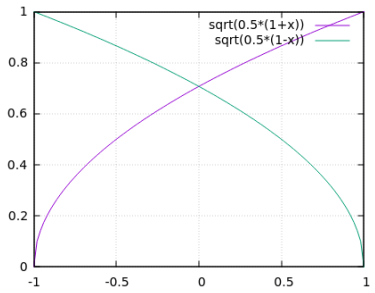
Sonido en videojuegos

FMOD Core API

1. Utilizar FMOD Core API para implementar una clase C++ de gestión y reproducción de sonidos estéreo (no posicionales) de modo que cada sonido será se gestionará a través de un objeto. La constructora tomará como argumento el archivo de sonido a reproducir y cargará ese sonido (sin reproducirlo aún). Debe incluir además los métodos reproducir, parar, pausar, modificar volumen y modificar el panorama, con el comportamiento natural.

   A continuación extender la clase con dos nuevos métodos para hacer *fadein* y *fadeout* con el sonido de manera manual. Estos métodos tomarán como argumento los milisegundos de demora para completar la operación y aplicarán una envolvente para hacer una variación lineal de volumen en dicho intervalo temporal. Con una idea similar puede implementarse un método que tome como argumento un nuevo sonido y un intervalo de tiempo, y haga un fundido de ambos sonidos (fadeout con el antiguo, fadein con el nuevo). Se puede mejorar la calidad de los fades con *equal-power crossfades* (véase `https://dsp.stackexchange.com/questions/14754/equal-power-crossfade`), con curvas como estas (la curva verde representa el fadeout de un sonido y la rosa el fadein del otro):

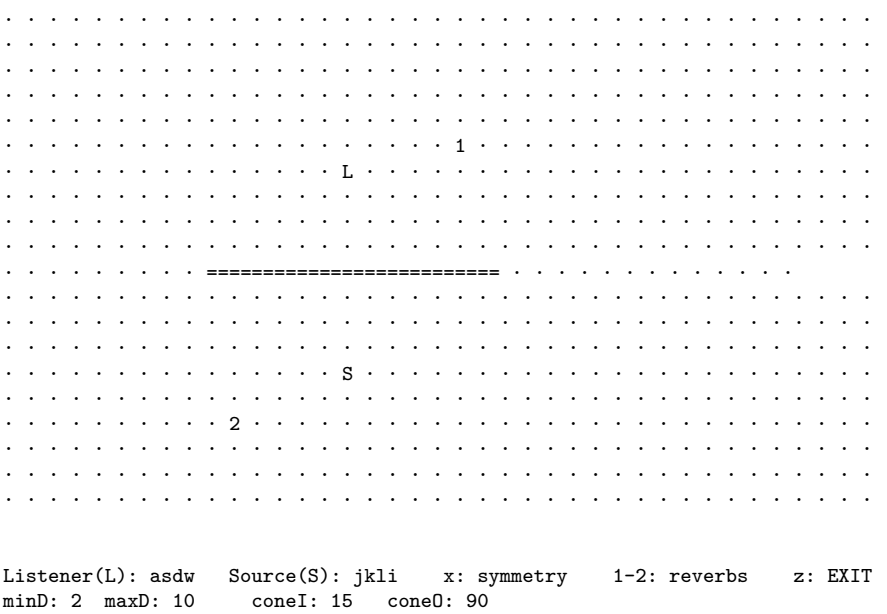
   Por último, pueden implementarse versiones alternativas para estos métodos utilizando los métodos de FMOD `addFadePoint` (consultar documentación de FMOD).

2. En este ejercicio pretendemos crear una escena sonora en el plano para experimentar con el posicionamiento 3D elemental con FMOD. Situaremos el oyente y una fuente sonora en el plano, y los moveremos con los controles habituales “asdw” y “jkli”, respectivamente. Para ello se hará un bucle elemental que que procese el input y haga un renderizado sencillo en consola. Para el sonido podemos utilizar la muestra del archivo *footstep.wav* reproducida en loop.

   A continuación:

   - Utilizar conos para orientar la fuente sonora e incluir controles para incrementar o decrementar los ángulos interior y exterior de los mismos.
   - Incluir inputs para modificar los parámetros `minDistance` y `maxDistance` de la fuente.
   - Permitir que la fuente se mueva de una posición a su simétrica (con respecto a un eje o al centro de coordenadas) en intervalos de tiempo regulares para experimentar el efecto doppler.
   - Utilizar la clase `Geometry` para incluir un cuadrado en la escena que pueda interponerse entre oyente y emisor, y experimentar el fenómeno de obstrucción con distintos parámetros.
   - Incluir dos reverb posicionales a ambos lados del cuadrado para experimentar la mezcla de reverbs correspondientes ambos recintos acústicos.

   El renderizado en consola puede tener el siguiente aspecto:



Donde `L` representa el oyente, `S` la fuente, 1 y 2 los centros de las reverbs y los caracteres "`=`" el muro (cuadrado). A continuación añadir más muros y más reverbs, y experimentar con distintas localizaciones de las mismas.