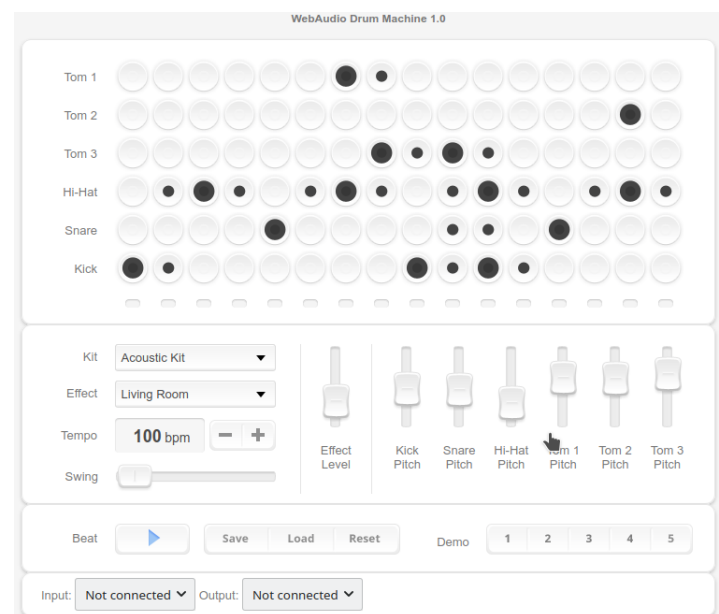
1. Los sonidos de instrumentos se graban en estudio
2. Pequeñas muestras se guarda en ROM (tabla de ondas)
3. Las aplicaciones los utilizan.

La calidad es buena si lo son las muestras. Sin embargo, para reproducir con fidelidad un sonido no basta con guardar una grabación, ya que cuando se toca una nota diferente, no sólo se cambia la frecuencia del sonido sino otros parámetros importantes.



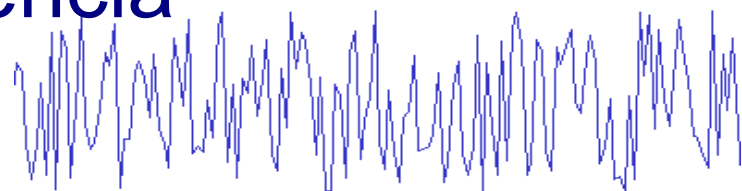
Edición y modelado del sonido (VIII)

- Síntesis por tabla de ondas
 - Utilizada en sintetizadores software, samplers y cajas de ritmos (drum machines)
- P. ej.: Hydrogen,
Shiny Happy WebAudio
MIDI-fied Drum Machine



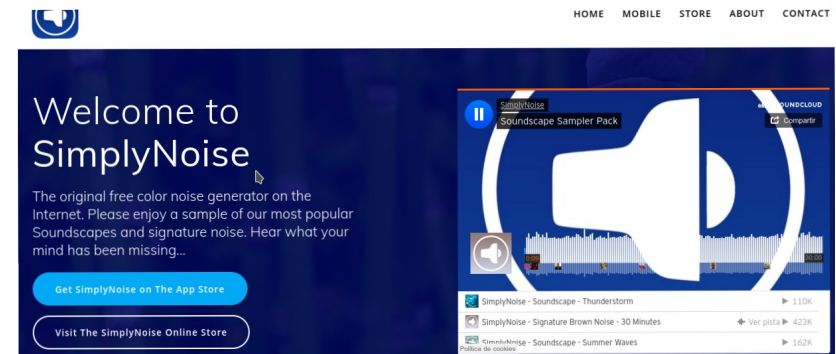
Edición y modelado de sonido (IX)

- Síntesis aditiva, sustractiva, modular, granular, ...
 - Modelado del sonido en frecuencia + Ecualización
- Modelado del sonido en frecuencia
 - Generación: ruido blanco
 - Señal no correlada en el **eje del tiempo**: la señal toma valores sin ninguna relación unos con otros.
 - Tiene una densidad espectral de potencia plana o casi: en el **dominio de la frecuencia** veríamos todas las componentes con la misma (o casi) amplitud.
 - Sino, se dice que el ruido está "coloreado"
 - Ejemplos en la wikipedia: Color Noise



Edición y modelado del sonido (X)

- Ejemplo: generador de ruido
 - SimplyNoise <<http://simplynoise.com/>>, SimplyRain --> Apps

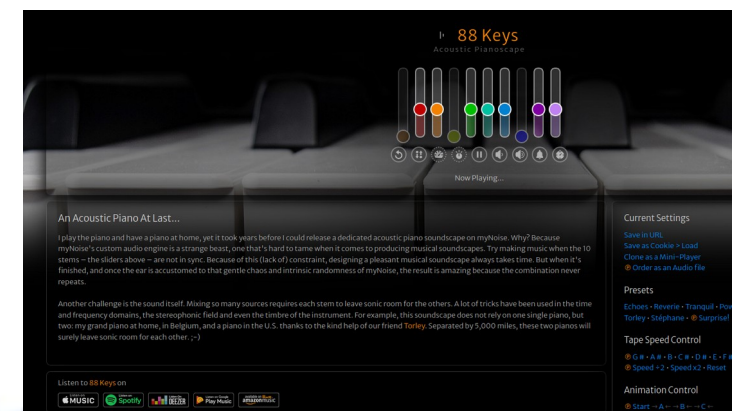
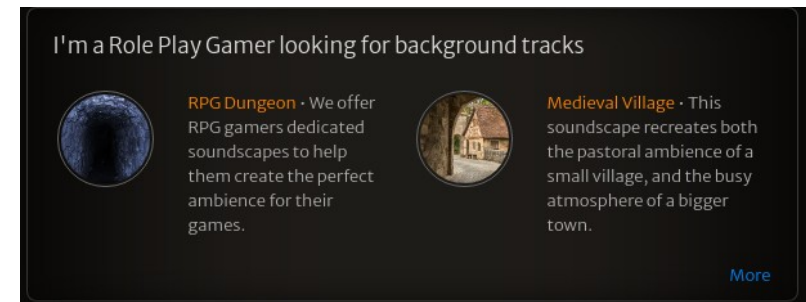
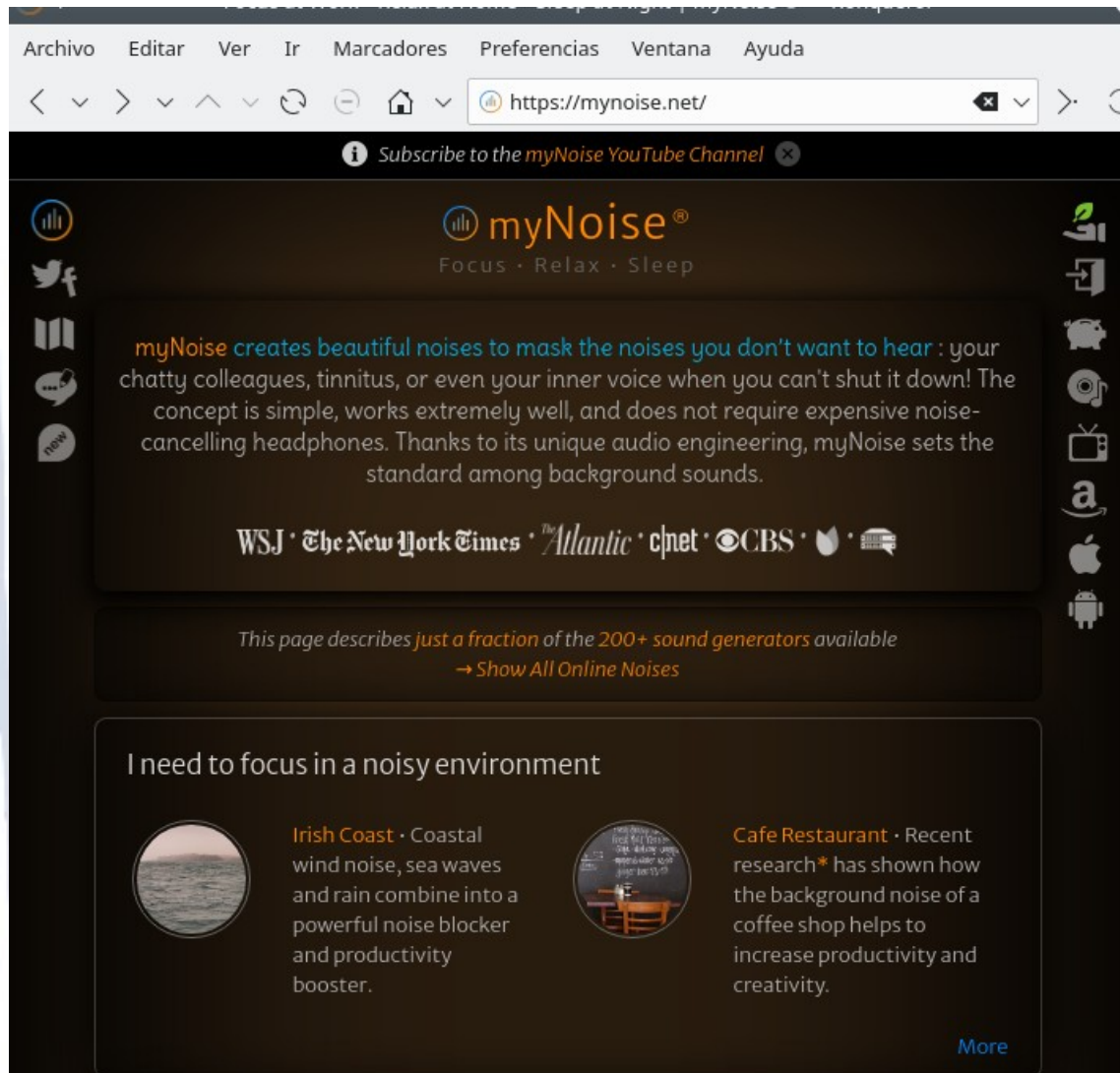


- Noise machine, Rain Noise
 - Custom-shaped Online Noise Machines
 - Frequency-Shaped Natural Rain Noise Generator
 - ...



Edición y modelado del sonido (y XI)

- Recreaciones sonoras ← myNoise



El estándar MIDI

- *Musical Instrument Digital Interface (1981)*
 - Protocolo diseñado para la grabación y reproducción de música en sintetizadores digitales
- Multiplataforma:
independiente
de
dispositivo y
resolución



El estándar MIDI (2)

- MIDI no transmite sonidos sino más bien mensajes a los que un dispositivo responde generando un sonido:
 - Canal (instrumento), nota, fuerza (velocidad), presión, forma de la interpretación, ...
- Descripción:
 - Dispositivos: emisores y/o receptores (*μprocs.*, *puertos*)
 - Canales: 1-16 simultáneos (Máx. 128 instrumentos reales o virtuales)
 - Mensajes. *General MIDI (GM)*: estándar basado en un conjunto común de 128 sonidos.

Ejemplo de dispositivo MIDI

- Bateria MIDI (Ride cymbal | Toms | Bass drum | Snare | Hi-hat | Crash)
 - Caja sorda (*patch*) + sensor
 - PIC
 - *Hydrogen*



A l'interior de la caixa sorda hem introduït un sensor piezoelèctric. El conjunt va connectat al cervell electrònic per una de les seues entrades

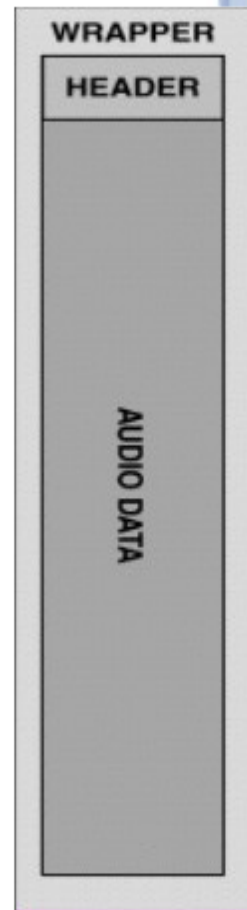
Un PIC transforma els impulsos rebuts a senyals MIDI que són enviades a l'ordinador

El PC, amb el programari adequat, llança en temps real un so digitalitzat per a cada nota, intentant emular de la forma més realista possible l'instrument desitjat

El diagrama mostra la connexió entre la bateria MIDI, un PIC, un cervell electrònic i un PC. Una bateria MIDI està connectada a un PIC, que a la seva vegada està connectat a un cervell electrònic. El cervell electrònic està connectat a un PC. El PC està connectat a un monitor i un altaveu.

Formatos de almacenamiento

- Forma estructurada
 - * MOD
 - * S3M, IT y XM
- Formas de onda: 2 estilos
 - * Sólo datos (*raw*)
 - Estándar Audio CD (CD-A)
 - $\approx 700\text{MB}$; ≈ 74 minutos;
44.1KHz, 16 bits PCM, 2 canales
 - * Con cabecera
 - Metadatos
 - freq. muestreo, n° de bits, n° de canales, etc.
 - + sonido (PCM)
 - Codificaciones lineales y logarítmicas
 - LPCM (PCM) vs ADPCM,
 μ -law, A-law, ...



Formatos de almacenamiento

- Formas de onda: 2 estilos

- * Con cabecera

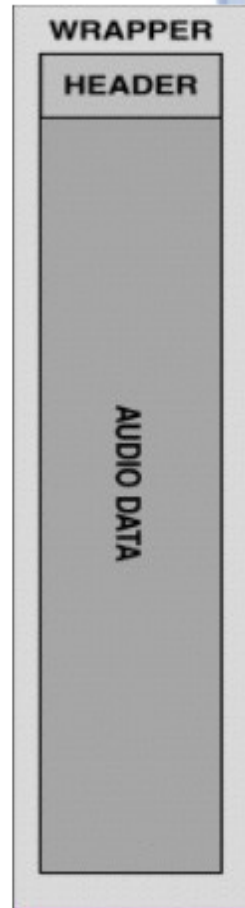
- Metadatos
 - + sonido (PCM)

- * Formatos

- Sin compresión
 - Con compresión
 - Sin pérdidas (lossles)
 - Con pérdidas (lossy)

- * Contenedores (*Wrappers*):

- Vídeo: QuickTime, AVI, Matroska, MP4, Ogg
 - Sólo audio: MXF



Formatos de almacenamiento

- Formatos sin compresión (o comprimidos sin pérdidas)

Extensión	Nombre	Origen	Comentarios
.au ó .snd	AU mu-law	NeXT, Sun	Frecuencia de muestreo variable. Soportado por todas las plataformas
.aif(f)	AIFF AIFC	Apple (Mac) SGI	Frec. de muestr., tamaño de la muestra y núm de canales variables. AIFC: comprimido.
.iff	IFF/8SVX	Amiga	Frec. de muestr. y núm de canales variables. Sólo 8 bits.
.voc	VOC	Soundblaster	Frec. de muestr. variable. Sólo 8 bits, 1 canal.
.wav	RIFF, WAVE	Microsoft	Frec. de muestr., tamaño de la muestra y núm de canales variables.
.mod .nst		Amiga	

- Actualmente
 - MOD → IT, XM, S3M
 - FLAC (*Free Lossless Audio Codec*)

Formatos de almacenamiento (3)

Formatos con compresión con pérdidas

- **MP3** o *MPEG-1 Layer 3* (y otros que siguen el estándar MPEG): de 2:1 a 120:1, ideal para música con 128 kbps (.mp3)
- **AAC** or *Advanced Audio Coding*: potenciado por Apple, es parte de las especificaciones de MPEG-2 y MPEG-4 (.aac, .m4a, .m4p)
- **MP3-Pro**: mejora a altas frecuencias permitiendo *bitrates* de 64 kbps (.mp3, .mp3pro)
- **Real Audio**: audio continuo de Real Networks, hasta 60:1, ideal para voz (.ra, .rm, or .ram)
- **Windows Media Audio**: propio de Microsoft, puede codificar a 64kbps con calidad CD (.wma)
- **Ogg Vorbis**: sin patentes, de características similares al MP3-Pro (.ogg)
- ¿Cómo seleccionar los mejores codecs de audio?
 - Utilizando *codec listening tests*
 - La mayoría siguen la forma de una comparación “doble-ciego”
 - Para obtener resultados significativos, deben utilizar tasas de bits (bitrates) idénticos o muy similares.

Compresión de información multimedia

¿Por qué comprimir la información MM?

Las representaciones digitales de datos multimedia (voz, música, secuencias de vídeo...) requieren, por unidad de tiempo, un gran número de bits, ya sea para su **almacenamiento** o su **transmisión**.

Compresión de información multimedia (II)

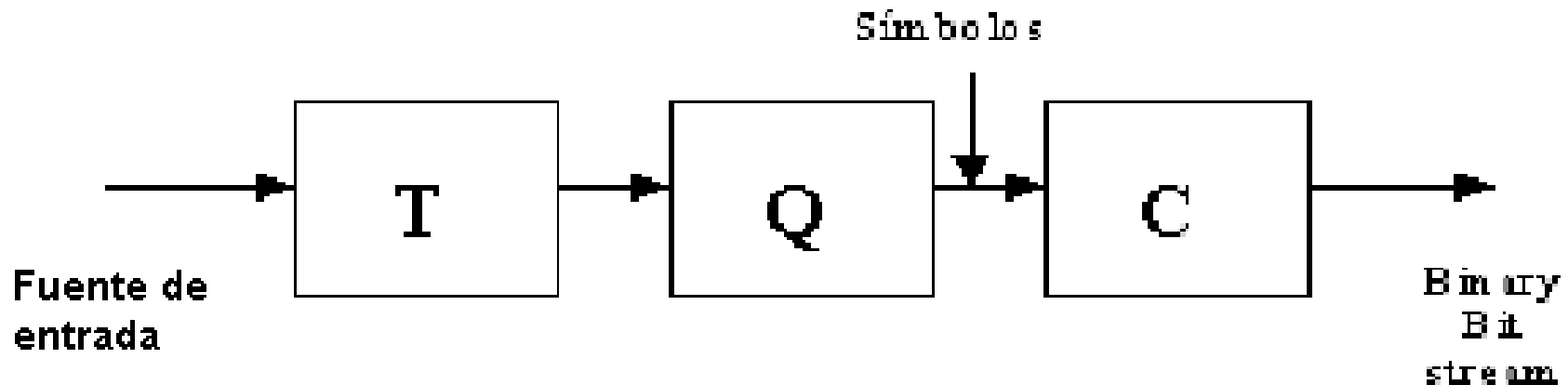
Definición:

- ★ La compresión de datos es la representación de una fuente en forma digital, con tan pocos bits como sea posible, mientras se mantiene una pérdida aceptable en fidelidad.
- ★ La fuente puede ser datos, voz, música, imágenes, etc., o cualquier señal que necesite ser almacenada o transmitida.
- ★ El término compresión se refiere tanto a:
 - la compresión sin pérdidas (*lossless*): la fuente se preserva perfectamente durante la representación.
 - la compresión con pérdidas (*lossy*): no se preserva perfectamente.

Compresión de información multimedia (III)

- Conceptos básicos (*Information Theory*)
 - Diagrama de bloques de un proceso de compresión:

Transformación → Cuantificación → Codificación



- Simétrico vs asimétrico
- Pueden existir (errores) pérdidas
 - Cuantización y submuestreo antes de la codificación
- Redundancia en la información

Compresión de información multimedia (IV)

- Técnicas

- Sin pérdidas (*lossless*)

- RLE
 - Huffman
 - *Delta Encoding*
 - Lempel-Ziv-Welch

- Con pérdidas (*lossy*)

- Depende del media y su percepción

Compresión sin pérdidas

- RLE (*Run-Length Encoding*)

57 57 57 57 57 57 110 110 110 132 55 200 200 200 200 200 200 200 200 200

{5,57} {2,110} {0,132}{0,55} {9, 200}

- Huffman

Valor de pixel	Frecuencia de aparición	Código (binario)
54	132	0
22	84	10
112	57	11

112 112 112 54 54 54 22 22

Compresión sin pérdidas

- Delta

original data stream: 17 19 24 24 24 21 15 10 89 95 96 96 96 95 94 94 95 93 90 87 86 86 ...

move
delta
delta
delta
...

delta encoded: 17 2 5 0 0 -3 -6 -5 79 6 1 0 0 -1 -1 0 1 -2 -3 -3 -1 0 ...

- Lempel-Ziv-Welch

Example Code Table

code number	translation
0000	0
0001	1
⋮	⋮
0254	254
0255	255
0256	145 201 4
0257	243 245
⋮	⋮
4095	xxx xxx xxx

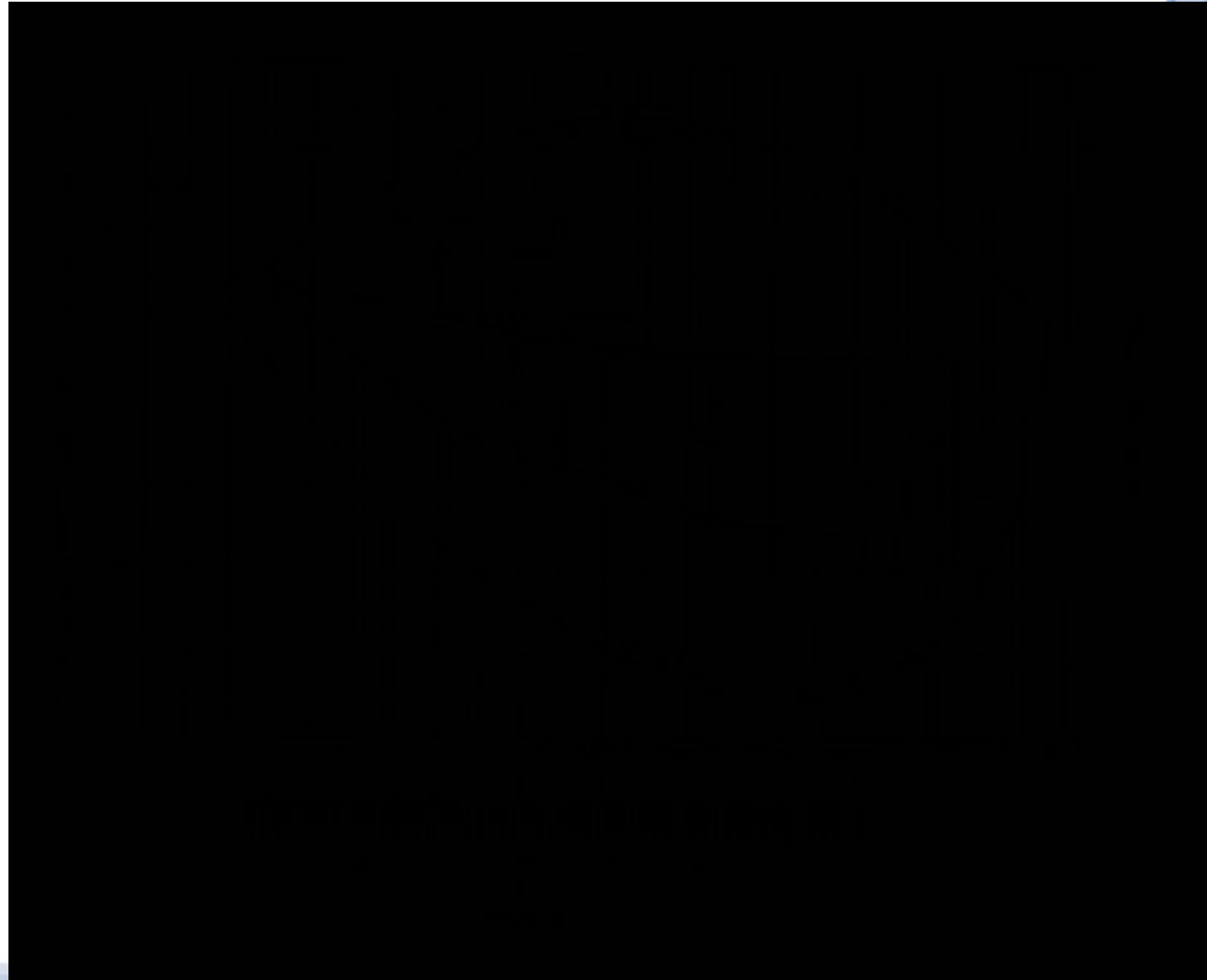
123 145 201 4 119 89 243 245 59 11 206 145 201 4 243 245 ...

123 256 119 89 257 59 11 206 256 257 ...

Diagram illustrating the Lempel-Ziv-Welch (LZW) encoding process. The original data stream is shown above the encoded stream. The encoded stream consists of indices (123, 256, 119, 89, 257, 59, 11, 206, 256, 257, ...) which correspond to the code numbers in the Example Code Table. The diagram shows how the original data is broken down into tokens (145 201 4, 243 245, 145 201 4, 243 245) and how these tokens are mapped to the encoded stream using the code table.

Compresión de audio con pérdidas

- Respuesta del oído humano
 - 2 .. 4 kHz
 - 20 .. 20KHz



Compresión de audio con pérdidas

- Eliminar o codificar con menos detalle las componentes de sonido que son menos perceptibles (codificación perceptual):
 - Bandas de frecuencias muy bajas o muy altas
 - Sonidos enmascarados por otros
 - Frecuencias bajas que siguen inmediatamente a frec. bajas.
 - Redundancias de señales estéreo.

- Ejemplo: compresión MP3

Sonido estéreo con calidad CD (16 bits y 44.1kHz, por canal)
 \cong 1400 kbps (sin comprimir)



MP3: 128..112 kbps (factor 12)

Caso de estudio: plataforma NDS

- Ejemplos
 - *MaxMod Demo*
 - *DSSpeech*
 - *AndroDaft / DSDaft*
- Hardware de sonido
- SDK: Libnds + MaxMod



Caso de estudio: Ejemplos

- MaxMod
 - Vídeo: demo
 - MaxMod Demo



DS Demonstration

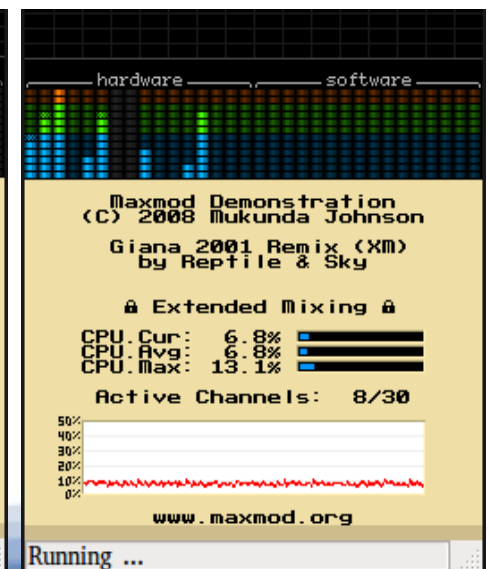
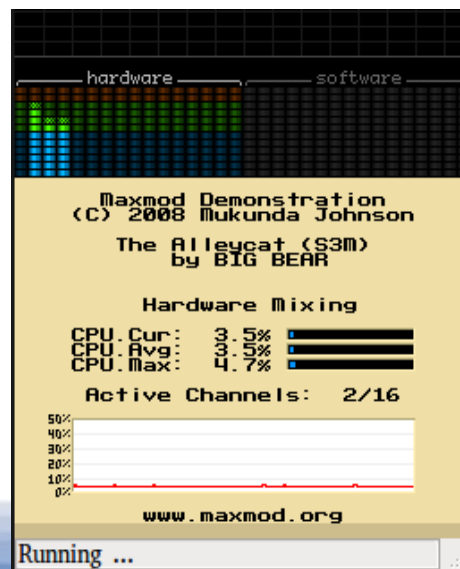
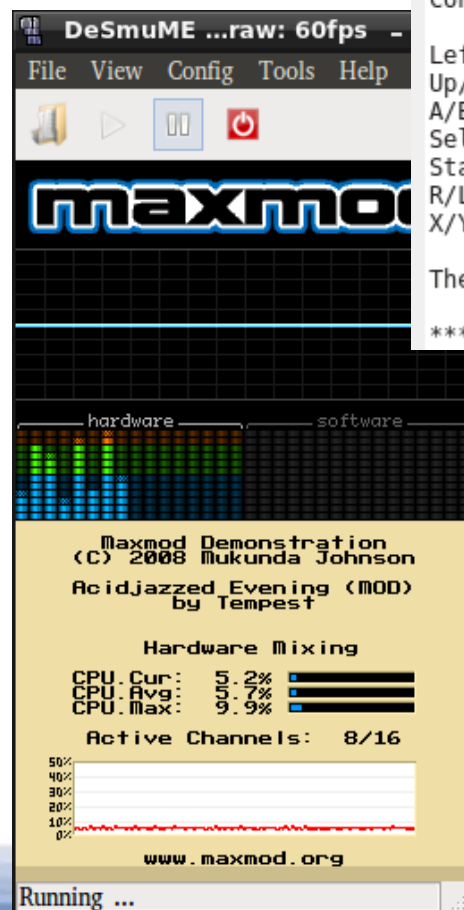
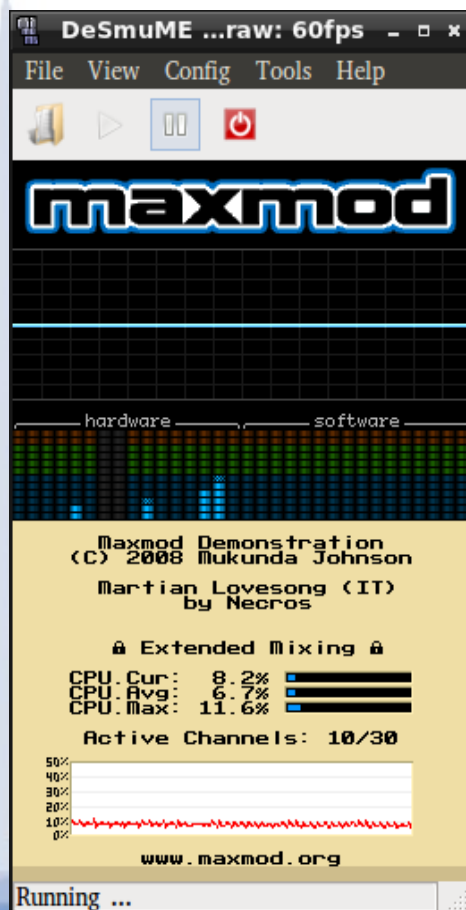
Copyright (c) 2008, Mukunda Johnson (mukunda@maxmod.org)

Welcome to the Maxmod Demo! Please use hardware to test the program; emulators do not reproduce the sound properly.

Controls:

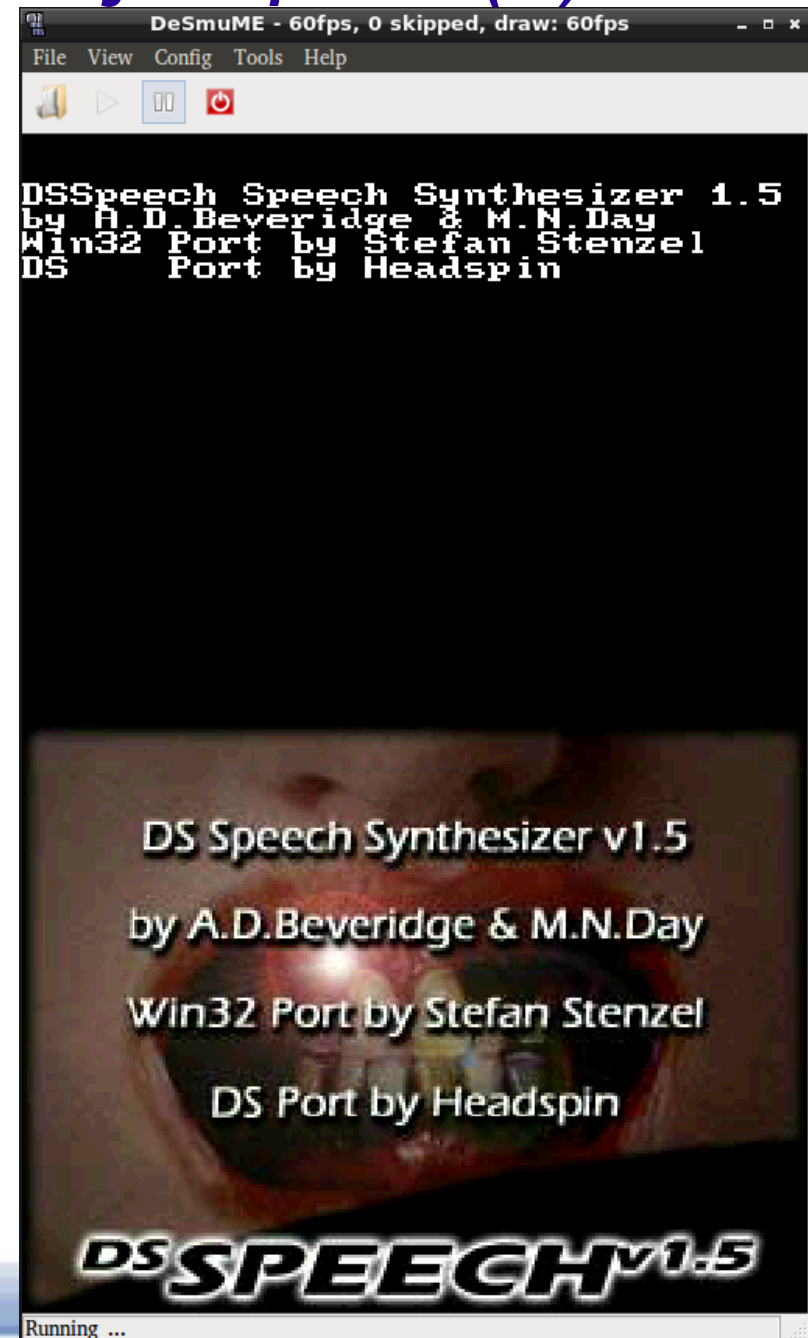
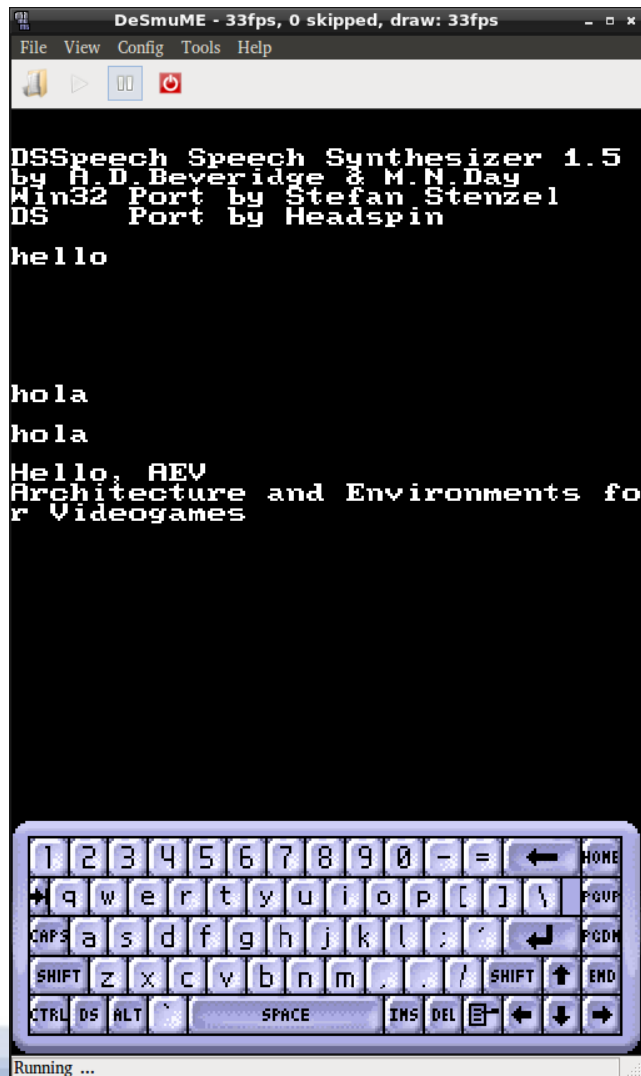
Left/Right: Change song
 Up/Down : Change audio mode (some songs have this locked)
 A/B : Play sound effect
 Select : Play jingle
 Start : Pause/Resume
 R/L : Increase/Decrease tempo
 X/Y : Increase/Decrease pitch (fun!)

The songs used in the demo are copyrighted by their respective authors.



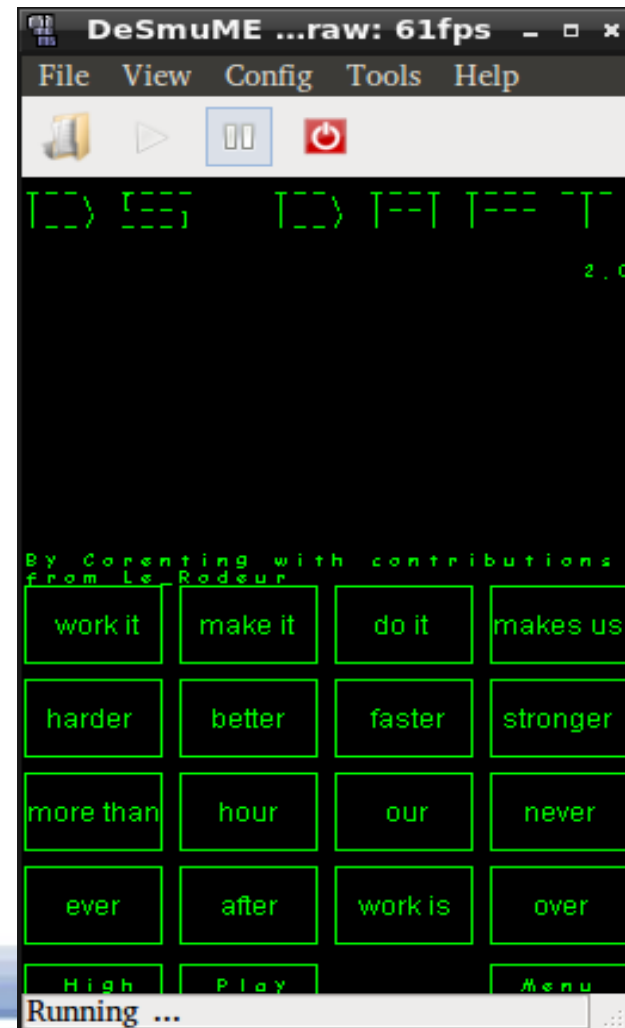
Caso de estudio: Ejemplos (II)

- DSSpeech
 - NEO Coding Contest '05



Caso de estudio: Ejemplos (y III)

- AndroDaft / DSDaft ← corenting
 - AndroDaft: Streaming ← Maxmod + NitroFS
 - DsDaft ←- NightFox Lib.



Caso de estudio: hardware de audio

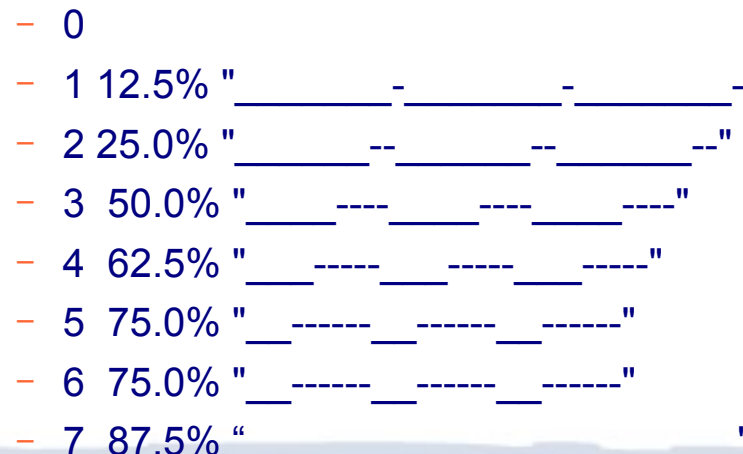
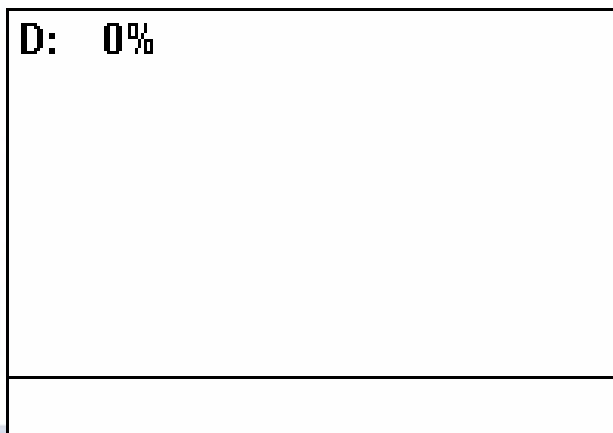
- Arquitectura de la NDS
 - Salida
 - 16 canales *hardware* independientes
 - Frecuencia de muestreo, tamaño de muestra y número de muestras
 - *enabled or not, looping mode/one shot, volume, etc.*
 - 2 canales (estéreo) y balance (*panning*)
 - Màx frec. de muestreo 32 KHz
 - Remuestrear a 32768 Hz por Hw. (*nearest-neighbor algorithm*)
 - Entrada
 - Micrófono
 - 1 canal
 - Modos de 8 y 12 bits PCM

Caso de estudio: hardware de audio (II)

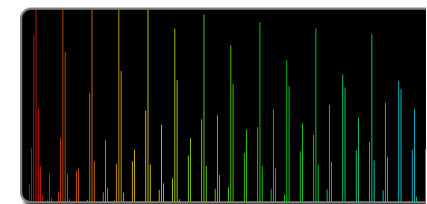
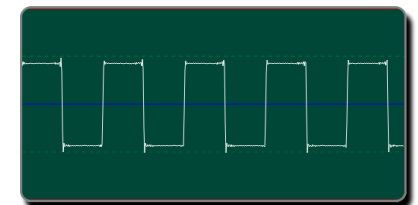
- Arquitectura NDS
 - ARM9 (*libfat / nitrofs*) + ARM7 (*audio hardware*)
- Formato nativo
 - *raw format*
 - 8 o 16 bit, PCM / ADPCM.
 - Sin comprimir y sin cabecera
 - ¿Soporte para otros formatos (wav, mp3, OggVorbis, ...)?
 - Conversión previa
 - Audacity, sox tool, wav2gba, ...,
 `${DEVKITARM}/bin/mmutil` (*Maxmod Utility*)
 - vs “en el momento”
 - Helix, ...

Caso de estudio: hardware de audio (III)

- *Programmable Sound Generator (PSG)*
 - *List of sound chips* https://en.wikipedia.org/wiki/List_of_sound_chips.
- Wave Duty o Ciclo de trabajo
 - En la NDS, el ciclo de trabajo se define como:
 - 8 transiciones: *HIGH* or *LOW* samples
 - The sound frequency is 1/8th of the selected sample rate
 - The duty cycle always starts at the begin of the *LOW* period when the sound gets (re-)started.



Duty cycle control:



Caso de estudio: SDK de audio

- Tipos → *libnds + maxmod*
 - Sonidos
 - Síntesis de sonidos simples
 - Múltiples instancias sonando a la vez (*mixed* + FX).
 - Suelen cargarse completamente en memoria (y descargarse)
 - Música
 - Sólo una instancia en reproducción
 - Si no caben en RAM (DS 4 MB) ... *stream*
 - Bloques (*chunks*)
 - Leer + reproducir + descargar
 - *ring buffer* o *double-buffering audio*
 - Latencia vs compensación (*skip tradeoff*)

Caso de estudio: SDK de audio (II)

- SDK = libnds + Maxmod
 - *libnds: Simple Sound Engine (SSE)*
 - “Simple Sound System” → *system.h*
 - Ops. Básicas
 - Volumen, balance, ...
 - Micrófono
 - Generador de ruido
 - Acceso al *Programmable Sound Generator (PSG)*
 - Maxmod
 - *Reproduce forma de ondas (sample-playing)*
 - *Reproduce ficheros estructurados (mod-playing)*
 - *Gestiona el modo streaming (buffers).*

Caso de estudio: SDK de audio (II)

- API libnds: salida (síntesis y reproducción)
 - int soundPlayNoise (u16 freq, u8 volume, u8 pan)
 - int soundPlayPSG (**DutyCycle** cycle, u16 freq,
u8 volume, u8 pan)
 - int soundPlaySample (const void * data,
SoundFormat format,
u32 dataSize, u16 freq,
u8 volume, u8 pan,
bool oop, u16 loopPoint)
 - *SoundFormat; 8 o 16 bit PCM, PSG, ADPCM.*

Caso de estudio: SDK de audio (III)

- API *libnds*: entrada

- *int soundMicRecord* (*void ** *buffer*,
 u32 *bufferLength*,
 MicFormat *format*,
 int *freq*,
 MicCallback *callback*)
- Modos ← *MicFormat*
 - 8-bit PCM
 - 12-bit PCM

Caso de estudio: SDK de audio (IV)

- *API libnds: decompress.h*
- *Tipos de descompresión disponibles*
 - *enum DecompressType {
 LZ77Vram,
 HUFF,
 RLE,
 RLEVram
}*

Caso de estudio: SDK de audio (V)

- API *maxmod* (*DS Programming Guide*)

- Proyecto

- Directorio *AUDIO* := *audio*

- *blaster.wav*, *phaser.wav*, *bonk.wav*

- *title.mod*, *ingame.it*, *credits.s3m*, *onemoresong.xm*

- Makefile

- LIBS* := *-lmm9 -lnds9*

- ...

- soundbank.bin*: *\$(AUDIOFILES)*

- @mmutil* *\$^ -osoundbank.bin -hsoundbank.h -d*

- Código

- #include <maxmod9.h>*

- #include "soundbank.h"*



```
#define SFX_BLAZER      0
#define SFX_PHASER      1
#define SFX_BONK        2
#define MOD_TITLE       0
#define MOD_INGAME      1
#define MOD_CREDITS     2
#define MOD_ONEMORESONG 3
#define MSL_NSONGS      4
#define MSL_NSAMPS      156
#define MSL_BANKSIZE    160
```

- Operaciones sobre
efectos (SFX) o música (MOD)

Caso de estudio: SDK de audio (VI)

- Efectos

- Carga

- `mmLoadEffect(SFX_BLASTER);`

- Reproducir

- `mmEffect(SFX_BLASTER);`

ó

```
mm_sound_effect sound;
```

```
sound.id      = SFX_BLASTER; // sample ID (make sure it is loaded)
```

```
sound.rate    = 0x400/2;    // playback rate, 1024 = original sound
```

```
sound.handle  = 0;          // 0 = allocate new handle
```

```
sound.volume  = 200;        // 200/255 volume level
```

```
sound.panning = 128;        // centered panning
```

```
mmEffectEx( &sound );
```

Caso de estudio: SDK de audio (VII)

- Efectos

- *Modificar en tiempo de ejecución*

- *// Change pitch to +1 octave
mmEffectRate(mysound, 1024*2);*
 - *// Change volume to half level (128/255)
mmEffectVolume(mysound, 128);*
 - *// Change panning to far-right
mmEffectPanning(mysound, 255);*

- *Liberar recursos*

- *mmUnloadEffect(SFX_BLASTER);*

Caso de estudio: SDK de audio (VIII)

- Música

- Carga

- `mmLoad(MOD_TITLE);`

- Reproducir

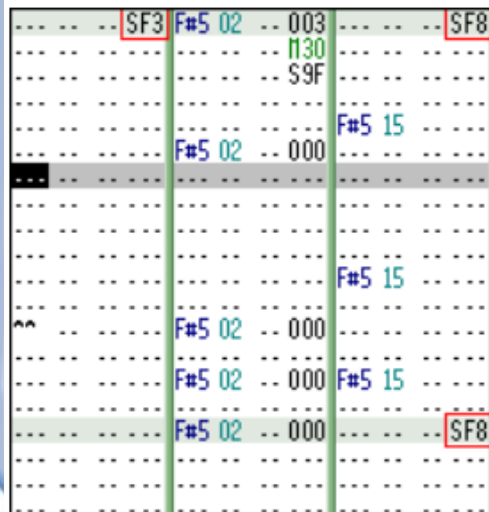
- `mmStart(MOD_TITLE, MM_PLAY_LOOP); // MM_PLAY_ONCE`

- Liberar recursos

- `mmUnload(MOD_TITLE);`

- Eventos → sincronizar (Sfx en S3M/IT ó Efx en MOD/SM)

- `mmSetEventHandler(myEventHandler);`



```
mm_word myEventHandler( mm_word msg, mm_word param )
{
    switch( msg )
    {
        case MMCB_SONGMESSAGE:
            // Process song message
            break;
        case MMCB_SONGFINISHED:
            // A song has finished playing
    }
}
```

Práctica NDS

- Formas de onda
 - Comprobar sobre la plataforma los conceptos básicos
 - Síntesis de sonido (PSG)
 - Reproducción de formas de onda
- Formatos de audio estructurado
 - Sincronización de acciones con la banda sonora
- ¿NFlib?

Caso de estudio: plataforma 3DS

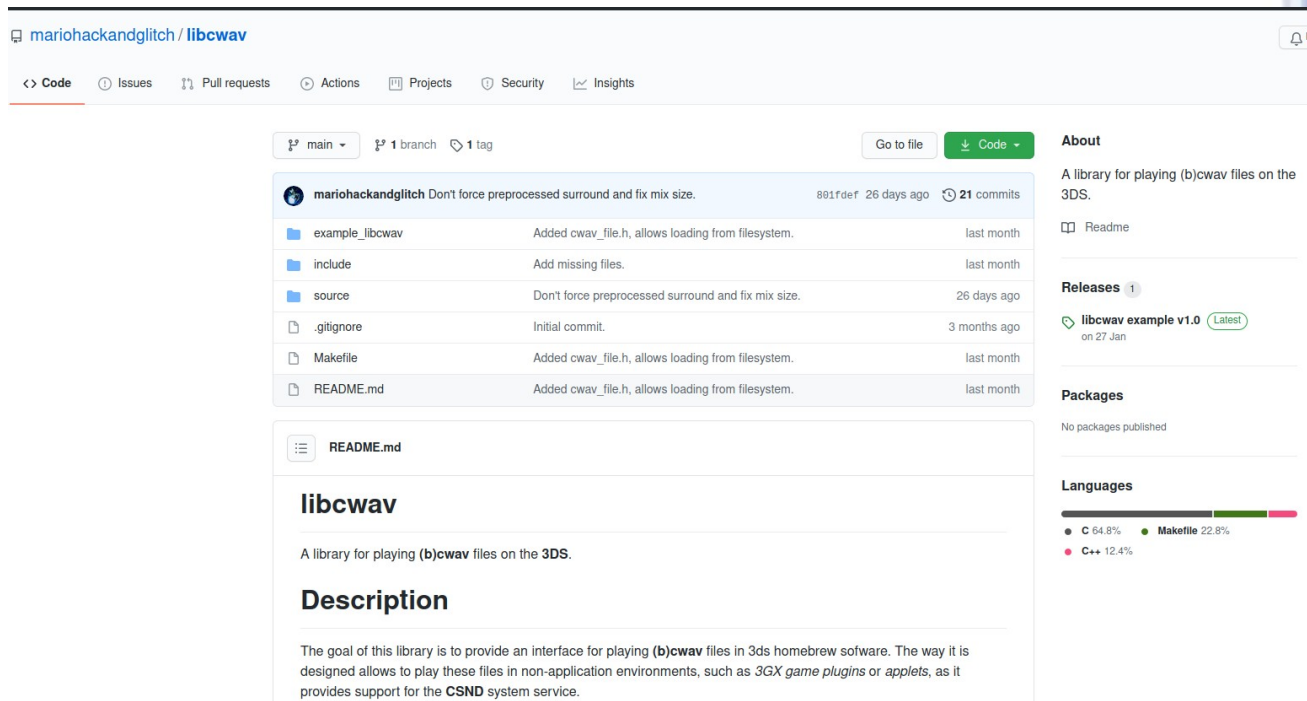
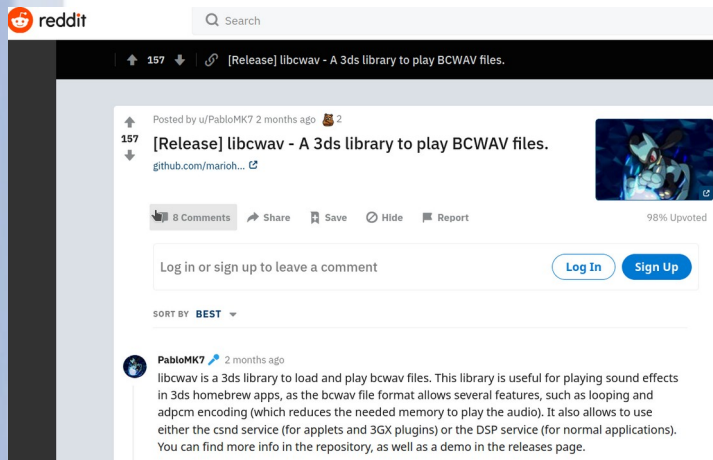
- Hardware de sonido
 - CSND
 - ~ PSG de la NDS
 - Generador señales simples y ruido blanco
 - Acceso al micrófono
 - Reproducción de muestras cortas < 3 seg.
 - DSP
 - Coprocesador para procesamiento de audio
 - *Streaming*
 - Formato BCWAV
- Emulador: Citra
- SDK: libctru y ejemplos

Hardware de sonido: formato BCWAV

- *Binary CTR Wave files (BCWAV).*
 - *The structure is very similar to Microsoft's Wave file.*
 - Microsoft's WAV structure is RIFF Header which defines the data inside which is WAVE, then the media player expects a "fmt " chunk and a "data" chunk.
 - Nintendo's format uses a CWAV header (no need for a general structure for media, only wave), which points to an INFO struct (the equivalent to fmt) and a DATA struct (the equivalent to data).

libcwav ← PabloMK7


- *BCWAV stands for binary CTR wav, and that notation is shared between all of the NW4C file types (bcstm, bcmdl, etc). Using binary always seems redundant, so it's usually omitted. Also bcwav files are internally referred as CWAV, since the name.*



cwavtool ← PabloMK7

main - 1 branch 1 tag

Go to file Code -

 **mariohackandglitch** Update README.md

12736bb 2 days ago 78 commits

buildtools @ 4524b3a	Update buildtools.	3 years ago
source	Update README and LICENSE	2 days ago
.gitignore	Initial cwavtool release.	2 days ago
.gitmodules	Make buildtools shallow.	3 years ago
LICENSE	Update README and LICENSE	2 days ago
Makefile	Initial cwavtool release.	2 days ago
README.md	Update README.md	2 days ago

README.md

cwavtool

A tool for converting **WAV/OGG** files to **(B)CWAV** files.

Usage

This tool can convert to any encoding supported by the **(B)CWAV** file format (pcm16 by default). Optionally, a loop point can be specified:

```
> cwavtool.exe <args>
Available arguments:
-i/--input: WAV/OGG input file.
-o/--output: CWAV output file.
-e/--encoding: Optional. Encoding of the created CWAV (pcm8/pcm16/imaadpcm/dspadpcm).
-ls/--loopstartframe: Optional. Sample to return to when looping.
-le/--loopendframe: Optional. Sample to loop at or "end".
```

Credits & License

- This project is a modified work of [Steviece10's bannertool](#) and is licensed under the [MIT License](#).
- This project uses [David Bryant's adpcm-xq](#) for IMA-ADPCM encoding ([License](#)).
- This project uses [Jack Andersen's gc-dspadpcm-encode](#) for DSP-ADPCM encoding ([License](#)).
- Thanks to [JoanCoCo](#) for the compiled MAC binary.

About


Command line tool to convert WAV/OGG files to (B)CWAV files.

[Readme](#)

[View license](#)

Releases

1


 **Initial Release** Latest
2 days ago

Packages

No packages published

Contributors

7

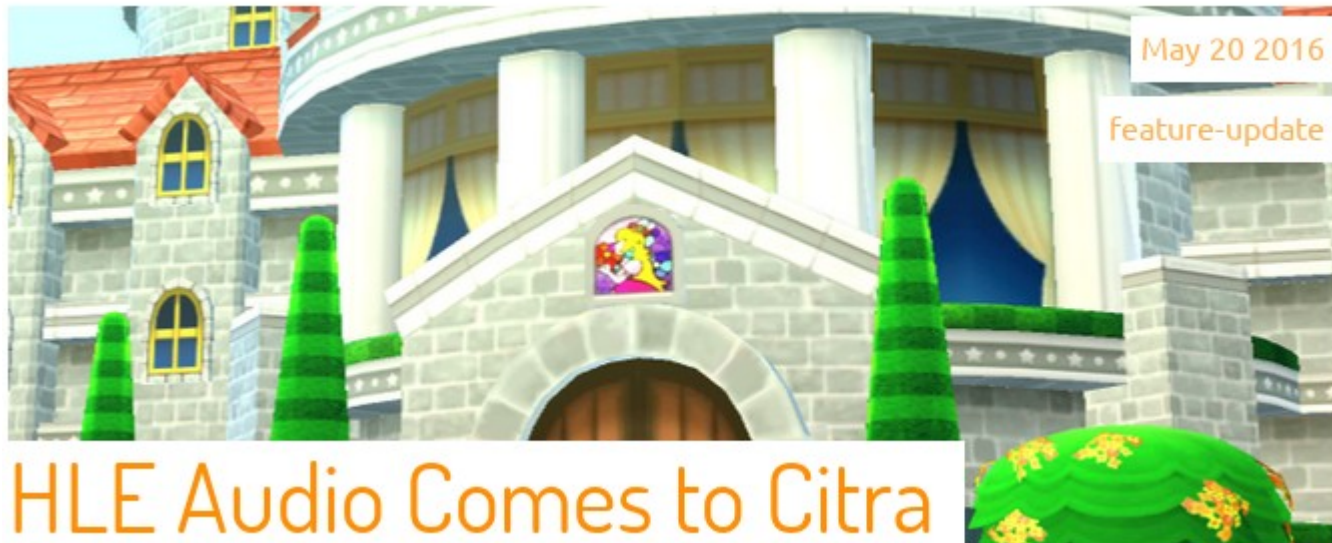


Languages

C 86.5% C++ 13.2%
Makefile 0.3%

Caso de estudio: plataforma 3DS

- Emulador Citra



Accurate Audio Emulation Has
Arrived

Caso de estudio: plataforma 3DS

- Emulador Citra

- Audio processing and output is done by a specialised coprocessor. These kinds of coprocessors are called Digital Signal Processors (DSPs).
- Games that run on the 3DS need to communicate with the DSP in order to play audio. They do this by two ways: via the the dsp::DSP service and via a shared memory region. The dsp::DSP service provides service calls for initialization of the DSP hardware including **firmware** upload. The shared memory region is used for communication between the game on the CPU and the firmware on the DSP.
- In order to emulate audio, Citra must emulate the dsp::DSP service and also understand the layout of the DSP shared memory region. One must understand what writing to various addresses in the shared memory region does. One must also understand what happens between data being fed to the DSP firmware and audio coming out of the speakers.



SDK: libctru

- Ejemplos

- *mic (NDS ~ 3DS)*

- Usa CSND para grabar desde el micrófono y reproducir lo grabado.

- *streaming (NDS ~ 3DS)*

- Comprobar sobre la plataforma los conceptos básicos
 - Síntesis de sonido (*Programmable Sound Generator*) → CSND
 - Reproducción de formas de onda → NDSP

- *filters (3DS !m)*

- Procesado de la señal de audio → NDSP

- *Opus-decoding (3DS !m)*

- Reproducir un fichero OPUS ← libopus (PORTLIBS).

SDK: libctru

- **csnd.h** <\$DEVKITPRO/libctru/include/3ds/ndsp/csnd.h>

CSND service. Usage of this service is deprecated in favor of NDSP.

```
/// Maximum number of CSND channels.
#define CSND_NUM_CHANNELS 32

/// Creates a CSND timer value from a sample rate.
#define CSND_TIMER(n) (0x3FEC3FC / ((u32)(n)))

...

/// CSND encodings.
enum {
    CSND_ENCODING_PCM8 = 0, ///< PCM8
    ...
    CSND_ENCODING_PSG,    ///< PSG (Similar to DS?)
};

/// Sound flags.
enum {
    SOUND_LINEAR_INTERP = BIT(6),    ///< Linear interpolation.
    ..
    SOUND_FORMAT_PSG =
        SOUND_FORMAT(CSND_ENCODING_PSG),    ///< PSG
    SOUND_ENABLE = BIT(14),            ///< Enable sound.
};

/// Capture modes.
enum {
    CAPTURE_REPEAT = 0,    ///< Repeat capture.
    ...
    CAPTURE_ENABLE = BIT(15),    ///< Enable capture.
};

...
```

```
/// Duty cycles for a PSG channel.
typedef enum {
    DutyCycle_0 = 7, ///< 0.0% duty cycle
    ...DutyCycle_87 = 6    ///< 87.5% duty cycle
} CSND_DutyCycle;

.....

CSND_CapInfo;

// See here regarding CSND shared-mem commands, etc:
// http://3dbrew.org/wiki/CSND\_Shared\_Memory

extern vu32* csndSharedMem;    ///< CSND shared memory.
extern u32 csndSharedMemSize;    ///< CSND shared memory size.
extern u32 csndChannels;    ///< Bitmask of channels that are allowed for usage.

...
Result CSND_AcquireCapUnit(u32* capUnit);
...
...

void CSND_SetDuty(u32 channel, CSND_DutyCycle duty);

// Sets CSND's noise channel registers.
void CSND_SetChnRegsNoise(u32 flags, u32 chnVolumes, u32 capVolumes);

// Plays a sound.
Result csndPlaySound(int chn, u32 flags, u32 sampleRate,
                    float vol, float pan, void* data0, void* data1, u32 size);

...
```

SDK: libctru

- **ndsp.h** <\$DEVKITPRO/libctru/include/3ds/ndsp/ndsp.h>

// Interface for Nintendo's default DSP component.

/// Sound output modes.

```
typedef enum {  
    NDSP_OUTPUT_MONO    = 0, ///  
    NDSP_OUTPUT_STEREO  = 1, ///  
    NDSP_OUTPUT_SURROUND = 2, ///  
} ndspOutputMode;
```

/// Wave buffer struct.

```
struct tag_ndspWaveBuf {  
    union {  
        s8*    data_pcm8; ///  
        s16*   data_pcm16; ///  
        u8*    data_adpcm; ///  
    };  
    const void* data_vaddr; ///  
    u32 nsamples; ///  
    ndspAdpcmData* adpcm_data; ///  
    u32 offset; ///  
    bool looping; ///  
    u8 status; ///  
    u16 sequence_id; ///  
    ndspWaveBuf* next; ///  
};
```

/// Wave buffer type.

```
typedef struct tag_ndspWaveBuf ndspWaveBuf;
```

Result ndspInit(void);

void ndspExit(void);

/**

* @brief Sets up the DSP component.
* @param binary DSP binary to load.
* @param size Size of the DSP binary.
* @param progMask Program RAM block mask to load the binary to.
* @param dataMask Data RAM block mask to load the binary to.
*/

void ndspUseComponent(const void* binary, u32 size, u16 progMask, u16 dataMask);

void ndspSetMasterVol(float volume);

void ndspSetCapture(ndspWaveBuf* capture);

void ndspSurroundSetDepth(u16 depth);

void ndspSurroundSetPos(ndspSpeakerPos pos);

void ndspSurroundSetRearRatio(u16 ratio);

///Auxiliary output

void ndspAuxSetEnable(int id, bool enable);

void ndspAuxSetVolume(int id, float volume);

void ndspAuxSetCallback(int id, ndspAuxCallback callback, void* data);

...

SDK: libctru

- Ejemplo: *mic*

```
u32 micbuf_size = 0x30000; u32 micbuf_pos = 0; u8* micbuf = memalign(0x1000, micbuf_size);
printf("Initializing CSND...\n");
if(R_FAILED(csndInit())) {
    ...
} else printf("CSND initialized.\n");
printf("Initializing MIC...\n");
if(R_FAILED(micInit(micbuf, micbuf_size)))
    ...
if(kDown & KEY_A) {
    audiobuf_pos = 0; micbuf_pos = 0; printf("Stopping audio playback...\n");
    CSND_SetPlayState(0x8, 0);
    if(R_FAILED(CSND_UpdateInfo(0))) printf("Failed to stop audio playback.\n");

    printf("Starting sampling...\n");
    if(R_SUCCEEDED(MICU_StartSampling(MICU_ENCODING_PCM16_SIGNED, MICU_SAMPLE_RATE_16360, 0, micbuf_datasize, true)))
        printf("Now recording.\n");
    ...
    if(hidKeysUp() & KEY_A) { printf("Stoping sampling...\n");
    if(R_FAILED(MICU_StopSampling())) printf("Failed to stop sampling.\n");

    printf("Starting audio playback...\n");
    if(R_SUCCEEDED(GSPGPU_FlushDataCache(audiobuf, audiobuf_pos))) &&
        R_SUCCEEDED(csndPlaySound(0x8, SOUND_ONE_SHOT | SOUND_FORMAT_16BIT, 16360, 1.0, 0.0, (u32*)audiobuf, NULL, audiobuf_pos))
        printf("Now playing.\n");
    ...
}
micExit();
...
csndExit();
```

SDK: libctru

- Ejemplo: *streaming*

```
#define SAMPLERATE 22050
#define SAMPLESPERBUF (SAMPLERATE / 30)
#define BYTESPERSAMPLE 4

void fill_buffer(void *audioBuffer, size_t offset, size_t size, int frequency) {
    u32 *dest = (u32*)audioBuffer;
    for (int i=0; i<size; i++) {
        s16 sample = INT16_MAX * sin(frequency*(2*M_PI)*(offset+i)/SAMPLERATE);
        dest[i] = (sample<<16) | (sample & 0xffff);
    }
    DSP_FlushDataCache(audioBuffer, size);
}

int main(int argc, char **argv) {
    PrintConsole topScreen;
    ndspWaveBuf waveBuf[2];
    gfxInitDefault();
    ...
    printf("libctru streaming audio\n");
    u32 *audioBuffer = (u32*)linearAlloc(SAMPLESPERBUF*BYTESPERSAMPLE*2);
    bool fillBlock = false;
    ndspInit();
    ndspSetOutputMode(NDSP_OUTPUT_STEREO);
    ndspChnSetInterp(0, NDSP_INTERP_LINEAR);
    ndspChnSetRate(0, SAMPLERATE);
    ndspChnSetFormat(0, NDSP_FORMAT_STEREO_PCM16);
    ...
    ndspChnSetMix(0, mix);

    int notefreq[] = { 262, 294, 339, 349, 392, 440, 494, 440, 392, 349, 339, 294};
    int note = 4;
```

```
memset(waveBuf, 0, sizeof(waveBuf));

waveBuf[0].data_vaddr = &audioBuffer[0]; waveBuf[0].nsamples =
SAMPLESPERBUF; waveBuf[1].data_vaddr =
    &audioBuffer[SAMPLESPERBUF];
waveBuf[1].nsamples = SAMPLESPERBUF;
size_t stream_offset = 0;
fill_buffer(audioBuffer, stream_offset, SAMPLESPERBUF * 2, notefreq[note]);
stream_offset += SAMPLESPERBUF;
ndspChnWaveBufAdd(0, &waveBuf[0]);
ndspChnWaveBufAdd(0, &waveBuf[1]);

printf("Press up/down to change tone\n");
while(aptMainLoop()) {
    ...
    hidScanInput();
    ...
    if (note<0) note = sizeof(notefreq)/sizeof(notefreq[0])-1;
}
if (kDown & KEY_UP) {
    note += 1; if (note > (sizeof(notefreq)/sizeof(notefreq[0])-1)) note = 0;
}

if (waveBuf[fillBlock].status == NDSP_WBUF_DONE) {
    fill_buffer(waveBuf[fillBlock].data_pcm16, stream_offset,
        waveBuf[fillBlock].nsamples, notefreq[note]);
    ndspChnWaveBufAdd(0, &waveBuf[fillBlock]);
    stream_offset += waveBuf[fillBlock].nsamples;
    fillBlock = !fillBlock;
}
}
ndspExit();
...
}
```

SDK: libctru

- Ejemplo: *filters*

```
ndspInit();
ndspSetOutputMode(NDSP_OUTPUT_STEREO);
ndspChnSetInterp(0, NDSP_INTERP_LINEAR);
ndspChnSetRate(0, SAMPLERATE);
ndspChnSetFormat(0, NDSP_FORMAT_STEREO_PCM16);
...
ndspWaveBuf waveBuf[2];
memset(waveBuf, 0, sizeof(waveBuf));
waveBuf[0].data_vaddr = &audioBuffer[0];
waveBuf[0].nsamples = SAMPLESPERBUF;
waveBuf[1].data_vaddr = &audioBuffer[SAMPLESPERBUF];
waveBuf[1].nsamples = SAMPLESPERBUF;

size_t stream_offset = 0;

fill_buffer(audioBuffer, stream_offset, SAMPLESPERBUF * 2, notefreq[note]);
stream_offset += SAMPLESPERBUF;

ndspChnWaveBufAdd(0, &waveBuf[0]);
ndspChnWaveBufAdd(0, &waveBuf[1]);
..
```

```
..
if (kDown & KEY_LEFT) {
    filter--;
    ...
} else if (kDown & KEY_RIGHT) {
    Filter++;
    ---
}
switch (filter) {
    ...
    ndspChnlirBiquadSetEnable(0, false);
    ...
    ndspChnlirBiquadSetParamsLowPassFilter(0, 1760.f, 0.707f);
    ...
    ndspChnlirBiquadSetParamsHighPassFilter(0, 1760.f, 0.707f);
    ...
    ndspChnlirBiquadSetParamsBandPassFilter(0, 1760.f, 0.707f);
    ...
    ndspChnlirBiquadSetParamsNotchFilter(0, 1760.f, 0.707f);
    ...
    ndspChnlirBiquadSetParamsPeakingEqualizer(0, 1760.f, 0.707f, 3.0f);
}
if (waveBuf[fillBlock].status == NDSP_WBUF_DONE) {
    fill_buffer(waveBuf[fillBlock].data_pcm16, stream_offset,
    waveBuf[fillBlock].nsamples, notefreq[note]);
    ndspChnWaveBufAdd(0, &waveBuf[fillBlock]);
    stream_offset += waveBuf[fillBlock].nsamples;
    fillBlock = !fillBlock;
}.
..
ndspExit();
```


SDK: libopus (PORTLIBS)

• Ejemplo: opus-decoding

```
...
OggOpusFile *opusFile = op_open_file(PATH, &error);
...
if(!audiolInit()) { // Attempt audiolInit
...
// Set the ndsp sound frame callback which signals our audioThread
ndspSetCallback(audioCallback, NULL);

// Spawn audio thread
...
// Start the thread, passing our opusFile as an argument.
const Thread threadId = threadCreate(audioThread, opusFile,
                                     THREAD_STACK_SZ, priority,
                                     THREAD_AFFINITY, false);
...
while(aptMainLoop()) { // Standard main loop
    ...
    // Your code goes here
    u32 kDown = hidKeysDown();
    ...
}
// Signal audio thread to quit
s_quit = true;
LightEvent_Signal(&s_event);
// Free the audio thread
threadJoin(threadId, UINT64_MAX);
threadFree(threadId);
// Cleanup audio things and de-init platform features
audioExit();
ndspExit();
op_free(opusFile);
```

```
// NDSP audio frame callback

// This signals the audioThread to decode more things
// once NDSP has played a sound frame, meaning that there should be
// one or more available waveBufs to fill with more data.
void audioCallback(void *const nul_) {
    ...
    LightEvent_Signal(&s_event);
}

// Audio thread
// This handles calling the decoder function to fill NDSP buffers as necessary
void audioThread(void *const opusFile_) {
    OggOpusFile *const opusFile = (OggOpusFile *)opusFile_;
    while(!s_quit) { // Whilst the quit flag is unset, search our waveBufs and
        //fill any that aren't currently queued for playback
        // (i.e, those that are 'done')
        for(size_t i = 0; i < ARRAY_SIZE(s_waveBufs); ++i) {
            if(s_waveBufs[i].status != NDSP_WBUF_DONE) {
                continue;
            }
            if(!fillBuffer(opusFile, &s_waveBufs[i])) { // Playback complete
                return;
            }
        }
        // Wait for a signal that we're needed again before continuing,
        // so that we can yield to other things that want to run
        // (Note that the 3DS uses cooperative threading)
        LightEvent_Wait(&s_event);
    }
}

// Main audio decoding logic
// This function pulls and decodes audio samples from opusFile_ to fill waveBuf
```

SDK: libopus (PORTLIBS)

• Ejemplo: opus-decoding

```
// Main audio decoding logic
// This function pulls and decodes audio samples from opusFile_ to fill waveBuf_
bool fillBuffer(OggOpusFile *opusFile_, ndspWaveBuf *waveBuf_) {
    ...
    // Decode samples until our waveBuf is full
    int totalSamples = 0;
    while(totalSamples < SAMPLES_PER_BUF) {
        int16_t *buffer = waveBuf_->data_pcm16 + (totalSamples * CHANNELS_PER_SAMPLE);
        const size_t bufferSize = (SAMPLES_PER_BUF - totalSamples) * CHANNELS_PER_SAMPLE;

        // Decode bufferSize samples from opusFile_ into buffer,
        // storing the number of samples that were decoded (or error)
        const int samples = op_read_stereo(opusFile_, buffer, bufferSize);
        if(samples <= 0) {
            if(samples == 0) break; // No error here
            printf("op_read_stereo: error %d (%s)", samples, opusStrError(samples));
            break;
        }
        totalSamples += samples;
    }

    // If no samples were read in the last decode cycle, we're done
    if(totalSamples == 0) {
        printf("Playback complete, press Start to exit\n");
        return false;
    }

    // Pass samples to NDSP
    waveBuf_>nsamples = totalSamples;
    ndspChnWaveBufAdd(0, waveBuf_);
    DSP_FlushDataCache(waveBuf_>data_pcm16, totalSamples * CHANNELS_PER_SAMPLE * sizeof(int16_t));

    ...
    return true;
}
```

s
at there should be
a.

P buffers as necessary

usFile_;
ch our waveBufs and
d for playback

++i) {
IE) {

ayback complete

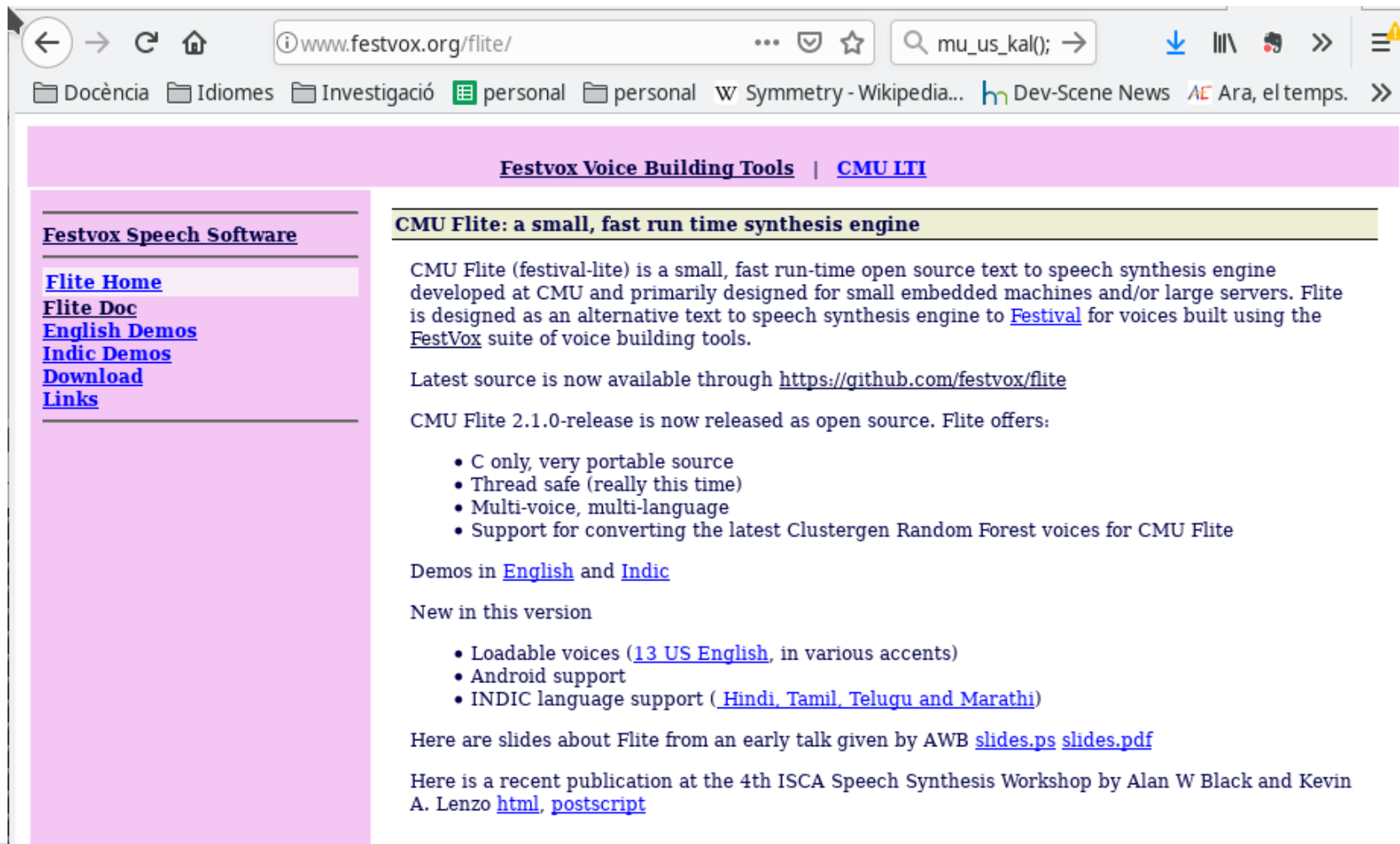
re continuing,
p run
)

Portlibs para N3DS

- Procesado de audio: libSDL_mixer
- ID3: libid3tag
- MIDI: libmikmod, libWildMidi.a
- MP3: libmad, libout123, libmpg123
- OGG Vorbis: libogg, libvorbisidec
- OPUS: libopus, libopusfile, libopusurl
- Sintetizador de voz: libflite, libflite_usenglish, libflite_cmulex, libflite_cmu_time_awb, libflite_cmu_us_slt, libflite_cmu_grapheme_lang, libflite_cmu_us_rms, libflite_cmu_grapheme_lex, libflite_cmu_us_kal16, libflite_cmu_us_kal, libflite_cmu_indic_lang, libflite_cmu_indic_lex

Portlibs

- Flite



The screenshot shows a web browser window with the address bar displaying `www.festvox.org/flite/`. The browser's address bar also shows a search query `mu_us_kal();` and a download icon. The browser's tab bar shows several open tabs: "Docència", "Idiomes", "Investigació", "personal", "Symmetry - Wikipedia...", "Dev-Scene News", and "Ara, el temps.". The website content is divided into a left sidebar and a main content area. The sidebar, titled "Festvox Speech Software", contains links to "Flite Home", "Flite Doc", "English Demos", "Indic Demos", "Download", and "Links". The main content area, titled "CMU Flite: a small, fast run time synthesis engine", contains the following text: "CMU Flite (festival-lite) is a small, fast run-time open source text to speech synthesis engine developed at CMU and primarily designed for small embedded machines and/or large servers. Flite is designed as an alternative text to speech synthesis engine to [Festival](#) for voices built using the [FestVox](#) suite of voice building tools." "Latest source is now available through <https://github.com/festvox/flite>" "CMU Flite 2.1.0-release is now released as open source. Flite offers:"

- C only, very portable source
- Thread safe (really this time)
- Multi-voice, multi-language
- Support for converting the latest Clustergen Random Forest voices for CMU Flite

 "Demos in [English](#) and [Indic](#)" "New in this version"

- Loadable voices ([13 US English](#), in various accents)
- Android support
- INDIC language support ([Hindi](#), [Tamil](#), [Telugu](#) and [Marathi](#))

 "Here are slides about Flite from an early talk given by AWB [slides.ps](#) [slides.pdf](#)" "Here is a recent publication at the 4th ISCA Speech Synthesis Workshop by Alan W Black and Kevin A. Lenzo [html](#), [postscript](#)"

Portlibs

- MikMod



MikMod homepage

[What is MikMod?](#) | [News](#) | [Features](#) | [Screenshots](#) | [License](#) | [Files](#) | [Version Control](#) | [Documentation](#) | [Mailing list](#) | [Ports and other versions](#) | [Where to get music](#)

What is MikMod?

Mikmod is a module player and library supporting many formats, including mod, s3m, it, and xm. Originally a player for MS-DOS, MikMod has been ported to other platforms, such as Unix, Macintosh, BeOS, and Java(!!)

Mikmod main authors are Jean-Paul Mikkers (MikMak), Jake Stine (Air Richter) and Frank Loemker. Steve McIntyre was the first Unix maintainer, followed by Peter Amstutz, Miodrag Vallat and finally Raphaël Assénat.

Unfortunately, since Raphaël Assénat did not have enough free time to work on MikMod those days, releases somewhat came to an halt. This is why he handed the baton to [Shlomi Fish](#) in order to add new features, fix bugs and bring the project further. As of September 2013, Shlomi handed the baton to [Ozkan](#) as the maintainer for the project.

Bibliografía y enlaces



- *AudacityTeam.org* <<http://web.audacityteam.org/>>
- *Enciclopedia sobre Alta Fidelidad, Cine en Casa, Imagen y Acústica*
<http://www.duiops.net/hifi/enciclopedia/index.html>
- *LABORATORIO DE SONIDOS* <<http://pagciencia.quimica.unlp.edu.ar/labsonid.htm>>
- *E. Asensio. (2013). Estudio práctico de los API no oficiales de desarrollo e interacción con audio para Nintendo DS sobre GNU/Linux.* <
<http://hdl.handle.net/10251/32744>>.
A guide to homebrew development for the Nintendo DS <
<http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html>>
- *Jaeden Amero (Patater). 2010. Introduction to Nintendo DS Programming*
<<http://www.patater.com>>
- *Développement amateur sur Nintendo DS Partie 4*
<<http://www.portabledev.com/pages/ds/tutoriels/tutos.-chris-double/partie-4-le-son.php>>.
- *3DS Brew* <<https://www.3dbrew.org>>.
- *Citra – Emulator* <<https://citra-emu.org/>>.
- *libctr documentation* <<http://smealum.github.io/ctrulib/>>.