ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto fin de Carrera Grado

Grado en Ingeniería Informática de Computadores

Fundamentos del Lenguaje Csound

Realizado por
Rafael Escudero Lirio

Dirigido por
Víctor Jesús Díaz Madrigal

Departamento
Lenguajes y Sistemas Informáticos

Sevilla, septiembre de 2020 (v.0.0.1)

Resumen

El lenguaje de programación Csound está principalmente destinado a la síntesis y producción de sonido en un ámbito musical. Es un lenguaje de código abierto, accesible en la plataforma GitHub y su compilador está programado en C, de ahí su nombre. Recibe periódicamente contribuciones de desarrolladores de diferentes partes del mundo y se encuentra en su versión 6.14.0 a la fecha de realización de este trabajo. Es relativamente poco conocido al ser de uso muy específico y al estar toda su documentación dedicada a personas angloparlantes.

Al ser éstas las circunstancias del lenguaje se ha formado a su alrededor una comunidad dedicada que se agranda con el paso del tiempo, con el fin de seguir desarrollando su tecnología y de ahí la justificación del principal motivo del presente trabajo de fin de grado:

Dar a conocer los fundamentos del lenguaje Csound.

Para tal fin, se estructurará el presente trabajo a modo de guía introductoria de uso del lenguaje. Destacando las principales características de éste y priorizando la escalada progresiva de complejidad al decidir el orden de exposición de los conceptos, con intención de favorecer el aprendizaje a medida que se vaya usando el documento. Se priorizará también que el contenido se exponga de manera unitaria para favorecer el uso del presente documento como guía de consulta rápida de conceptos básicos de Csound.

Se exponen a continuación los principales objetivos del presente trabajo:

- Dar a conocer en mayor medida el lenguaje de programación Csound.
- Proporcionar una guía de aprendizaje introductorio al lenguaje Csound.
- Proporcionar un documento de consulta rápida de conceptos del lenguaje Csound.
- Proporcionar ejemplos prácticos de programación usando el lenguaje Csound a modo de demostración capacitiva del lenguaje.

Por último destacar que tecnologías como Csound invitan al trabajo colaborativo e interdisciplinar en distintos ámbitos como son en este caso la informática y la música. Es por ello fundamental dar a conocer sus diferentes usos con el fin último de ampliar el desarrollo de los conocimientos tecnológicos.

Índice general

Índice de cuadros

Índice de figuras

Índice de código

CAPÍTULO 1

Introducción al Lenguaje y Fundamentos del Sonido

1.1– ¿Qué es Csound?

Csound es un lenguaje de programación de alto nivel orientado a objetos dedicado a la síntesis, edición y producción de sonido. Su sintaxis es concreta y su compilador está codificado en lenguaje C. Entre los usos prácticos del lenguaje podemos encontrar ejemplos en la página oficial `https://csound.com/projects.html` de proyectos dedicados a la síntesis de música en directo, edición de sonido mediante efectos generados programáticamente y creación de interfaces VST o instrumentos virtuales entre otros.

Podemos referirnos a Csound como un "Compilador de Sonido".

1.2– ¿Por qué usar Csound?

Hay muchas buenas razones para usar Csound. Csound es software libre de código abierto con licencia LGPL, está en constante desarrollo y tiene una comunidad de desarrolladores que crece día a día,proporciona además un punto de conexión entre la disciplina informática y el ámbito de la música y el sonido. Es además una gran herramienta científica al facilitar la exposición y experimentación de conceptos relacionados a las ondas del sonido, con una amplia librería de funciones y objetos útiles para ello. En lo que se refiere a la composición musical, Csound se ha usado principalmente para generar música electrónica a lo largo de su historia aunque podemos encontrar a compositores de cualquier género músical o tipo de instrumento sacándole provecho al lenguaje. No es de extrañar vistas las tendencias actuales en el mundo de la música, donde para triunfar a nivel multitudinario es prácticamente imprescindible contar con un buen productor que sepa embellecer el sonido. Es frecuente ver a usuarios del lenguaje interpretando su música en directo con ayuda directa de éste. Y por último debo destacar que Csound es compatible con los principales sistemas operativos del mercado, desde Windows a iOS pasándo por distrbuciones Linux. Y como veremos más adelante en esta guía, sus funciones pueden llamarse desde el código de otros lenguajes como Python, java o C.

1.3– Breve historia de Csound

De manera resumida, Csound fue desarrollado en un principio por Barry Vercoe (véase la figura 1.1) en 1985 en el MIT[1] Media Lab. Y desde la década de los años 1990, una amplia variedad de desarrolladores ha colaborado a su código abierto, aportando además documentación, ejemplos y artículos sobre el lenguaje.

Figura 1.1: Barry Vercoe

Para hablar de los verdaderos orígenes de Csound debemos remontarnos a la década de los años 1970, a los orígenes de la producción informática de sonido. Matt Mathews creó MUSIC, el primer lenguaje informático para la generación de ondas de audio digital. En éste se basarían sus posteriores iteraciones: Music1, Music2, Music3, Music4, Music4B... Hasta llegar a Music11, desarrollado por el mentado Barry Vercoe, y del cual Csound es sucesor directo. Posteriormente el desarrollo del lenguaje ha continuado gracias a su comunidad con John Fitch de la University of Bath a la cabeza y como dueño del repositorio de código abierto en la plataforma GitHub. En la actualidad se realizan periódicamente las ICSC[2], conferencias dedicadas a Csound a nivel internacional de manera periódica, siendo la última fecha de realización el 27 de septiembre de 2019 en la actualidad del presente documento.

1.4– Las características del lenguaje

A continuación se muestra una lista de las características principales y técnicas del lenguaje:

- Usado para la síntesis, edición y análisis de sonido y música.
- Lenguaje de código abierto.
- Programación funcional.
- Licencia de distribución LGPL.
- Orientado a objetos.
- Compilador programado en lenguaje C.
- 30 años de desarrollo.
- Compatibilidad con otros lenguajes y retrocompatibilidad de versiones.

[1] Massachusetts Institute of Technology
[2] International Csound Conference

1.5– Alternativas a Csound

Voy a hablar de tres principales alternativas a Csound, siendo estos lenguajes dedicados principalmente a la síntesis, edición y análisis del sonido.

Lenguajes Gráficos

- Max/MSP
- PureData

Lenguaje Escrito

- SuperCollider

1.5.1. Max/MSP

Max es un lenguaje de programación gráfico (véase figura 1.2) dedicado a la música y sonido al igual que Csound pero a diferencia de éste, no es de código abierto. Lo desarrolla y mantiene la empresa Cycling '74 y actualmente puede probarse gratuitamente durante los primeros 30 días de uso. Podría decirse que es por excelencia el lenguaje comercial para el desarrollo de aplicaciones comerciales destinadas al ámbito musical.

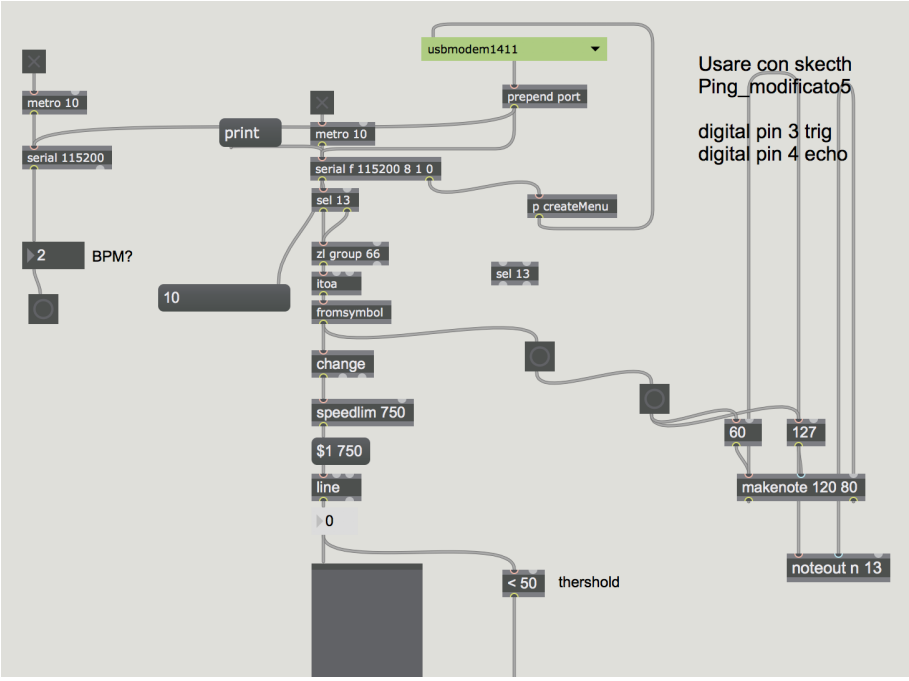

Figura 1.2: Ejemplo Max/Msp

1.5.2. PureData

PureData podría ser considerado el contraparte de código abierto a Max/MSP, pues al igual que éste se trata de un lenguaje gráfico (véase la figura 1.3) pero en esta ocasión de libre desarrollo como Csound. La principal ventaja de este lenguaje respecto a Csound es su paradigma gráfico que puede resultar atractivo a profesionales del ámbito musical o científico que no estén totalmente acostumbrados a usar los tradicionales lenguajes escritos y encuentren en PureData un acercamiento más amigable al mundo de la programación

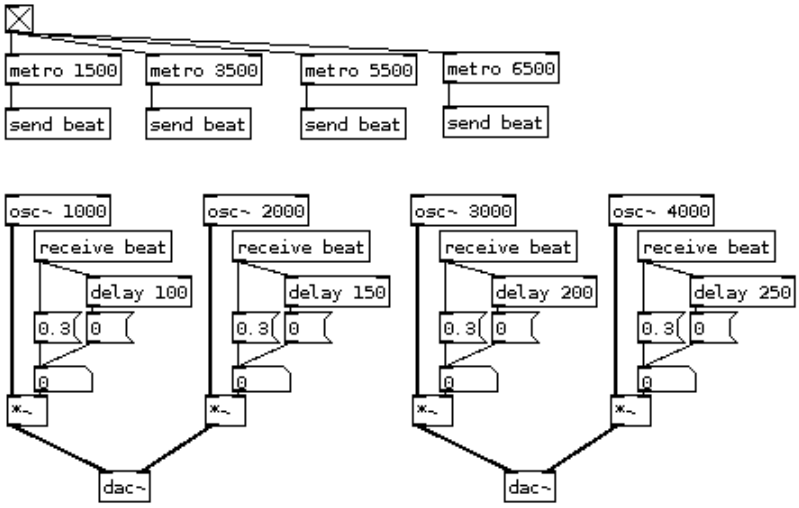

Figura 1.3: Ejemplo PureData

1.5.3. SuperCollider

SuperCollider es de entre las alternativas aquí descritas la más parecida a Csound puesto que además de tener una sintaxis de lenguaje escrito, es también de código abierto. Cuenta además con una sintaxis parecida a lenguajes muy conocidos como C o Ruby, de esta forma parecería más atractivo a iniciados y profesionales de la programación en una primera instancia. Por tanto para observar las verdaderas diferencias entre lenguajes como Csound y SuperCollider tendremos que indagar más a fondo en estos lenguajes y aprender sus características más concretas como haremos en esta ocasión con Csound.

1.6– IDEs para usar Csound

Existen varias alternativas en lo que respecta a entornos de desarrollo de Csound, a continuación se describen 3 opciones[3]:

1.6.1. CsoundQT

CsoundQT es el entorno de desarrollo predeterminado de Csound, prueba de ello es que se instala automáticamente cuando instalamos Csound en el equipo. Tiene una interfaz sencilla, una librería muy completa de ejemplos y capacidad de ampliación con extensiones. Es un buen entorno para tomar una aproximación lo más simplificada posible al lenguaje.

Versión actual: 0.9.6

Plataformas disponibles: Windows, OSX, Debian/Ubuntu.

Página principal: `http://csoundqt.github.io/`

[3]En los ejemplos de código de este documento se ha usado principalmente el IDE Cabbage, no obstante se especificará el IDE usado en cada ocasión en caso de ser necesario por diferencias de uso sustanciales.

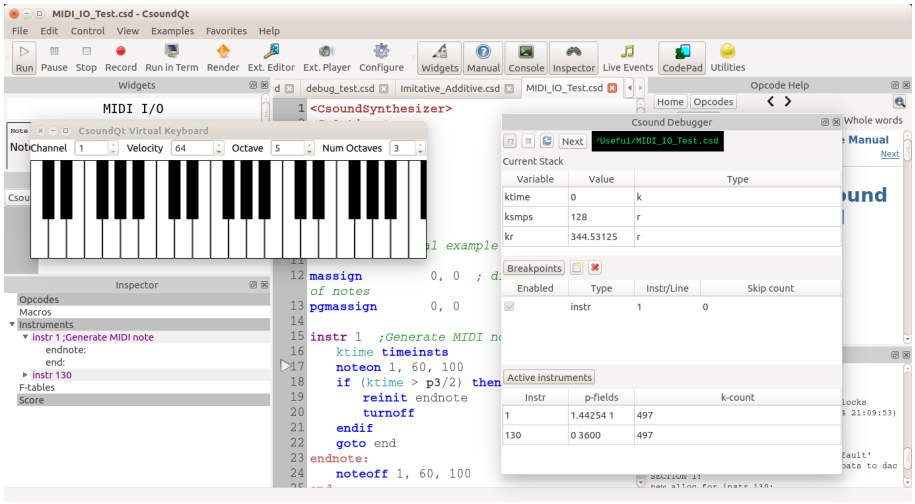

Figura 1.4: CsoundQt

1.6.2. Blue

Blue está enfocado al uso intermedio/avanzado del lenguaje Csound contando con varios plugins por defecto y una interfaz más recargada pero no por ello más difícil de usar o menos legible. Posee varias ventajas tipo framework como los Polyobject, NoteProcessors o el Orchestra manager y es de nuevo una gran opción para tener una primera toma de contacto con el Lenguaje.

Versión actual: 2.8.0

Plataformas disponibles: Windows, OSX, Linux.

Página principal: `https://blue.kunstmusik.com/`

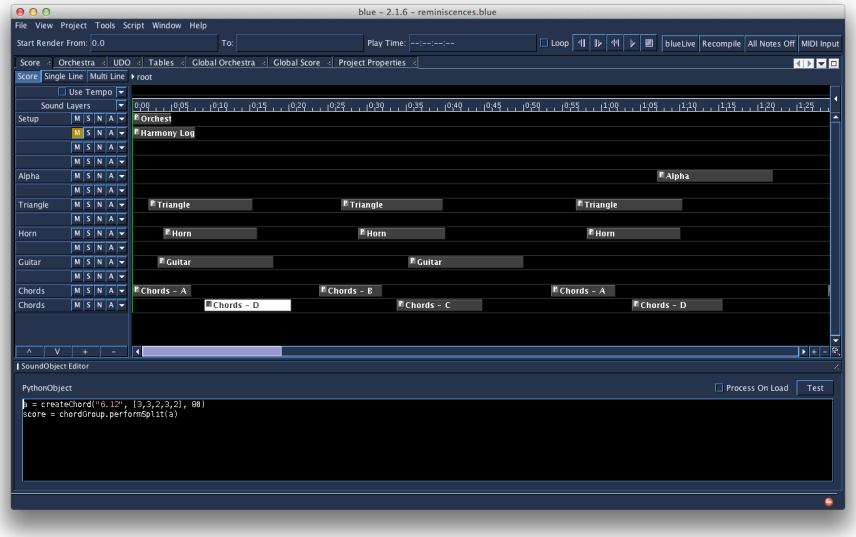

Figura 1.5: Blue

1.6.3. Cabbage

Cabbage es un IDE muy completo, posee una gran librería con utilidades de interfaz gráfica de manera que se facilita el desarrollo completo de instrumentos y plugins musicales existiendo incluso la opción de exportar el código que estamos programando como VST. Cuenta además con una interfaz personalizable y su instalador proporciona la opción de instalar la última versión de Csound si no lo teníamos instalado previamente. Por último cuenta con un instalador para sistemas android para que podamos usar los plugins que programemos con cabbage en estos sistemas.

Versión actual: 2.3.0

Plataformas disponibles: Windows, OSX, Linux.

Página principal: `https://cabbageaudio.com/`

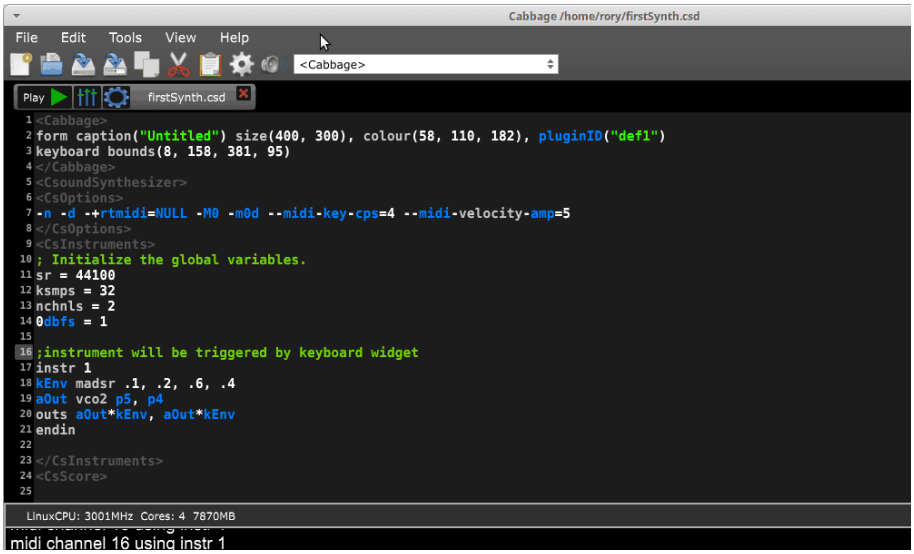

Figura 1.6: Cabbage

CAPÍTULO 2

La Sintaxis del Lenguaje

2.1– Hello Csound!

Empecemos por mostrar cómo sería el clásico *Hello World!* en Csound. Así tendremos un código básico al que hacer referencia a los largo de este capítulo[1]:

```
1  <CsoundSynthesizer>
2  <CsOptions>
3  </CsOptions>
4  <CsInstruments>
5
6  instr 1
7     prints "Hello World!%n"
8     aSin    oscils  0dbfs/4, 440, 0
9     out     aSin
10 endin
11 ;Esto es un comentario
12 </CsInstruments>
13 <CsScore>
14 i 1 0 1
15 </CsScore>
16 </CsoundSynthesizer>
```

Código 2.1: Hello World! en Csound

2.2– A tener en cuenta

Estas son algunas consideraciones básicas del lenguaje que debemos tener presentes:

- Es sensible a mayúsculas/minúsculas.
- Usa especificadores de formato al imprimir variables.
- Para realizar comentarios en el código usaremos ; al inicio de la línea o usar /* y */ para comentar varias líneas.
- Partes del código divididas en etiquetas XML.

[1] Durante el resto del presente capítulo se hará referencia a la figura de código anterior siempre que se hable de líneas concretas de código.

2.3– División por etiquetas

Csound divide la estructura de su código con etiquetas XML, empezaremos por hablar de las más básicas: <CsInstruments>y <CsScore>. Hasta llegar a etiquetas más exclusivas como <Cabbage>.

- **Etiqueta <CsInstruments>**: En esta etiqueta se incluirán las definiciones de los instrumentos que crearemos. Un poco más adelante trataremos de entender qué es un instrumento en Csound. En nuestro ejemplo la etiqueta <CsInstruments>abarca desde la línea 4 a la línea 12, y podemos observar la definición de un instrumentos entre las líneas 6 y 10.

- **Etiqueta <CsScore>**: Aquí haremos uso práctico de nuestros instrumentos, les diremos cómo ejecutarse y durante cuánto tiempo. En el código del que disponemos, la etiqueta <CsScore>abarca desde la línea 13 a la 15 y tenemos un ejemplo sencillo de ejecución en la línea 14 al que volveremos más adelante.

- **Etiqueta <CsOptions>**: Se inclurán aquí las especificaciones técnicas para interactuar con hardware u otros dispositivos.

- **Etiqueta <CsoundSynthesizer>**: Todo el código, incluidas las etiquetas mencionadas anteriormente, debe estar incluido en <CsoundSynthesizer>. Es la forma que tiene el compilador de saber dónde empieza y dónde acaba el código Csound.

- **Etiqueta especial <Cabbage>**: Es la única etiqueta que no debe estar dentro de <CsoundSynthesizer>puesto que es exlusiva de Cabbage, IDE que usaremos principalmente en este documento. En esta etiqueta incluiremos el código referente a las opciones de personalización de interfaz gráfica de nuestro programa, hablaremos más de ella en secciones referentes al uso del IDE Cabbage. Para mayor referencia y guía de uso de esta etiqueta y su contenido, visitar la sección “La etiqueta <Cabbage>”del capítulo “Cabbage: Guía de uso”.

2.4– Palabras reservadas

Las palabras reservadas son variables globales con una funcionalidad especial para Csound como delimitar bloques de código o determinando valores configurables. Pueden inicializarse a un valor determinado en nuestro código que queda posteriormente grabado en el tiempo de compilación. Veamos algunas de las palabras reservadas más comunes en Csound:

- **instr/endin**: Las palabras reservadas instr y endin sirven para determinar el comienzo y el final del bloque de código necesario para crear un instrumento en Csound. Podemos ver un ejemplo de uso en las líneas 6 y 10 de la figura de código 2.1. Además debemos asignar un nombre o identificador al instrumento acompañando a la palabra reservada instr, en nuestro ejemplo de código le damos el nombre “1” a nuestro instrumento.

- **sr**: Indica el valor del sample rate. Wl valor predeterminado es de 44100Hz, es decir, 44100 veces cada segundo. Normalmente usaremos un valor 44100 o de 48000 según la compatibilidad de la tarjeta de sonido de nuestro equipo.[2]

- **nchnls**: Se trata del número de canales de salida de audio que usamos en nuestro programa. Con nchnls = 1 conseguimos sonido mono, con nchnls = 2 stereo, nchnls = 4 cuadrafónico. (véase figura 2.1)

[2]Para mayor entendimiento de los conceptos relacionados al sampleo véase el apartado “El Sampleo y Sample Rate”del capítulo “Fundamentos del Sonido”

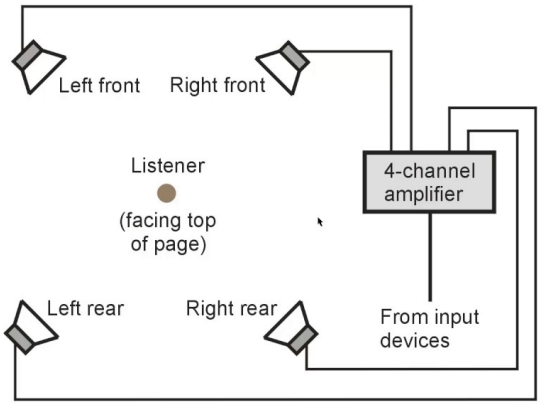

Figura 2.1: Sonido cuadrafónico

- **0dbfs**: Determina el valor relativo que usaremos en el programa como 0 decibelios y a partir del cual aumentaremos o disminuiremos los decibelios de volumen. Por defecto tiene un valor de 32767 aunque lo más lógico al usar un computador para determinar este valor es dar un valor absoluto de 0dbfs = 1 puesto que la intensidad final del sonido no depende únicamente de nuestro código sino de nuestros altavoces, la configuración de nuestra tarjeta de sonido o la configuación de nuestro IDE de Csound. Darle un valor 1 a 0dbfs nos permite tener una referencia útil y muy manejable además de ser la usada por convenio en los programas de audio digital en lugar de dar valores absolutos que bien pueden desembocar en resultados distintos a lo que deseábamos según el equipo en el que ejecutemos nuestro código. [3]

2.5– Las variables en Csound

En Csound encontramos 3 tipos principales de variable: **i**, **k** y **a**, que representan diferentes ratios de refresco para su valor. Para definir una variable de cualquiera de estos tipos basta con escribir i, k o a como primera letra del nombre de nuestra variable. Algunos ejemplos serían: iFreq, kAux, aCualquierNombre, etc...

Veamos a continuación qué implica elegir uno u otro tipo:

Cuando ejecutamos código Csound, una vez compilado el código e inicializada la configuración, el tiempo de ejecución se pasa recorriendo el código contenido de nuestros instrumentos en bucle. Podemos decir que nuestro código se recorre con una determinada frecuencia mientras se ejecuta, y dentro de esta frecuencia podemos seleccionar cómo de frecuentemente volveremos a determinar el valor de nuestra variables. Para determinar ese ratio usamos i,k y a.

- **Variables i**: Las variables refrescan su valor una única vez, cuando el instrumento es inicializado. A efectos prácticos se trata de constantes internas a los instrumentos.
- **Variables k**: Tienes una frecuencia de refresco media. Mayor a las variables i, menor a las variables a.
- **Variables a**: Tienen el mayor ratio de los 3 tipos puesto que estas variables refrescan su valor cada vez que Csound recorre su código.

[3] Para mayor entendimiento del concepto de decibelio, ir a la sección “El decibelio”del capítulo “Fundamentos del Sonido”

La principal razón para que tengamos estas diferente opciones es poder jugar con la optimización de nuestro código. Bien podríamos hacer que todas nuestras variables fuesen de tipo ‘ a’pero esto implicaría en la mayoría de los casos un aumento de carga innecesario durante el tiempo de ejecución, que puede ser crucial si nuestro objetivo pasa por ejecutar código en directo al interpretar una pieza musical.

2.6– Los Opcodes

Los Opcodes realizan diferentes funciones predefinidas, son de hecho el equivalente más parecido a una función de librería en un lenguaje de alto nivel convencional. Por ejemplo el Opcode **reverb** aplica reverberación a la señal entrante, el opcode **poscil** produce una señal oscilatoria de alta precisión. Existe una gran variedad de Opcodes (Actualmente más de 1500) con funcionalidades muy concretas y útiles en el mundo del sonido.

2.6.1. Sintaxis del Opcode

Veamos primero la sintaxis generalizada y después un ejemplo sencillo de uso de opcode. Analicemos su sintaxis para comprenderla:

```
1    aOutput opcode input1, input2, input3, ...
```

Código 2.2: Sintaxis generalizada de un Opcode

Refiriéndonos a la figura de código 2.2. **opcode** es el nombre del opcode que queremos usar, **input1**, **inpu2** e **input3** representan los diferentes atrbutos de entrada al opcode, son generalmente valores numéricos y su cantidad es indeterminada. Existen opcodes sin atributos de entrada. Por último **aOutput** es la variable de salida del opcode en la que vamos a guardar nuestro resultado esperado a la que haremos referencia más adelante en nuestro código. Podríamos resumir la sintaxis básica de uso de opcodes en: **salida** ← **función** ← **entrada/s**

```
1    aSin    oscils  0dbfs/4, 440, 0
```

Código 2.3: Ejemplo de uso del Opcode oscils

Observemos un uso práctico del opcode **oscils** en la figura de código 2.3:

- **Salida**: La variable de salida es **aSin** (variable de tipo ‘ a’) en la que quedará guardada la información de la onda generada y cuyo valor se refrescará cada vez que Csound recorra el instrumento contenedor.
- **Función**: El opcode es **oscils** cuya funcionalidad es generar una señal oscilatoria sinoidal, este opcode requiere de 3 variables de entrada de las que hablaremos a continuación.
- **Entradas**: Observamos 3 entradas, **0dbfs/4, 440, 0** → **iamp, icps, iphs**. Hablemos de cada una de ellas: 0dbfs/4 (recordemos el uso de 0dbfs como palabra reservada)
    - **iamp**: Cuyo valor es 0dbfs/4. Se trata del valor de la amplitud de onda de salida, es decir, la amplitud de la onda de salida tendrá el valor de un cuarto del valor marcado en nuestro código como 0 decibelios (**0dbfs**).
    - **icps**: Frecuencia de la onda de salida en Hz. El valor en nuestro ejemplo es de 440, por lo tanto obtendremos una frecuencia de onda de salida de 440Hz.
    - **iphs**: Determina la fase de onda de la onda de salida, nuestro valor es 0 por lo que nuestra onda de salida comienza en su fase 0.[4]

[4]Para comprender el concepto de fase de onda, acudir al capítulo “Fundamentos del sonido“ sección “La onda y sus características”

Como podemos observar, una vez comprendida, la sintaxis de uso de los opcodes es sencilla pero muy concreta al no parecerse del todo a la sintaxis de lenguajes más convencionales. Para usar de forma satisfactoria los opcodes debemos tener presente la librería de referencia canónica de Csound: `http://www.csounds.com/manual/html/PartReference.html` donde encontraremos un librería muy amplia de ejemplos de uso concreto de cada uno de los opcodes disponibles.

2.7– Los Score events

Los **Score events** son las líneas de código que van dentro de la etiqueta <CsScore>. Por ejemplo, volvamos a nuestro "Hola mundo"(figura de código 2.1), revisemos el score "**i 1 0 1**"y comprendamos su sintaxis:

Para ejecutar un instrumento usaremos la siguiente sintaxis base:

[i identificador instanteInicial duración]

Siendo **identificador** el nombre del instrumento que vamos a usar, **instanteInicial** el tiempo desde el momento de ejecución del código que tarda en iniciarse nuestro instrumento y **duración** la cantidad de tiempo durante la cual se ejecuta el instrumento.

2.8– Usando lo aprendido en un Sintetizador MIDI

Para afianzar lo aprendido sobre la sintaxis de Csound, vamos a programar un sintetizador MIDI básico que haga uso de las funcionalidades más elementales del lenguaje.

2.8.1. El instrumento base <CsInstruments>

Vamos a usar el opcode **vco2**, que genera una onda oscilatoria limitada por banda. Veamos en primer lugar cómo usar este opcode: **ares vco2 kamp, kcps**. Tenemos dos atributos de entrada obligatorios, **kamp** que define la amplitud de la onda y **kcps** para determinar su frecuencia. Para nuestro ejemplo, **aOut** contedrá la información referente a una de amplitud 1 y frecuencia 440Hz. Con la línea **out aOut** se indica que el valor de salida que se devuelve al usar nuestro instrumento es el de **aOut**.

Y aquí nuestro instrumento en su forma base:

```
1 instr 1
2 aOut vco2 1, 440
3 out aOut
4 endin
```

Código 2.4: Instrumento base del sintetizador

Inicialicemos además algunas variable globales y retoquemos algunas variables internas de nuestro instrumento:

```
1 sr = 44100
2 ksmps = 32
3 nchnls = 2
4 0dbfs = 1
```

Código 2.5: Variables globales inicializadas

Usaremos el opcode de esta manera: **aOut vco2 iAmp, iFreq**, siendo **iFreq = p4** y **iAmp = p5**. Estas variables, p5 y p4, las determinan los valores de entrada MIDI con el objetivo de usar nuestro sintetizador de forma dinámica.

2.8.2. Las opciones de configuración <CsOptions>

Veamos qué opciones de configuración podemos añadir a nuestro sintetizador. Será importante para que el uso de señales MIDI sea funcional así que usaremos la opción **-+rtmidi=NULL -M0 -m0d**. No usaremos plgins MIDI de tiempo real en este ejemplo así daremos el valor **NULL** a **-+rtmidi**, con -M seguido de un identificador podremos seleccionar qué dispositivo vamos a usar. Como sólo tendremos disponible el dispositivo virtual de Cabbage, usaremos **-M0**.

Para relacionar las variables de nuestro instrumento como comentamos anteriormente usaremos **–midi-key-cps=4** para pasar la frecuencia de la nota MIDI actual a **p4** y **–midi-velocity-amp=5** para pasar la velocidad de pulsación a —textbfp5.

Bastará de momento con tener un entendimiento básico de la etiqueta <CsOptions>de momento, aunque profundizaremos en ella más adelante.

Quedaría así nuestro código:

```
1 -+rtmidi=NULL -M0 -m0d --midi-key-cps=4 --midi-velocity-amp=5
```

Código 2.6: Opciones de configuración

2.8.3. Ejecutando nuestro instrumento <CsScore>

La etiqueta <CsScore>contiene nuestros **score events** auqnue en esta ocasión no vamos a necesitarlos como tal, únicamente usaremos el score **f0 z** que sirve para indicar a Csound que se quede esperando eventos tanto tiempo como queramos. Esto ideal para el caso puesto que queremos que Csound escuche nuestros eventos de señal MIDI.

2.8.4. Creando una interfaz <Cabbage>

Usaremos dos widgets de forma sencilla[5]. En primer lugar un widget **form** para crear la ventana base de la interfaz, que tendrá un identificador **size(Width, Height)** para definir el tamaño de la ventana, un identificador **colour(r, g, b)** para dar un color personalizado al fondo de la ventana en formato rgb y por último un identificador **pluginid(id)** (único identificador obligatorio) para dar un nombre identificativo a la ventana y poder hacer referencia a ella en el resto del código.

En segundo lugar usaremos un widget tipo **keyboard** para crear nuestro teclado virtual que hará las veces de instrumento MIDI. Tendrá un único identificador **bounds(x, y, width, height)** para determinar la posición de coordenadas y el tamaño del teclado.

Un posible ejemplo quedaría tal que así:

```
1 form caption("Primer Sinte") size(450, 300), colour(250, 110, 20),
     pluginID("sin1")
2 keyboard bounds(30, 150, 380, 100)
```

Código 2.7: La interfaz del teclado

[5]Se recomienda revisar la sección “Los widgets”del capítulo “Cabbage: Guía de uso”.

Y el resultado al ejecutar esta parte del código será algo parecido a esto:

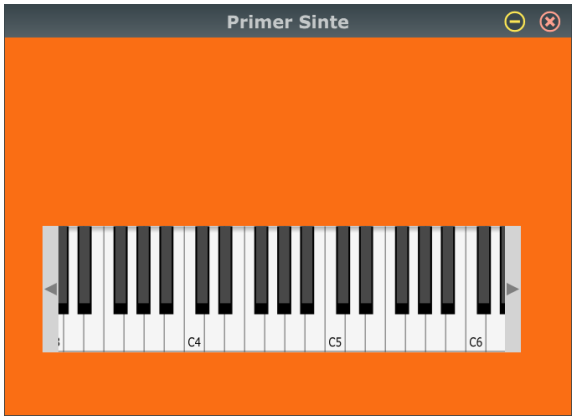

Figura 2.2: Nuestro primer sintetizador

2.8.5. El teclado básico

Si unimos todas las partes, nuestro código queda de esta manera:

```
<Cabbage>
form caption("Primer Sinte") size(450, 300), colour(250, 110, 20),
    pluginID("sin1")
keyboard bounds(30, 150, 380, 100)
</Cabbage>
<CsoundSynthesizer>
<CsOptions>
-+rtmidi=NULL -M0 -m0d --midi-key-cps=4 --midi-velocity-amp=5
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
iFreq = p4
iAmp = p5
aOut vco2 iAmp, iFreq
outs aOut, aOut
endin

</CsInstruments>
<CsScore>
f0 z
</CsScore>
</CsoundSynthesizer>
```

Código 2.8: Un teclado básico funcional

Este teclado es ya ciertamente funcional pero para tener un ejemplo más completo y poder llamar 'sintetizador' a este instrumento vamos a añadirle un envelope ADSR[6]. Así conseguiremos las funcionalidades básicas de cualquier instrumento eléctrico.

[6]Se recomienda revisar la sección "El ADSR"del capítulo "Fundamentos del sonido"

2.8.6. Convirtiendo el teclado en un sintetizador

En primer lugar, para añadir un envelope ADSR, podemos seleccionar alguna de las muchas opciones que ofrece Csound en materia de opcodes. Uno de los más sencillos y que puede servirnos perfectamente es el opcode **madsr** cuya sintaxis es **kenv madsr iatt, idec, islev, irel**. Siendo **iatt** el valor de ataque, **idec** el valor de decay, **islev** el valor de sustain u **irel** el valor de release. Recordemos también que las variables i son a efectos prácticos variables constantes del instrumento, que refescarán su valor en este caso entre pulsación y pulsación de una nota.

Bastaría ahora con aplicar este envelope a nuestra onda de salida, una de tantas maneras es multiplicando la salida del opcode **madsr** a los canales de salida de nuestro instrumento de esta forma: **outs aOut*kEnv, aOut*kEnv**.

Necesitamos también dar un valor a los parámetros **iatt**, **idec**, **islev** y **irel**. Podríamos dar valores fijos aunque esto no tendría un uso demasiado funcional, por lo tanto vamos a aprovechar las opciones de interfaz que proporciona Cabbage para vincular estos valores a 4 widgets de tipo slider. Veamos cómo crear el slider para el valor de ataque:

```
1 rslider bounds(12, 14, 105, 101), channel("ata"), range(0, 1, 0.01, 1,
     .01), text("Ataque")
```

Código 2.9: El slider para el valor de ataque

El widget slider no es nuevo, aunque debe destacarse el identificador **channel(chan)** que se usa para otorgar al slider un nombre de canal al que se vincula su valor de salida y que más tarde podremos usar en nuestro código.

Se omite la especificación de los otros 3 sliders que añadiremos pues su sintaxis es análoga cuidándonos únicamente de dar un nombre de canal distinto en cada ocasión.

Haremos uso también del sencillo opcode **chnget** para vincular los canales que hemos creado en los sliders con variables que Csound pueda procesar. Su sintaxis es básica: **ival chnget Sname** siendo Sname el nombre del canal asociado.

Quedaría nuestro código de vinculación de la siguiente manera:

```
1 iAtt chnget "ata"
2 iDec chnget "dec"
3 iSus chnget "sus"
4 iRel chnget "rel"
```

Código 2.10: Vinculación de canales a variables

2.8.7. Últimos detalles

Para concluir vamos a añadir un par de efectos útiles a nuestro sintetizador, el **cutoff** y la **resonancia**. Una forma sencilla de conseguirlo es usando el opcode **moogladder**, cuya sintaxis es: **asig moogladder ain, kcf, kres**. Siendo **ain** la onda a la que añadir los efectos, **kcf** el valor del cuttoff y **kres** el valor de resonancia.

Para el caso vamos a darle a **ain** el valor de aOut, es decir, nuestra onda resultante de usar el opcode vco2. Para los valores de cuttoff y resonancia crearemos simplemente un par de sliders y vincularemos sus valores a variables de Csound del mismo modo que hicimos antes.

2.8.8. El resultado final

Y aquí tenemos al fin el código completo de nuestro sintetizador con efectos:

```
<Cabbage>
form caption("Primer Sinte") size(450, 300), colour(250, 110, 20),
    pluginID("sin1")
keyboard bounds(14, 88, 413, 95)
rslider bounds(12, 14, 70, 70), channel("att"), range(0, 1, 0.01, 1,
    .01), text("Ataque")
rslider bounds(82, 14, 70, 70), channel("dec"), range(0, 1, 0.5, 1,
    .01), text("Decay")
rslider bounds(152, 14, 70, 70), channel("sus"), range(0, 1, 0.5, 1,
    .01), text("Sustain")
rslider bounds(222, 14, 70, 70), channel("rel"), range(0, 1, 0.7, 1,
    .01), text("Release")
rslider bounds(292, 14, 70, 70), channel("cutoff"), range(0, 22000,
    2000, .5, .01), text("Cut-Off")
rslider bounds(360, 14, 70, 70), channel("res"), range(0, 1, 0.7, 1,
    .01), text("Resonance")
</Cabbage>
<CsoundSynthesizer>
<CsOptions>
-n -d -+rtmidi=NULL -M0 -m0d --midi-key-cps=4 --midi-velocity-amp=5
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1
iFreq = p4
iAmp = p5

iAtt chnget "ata"
iDec chnget "dec"
iSus chnget "sus"
iRel chnget "rel"
kRes chnget "res"
kCutOff chnget "cutoff"

kEnv madsr iAtt, iDec, iSus, iRel
aOut vco2 iAmp, iFreq
aLP moogladder aOut, kCutOff, kRes
outs aLP*kEnv, aLP*kEnv
endin

</CsInstruments>
<CsScore>
f0 z
</CsScore>
</CsoundSynthesizer>
```

Código 2.11: Un sintetizador funcional

Con esto termina la introducción a la sintaxis del lenguaje. Como puede observarse, aunque Csound parezca algo enrevesado a primera vista y sin conocimientos previos, una vez entendido el esquema básico y la lógica sintáctica es fácil entender y escribir código lo suficientemente

elaborado como para obtener instrumentos completamente funcionales sin no demasiado esfuerzo. Seguiremos profundizando en el potencial de Csound en posteriores capítulos ahora que hemos alcanzado el nivel fundamental para poder seguir las lecciones sin perdernos.

CAPÍTULO 3

Siguiente capítulo

3.1– Cuadros (mal llamados Tablas)

Este es un ejemplo de inclusión de cuadro en el texto. Véase el cuadro 2.1

elemento elemento	elemento elemento	elemento elemento

Cuadro 3.1: Cuadro de prueba

CAPÍTULO 4

Cabbage: Guía de uso

Cabbage es un IDE para el lenguaje Csound. Es de código abierto y está desarrollado por Rory Walsh.

Será el principal IDE que usaremos a lo largo de este documento puesto que además de contar con todas las comodidades necesarias para el funcionamiento del lenguaje, aporta además funcionalidades para crear interfaces gráficas para nuestro software de manera muy simple pero vistosa.

Se presenta el siguiente capítulo con la intención de dar una guía básica referencial de funcionamiento a la que acudir en caso de ser necesario durante el curso del resto de contenidos.

4.1– Instalación de Cabbage

Cabbage puede instalarse en sistemas Windows, OSX y Linux. Posee incluso un instalador en versión beta para sistemas Android.

Pasos para la instalación en Windows y OSX:

- Acudir a la página `https://cabbageaudio.com/download/` donde encontraremos los enlaces de descarga.
- Seleccionar la versión adecuada para nuestro sistema, en este caso Windows u OSX.

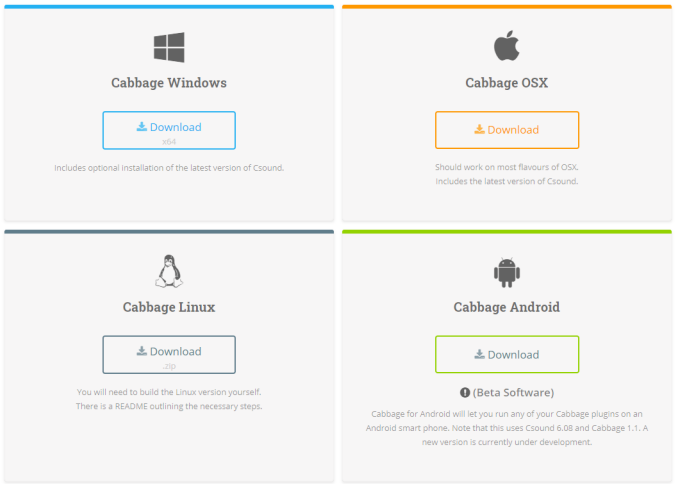

Figura 4.1: Versiones disponibles

- Ejecutamos el archivo descargado y seguimos los pasos de instalación. Para los instaladores de Windows y OSX se incluye una instalación automática del lenguaje Csound en nuestro sistema por lo que una vez instalado Cabbage todo estará listo para usar.

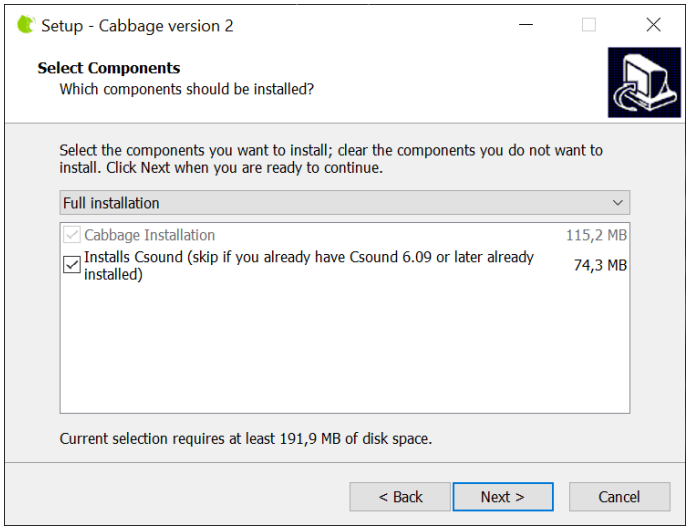

Figura 4.2: Pasos de la instalación

- Por último podemos ejecutar el acceso directo que se instala automáticamente en nuestro escritorio y empezar a usar Csound. Como se observa, Csound tiene una instalación muy fácil en estos sistemas.

4.2– La etiqueta <Cabbage>

A diferencia del resto de etiquetas, la etiqueta <Cabbage>es exclusiva al IDE y proporciona funcionalidades para el diseño de la interfaz de usuario. Veamos algunos de sus usos:

4.2.1. Los Witgets

Llamaremos a los diferentes elementos de interfaz gráfica que aporta Cabbage, widgets. Podemos dividirlos en dos tipos: interactivos (botones, sliders, barras de selección, etc...) y no interactivos (imágenes, indicadores, etc...).

Empecemos por un ejemplo de uso de un slider para comprender la sintaxis:

```
1 <Cabbage>
2    rslider bounds(10, 10, 100, 100), range(0, 1, .5)
3 </Cabbage>
```

Código 4.1: Ejemplo básico de un widget

En nuestra figura vemos que para hacer uso de un widget empezamos por escribir su nombre identificativo de tipo, en este caso **rslider**. Mástarde podemos definir una serie de identificadores para personalizar nuetro widget. Para especificar la posición de nuestro widget y su tamaño usaremos **bounds(x, y, width, height)**. En el caso de nuestro ejemplo estamos posicionando nuestro slider en las coodernadas XY (10,10) y le estamos dando un tamaño de 100*100 píxeles.

Podemos usar también el identificador **range(min, max, value, skew, incr)**. Sus valores min y max marcan el mínimo y máximo valor del slider en cuestión, el resto de parámetros son opcionales. **value** indica el valor inicial del slider. **skew** puede usarse para determinar la salida de

datos del slider de forma no lineal, su valor predeterminado es 1 pero al darle un valor por ejemplo de 0.5 conseguiríamos una salida de datos exponencial. **incr** determina el tamaño de los pasos incrementales que da el slider a usarlo, por ejemplo con un valor 0.4, de un valor cualquiera del slider a sus adyacentes habría necesariamente una distacia en valor a 0.4.

En nuestro ejemplo hemos creado un slider cuyo rango de valores va del 0 al 1 y cuyo valor inicial es 0.5.

CAPÍTULO 5

Fundamentos del Sonido

5.1– Introducción

Este capítulo tiene como función dar una breve introducción a la teoría física del sonido, en concreto a los conceptos fundamentalmente necesarios para entender los ejemplos expuestos en esta guía de Csound. Se presenta por ello como capítulo anexado o capítulo extra de modo que sirva de referencia rápida en otras partes del documento y de manera que un lector con manejo en estos términos pueda saltar su contenido cómodamente.

5.2– El Audio Digital

Para definir el audio digital debemos empezar por saber qué es el sonido:

5.2.1. ¿Qué es el sonido y cómo se transmite?

Sonido: "*Sensación producida en el órgano del oído por el movimiento vibratorio de los cuerpos, transmitido por un medio elástico, como el aire.*"[1]

A ese movimiento vibratorio que se transmite y viaja por el medio podemos llamarlo "Onda de Sonido". Y la forma más simple de describir un movimiento vibratorio, es decir, la onda más simple de todas; es mediante la forma senoidal:

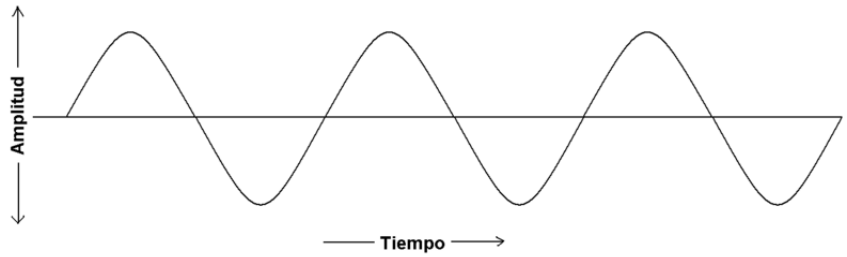

Figura 5.1: Onda Senoidal

[1]REAL ACADEMIA ESPAÑOLA: Diccionario de la lengua española, 23.ª ed., [versión 23.3 en línea]. ¡https://dle.rae.es¿[05 de julio de 2020].

Como sabemos, los medios transmisores están formador por moléculas que ocupan un determinado espacio. Podemos por lo tanto imaginar a una molécula que describe el movimiento vibratorio descrito anteriormente. Podríamos también dedir que cuando la molécula sobrepasa el punto inicial o punto 0 que definimos en la gráfica, la molécula está empujando al resto de moléculas que encuentra en su camino. De forma análoga, cuando la posición de la molécula tiene un valor menor al inicial decimos que la molécula está tirando del resto de moléculas de su entorno.

De esta manera se produce la transmisión del sonido.

5.2.2. La onda de sonido y sus características

Quedaba definida la onda de sonido en el apartado anterior. Si a continuación le añadimos la información de esa vibración de la que hablábamos es constante se producirá lo que llamamos "Onda Periódica".

Toda onda periódica posee 4 características:

- **Periodo**: Es la cantidad de tiempo que tarda la forma de la onda en repetirse, lo llamaremos T y lo expresaremos en segundos.
- **Amplitud**: Distancia máxima de los puntos de la onda respecto a la posición de eje Y (eje "Tiempo'én la figura). Podemos definirla también como la fuerza con la que las moléculas del medio consiguen empujar o tirar del resto de moléculas de su entorno.

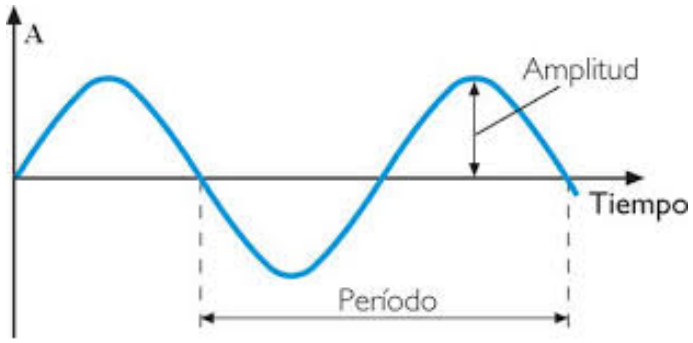

Figura 5.2: Periodo y Amplitud

- **Frecuencia**: La frecuencia de una onda expresa la cantidad numérica de veces que repite su movimiento durante un tiempo determinado. Si la definimos respecto al periodo decimos que es la cantidad numérica de periodos (o repeticiones de la forma de la onda) que ocurren durante un segundo. Se mide en Hercios o Hz, la representaremos con f y podremos calcularla fácilmento puesto que es la inversa del periodo:
  - Frecuencia = 1/Periodo
  - Periodo = 1/Frecuencia
- **Longitud de Onda**: Se trata de la distancia que va del punto inicial al punto final del recorrido de la onda marcada por un periodo. Se mide en metros.
- **Fase**: Punto de partida de la onda. Podemos observar al representar la onda en un gráfico y fijarnos en que el valor inicial del eje Y que no tiene que ser nesesariamente 0.

5.2.3. El Sampleo y Sample Rate

Para representar de manera digital una onda de sonido acústica necesitamos convertir sus valores analógicos a digitales. Esta onda tendrá un valor distinto por cada instante de tiempo y para conseguir recoger de alguna manera estos datos en un computador necesitaremos el concepto de sampleo.

El sampleo consiste en recoger un número determinado de valores en formato digital de una onda sonora por cada segundo de duración.

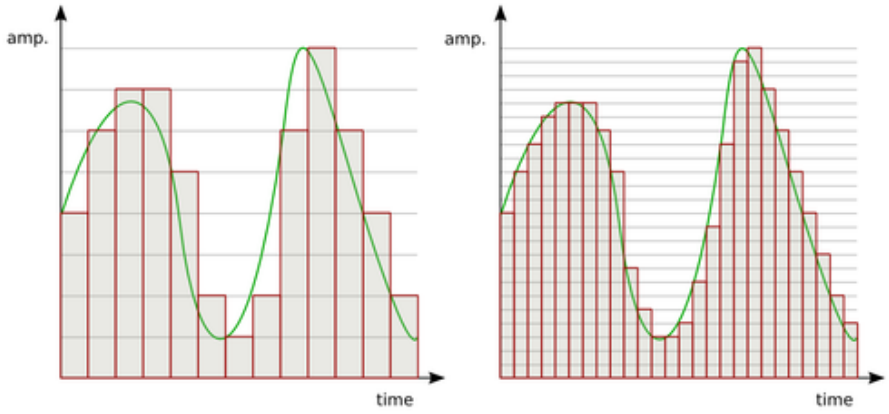

Figura 5.3: Ejemplo de sampleo de una onda

Aquí entra el concepto de "sample rate", que determina la frecuencia de muestreo en la recogida de datos de la onda. En la figura anterior podemos observaruna misma onda de sonido a la que se le realiza un sampleo primero con sample rate menor (izquierda), y con un sample rate mayor (derecha).

Por último debe destacarse que un sample rate mayor no implica necesariamente una coversión digital más feaciente para nosotros a un nivel pragmático puesto que el nivel máximo de frecuencia de datos sonoros que puede captar en oído humano ronda los 20Khz.

Contamos además con que debe respetarse el **Teorema de muestreo de Nyquist-Shannon**:

"Para representar una onda de manera digital que contenga frecuencias de hasta X Hz, es necesario usar un sample rate de al menos 2X muestras por segundo."

De otra manera, los valores digitales no representarían la onda de manera correcta dando lugar al aliasing y al muestreo incorrecto.

Se muestra a continuación una figura de ejemplo de sampleo en la que se toma un sample rate de 40000Hz. De la primera onda, que es de 10KHz observamos que recogemos los datos suficiente cada segundo como para captar toda la información contenida en ella. De hecho podemos observar también que con un sample rate de 20KHz en lugar de 40KHz también captaríamos toda la información necesaria como para recoger la onda al completo en su formato digital cumpliéndose así el Teorema de muestreo de Nyquist-Shannon.

Sin embargo la onda del segundo gráfico tiene una frecuencia de 30KHz y esto implica que con un sample rate de 40KHz como el mostrado, daría resultado una conversión de onda errónea. Dando incluso para este ejemplo una muestra de onda digital idéntica a la del primer gráfico. Sería necesario un sample rate de al menos 60KHz para recoger feacientemente los datos y conseguir una conversión digital satisfactoria de la onda.

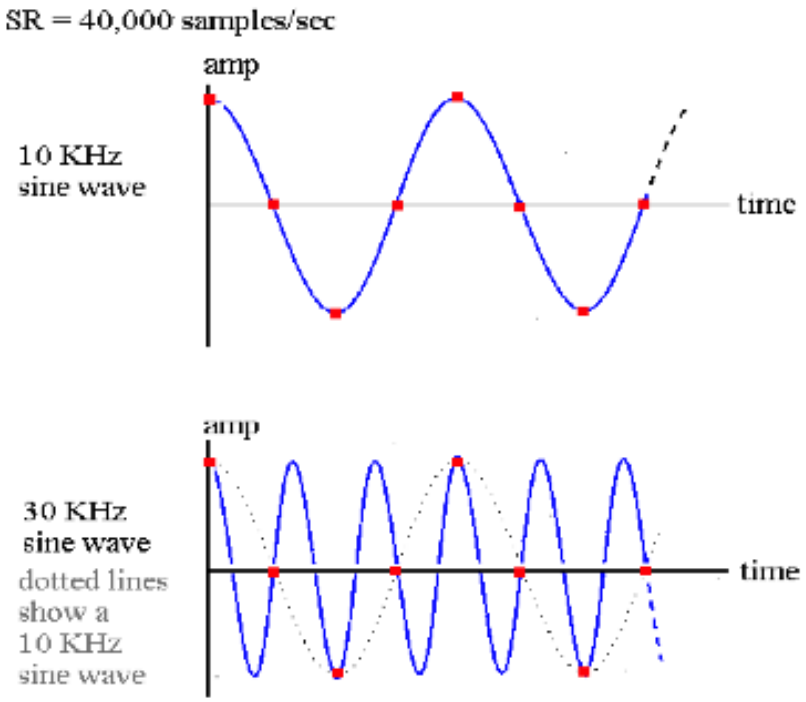

Figura 5.4: Un mismo sample rate para ondas distintas

5.3– Conceptos interesantes

Hablemos a continuación de algunos conceptos no necesariamente intrínsecos al sonido digital pero interesantes para entender mejor los conceptos necesarios para el uso adecuado de Csound.

5.3.1. El decibelio

El decibelio o 'db' es una unidad de medida que representa la intensidad de un sonido. Es la décima parte de un belio y siempre que hablamos de decibelio lo hacemos respecto a un valor de referencia preestablecido de intensidad, normalmente el marcado por el límite por debajo del oído humano en su capacidad para oír: $I_0 = 10^{-12} W/m^2$ que se da en los 1000Hz.

La fórmula para calcular los decibelios es: $10 * log_{10} * \frac{I}{I_0}$, es por tanto una fórmula logarítmica que depende de su valor de referencia I_0. Para una relación $\frac{I}{I_0}$ de 10 tenemos 10db, para una relación de 100 tendremos 20db, para 1000 30db, etc...

Un dato útil a tener en cuenta es que al doblar la amplitud de una onda de sonido obtenemos una diferencia de +6db. De manera análoga si partimos por la mitad el valor de la amplitud de la onda obtenemos un cambio de -6db.

5.3.2. El ADSR

ADSR son las siglas de **A**ttack, **D**ecay, **S**ustain y **R**elease en una onda y conforman la opción más común de envelope o envolvente sonoro, es decir, proporcionan parámetros para poder controlar una onda de sonido.

Veamos cada uno de estos parámetros para entenderlos en conjunto:

- **Attack**: El Attack o Ataque sería lo ocurrido antes de que la onda decaiga y se estabilice. Por ejemplo, un golpe de platillo produce un sonido con mucho ataque, el sonido de una nota tocada en una flauta dulce tendría normalmente poco ataque.
- **Decay**: El decay o decaimiento es lo sucedido entre el ataque ensu máximo punto y la fase estable de la onda. Al rasgar las cuerdas de una guitarra con una púa se produce un sonido con bastante decay.
- **Sustain**: El sustain o sostenibilidad es lo referido a la parte estable de la onda, su duración e intensidad máxima. Un golpe de caja tiene poco sustain, una nota tocada al aire en la cuerda de un bajo eléctrico tiene mucho sustain.
- **Release**: El release es la parte de la onda comprendida entre la fase estable o de sustain y la llegada al valor 0 de intensidad. Los instrumentos de cuerda tienen por lo gerenar una fase de release notable.

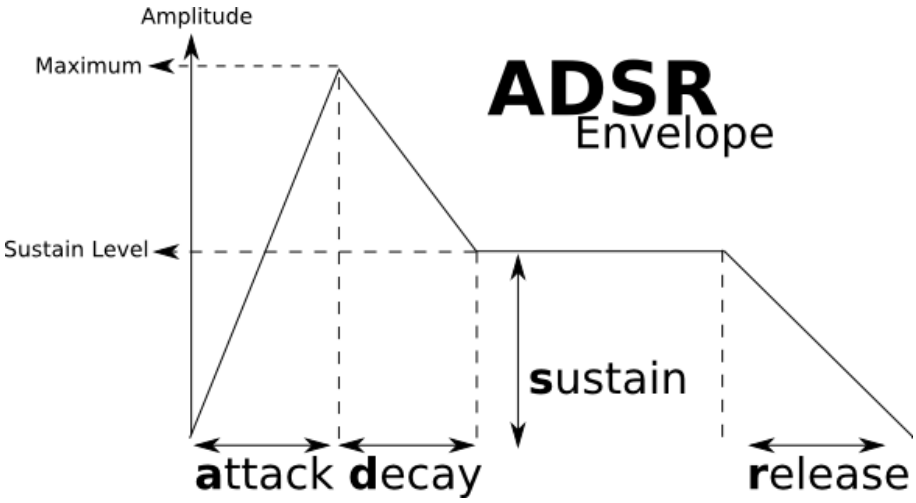

Figura 5.5: El envolvente ADSR de una onda

Es común en el sonido digital poder modificar estos parámetros a placer en instrumentos como los sintetizadores o en prácticamente cualquier instrumento con componentes eléctricos.

5.3.3. El Cutoff y la resonancia

Veamos este par de conceptos útiles para completar y mejorar nuestro sonido:

- **Cuttoff**: Se trata de un filtro de frecuencias. Normalmente se usa para bloquear determinados rangos de frecuencias altas, siendo esto de tipo LP (LOW PASS) aunque en instrumentos modernos pueden encontrar Cutoffs de tipo HP (HIGH PASS) que bloquearían en este caso frecuencias más bajas de lo deseado.
- **Resonancia**: Es el efecto que ocurre por ejemplo cuando la mesa que sujeta unos altavoces en marcha empieza a vibrar mientras algunos sonidos son producidos, querrá decir que el sonido que sale de esos altavoces producen una frecuencia coincidente con la frecuencia de resonancia de la tabla de la mesa y el sonido por tanto se ve amplificado. En el audio digital puede encontrarse la resonancia como un efecto más a añadir a nuestro arsenal de producción musical.

Referencias

Autores, V. (2014). *Escuela Técnica Superior de Ingeniería Informática.* ETSII. Descargado de `http://www.informatica.us.es` (fecha de consulta: 24 de Noviembre de 2014)

Bezos, J. (2007). The titlesec and titletoc packages. *TexEmplares*, *8*, 283–298.

de Sousa, J. M. (2004). *Ortografía y ortotipografía del español actual*. Trea.