ITUPublicaciones
Recomendaciones

Unión Internacional de Telecomunicaciones
Sector de Radiocomunicaciones

Recomendación UIT-R BS.2127-1

(11/2023)

Serie BS: Servicio de radiodifusión (sonora)

Reproductor de modelo de definición de audio para sistemas de sonido avanzados

Prólogo

El Sector de Radiocomunicaciones tiene como cometido garantizar la utilización racional, equitativa, eficaz y económica del espectro de frecuencias radioeléctricas por todos los servicios de radiocomunicaciones, incluidos los servicios por satélite, y realizar, sin limitación de gamas de frecuencias, estudios que sirvan de base para la adopción de las Recomendaciones UIT-R.

Las Conferencias Mundiales y Regionales de Radiocomunicaciones y las Asambleas de Radiocomunicaciones, con la colaboración de las Comisiones de Estudio, cumplen las funciones reglamentarias y políticas del Sector de Radiocomunicaciones.

Política sobre Derechos de Propiedad Intelectual (IPR)

La política del UIT-R sobre Derechos de Propiedad Intelectual se describe en la Política Común de Patentes UIT-T/UIT-R/ISO/CEI a la que se hace referencia en la Resolución UIT-R 1. Los formularios que deben utilizarse en la declaración sobre patentes y utilización de patentes por los titulares de las mismas figuran en la dirección web http://www.itu.int/ITU-R/go/patents/es, donde también aparecen las Directrices para la implementación de la Política Común de Patentes UIT-T/UIT-R/ISO/CEI y la base de datos sobre información de patentes del UIT-R sobre este asunto.

Series de las Recomendaciones UIT-R

(También disponible en línea en https://www.itu.int/publ/R-REC/es)

Series	Título
BO	Distribución por satélite
BR	Registro para producción, archivo y reproducción; películas en televisión
BS	**Servicio de radiodifusión (sonora)**
BT	Servicio de radiodifusión (televisión)
F	Servicio fijo
M	Servicios móviles, de radiodeterminación, de aficionados y otros servicios por satélite conexos
P	Propagación de las ondas radioeléctricas
RA	Radioastronomía
RS	Sistemas de detección a distancia
S	Servicio fijo por satélite
SA	Aplicaciones espaciales y meteorología
SF	Compartición de frecuencias y coordinación entre los sistemas del servicio fijo por satélite y del servicio fijo
SM	Gestión del espectro
SNG	Periodismo electrónico por satélite
TF	Emisiones de frecuencias patrón y señales horarias
V	Vocabulario y cuestiones afines

***Nota**: Esta Recomendación UIT-R fue aprobada en inglés conforme al procedimiento detallado en la Resolución UIT-R 1.*

Publicación electrónica
Ginebra, 2024

RECOMENDACIÓN UIT-R BS.2127-1*

Reproductor de modelo de definición de audio para sistemas de sonido avanzados

(Cuestión UIT-R 135-2/6)

(2019-2023)

Cometido

En esta Recomendación se especifica el reproductor de referencia para los sistemas de sonido avanzados, incluido el intercambio de programas, especificados en la Recomendación UIT-R BS.2051-2 y los metadatos de audio especificados en el modelo de definición de audio (ADM) de la Recomendación UIT-R BS.2076-1. El reproductor de audio convierte un conjunto de señales de audio con metadatos asociados en una configuración de señales de audio y metadatos diferente basándose en los metadatos del contenido y en los metadatos del entorno local.

NOTA – Se están preparando directrices explicativas sobre la utilización del reproductor.

Palabras clave

ADM, AdvSS, audio multicanal, audio por canales, audio por escenas, audio por objetos, metadatos, modelo de definición de audio, reproductor, sistema de sonido avanzado

La Asamblea de Radiocomunicaciones,

considerando

a) que en la Recomendación UIT-R BS.1909 – *Requisitos de calidad de funcionamiento para un sistema de sonido estereofónico multicanal avanzado para uso con o sin acompañamiento de imagen* se especifican los requisitos para un sistema de sonido avanzado para uso con o sin acompañamiento de imagen;

b) que en la Recomendación UIT-R BS.2051 – *Sistemas de sonido avanzados para la producción de programas* se especifica un sistema de sonido avanzado que es un sistema con una configuración de reproducción superior a las especificadas en la Recomendación UIT-R BS.775 o un sistema con una configuración de reproducción que admita señales de entrada basadas en canales, objetos o escenas, o su combinación con metadatos;

c) que en la Recomendación UIT-R BS.2076 – *Modelo de definición de audio* se especifica la estructura de un modelo de metadatos que permite describir de manera fiable el formato y el contenido de los archivos de audio;

d) que la Recomendación UIT-R BS.2094 – *Definiciones comunes para el modelo de definición de audio* contiene las definiciones comunes para el modelo de definición de audio;

e) que en la Recomendación UIT-R BS.2125 – *Representación en serie del modelo de definición de audio* se especifica un formato de metadatos basado en el modelo de definición de audio y segmentado en series temporales de tramas;

f) que la reproducción de sistemas de sonido avanzados exige la reproducción de los metadatos asociados a las señales para presentar el contenido a una de las configuraciones de altavoz de la Recomendación UIT-R BS.2051;

---

* Esta Recomendación ha de ponerse en conocimiento de ISO, CEI, SMPTE y ETSI.

g) que los usuarios de sistemas de sonido avanzados deben poder elegir libremente el método de reproducción;

h) que es conveniente que haya una especificación abierta de un único método de reproducción de referencia que pueda utilizarse para programas de sistemas de sonido avanzados;

i) que el reproductor de referencia único debe permitir a los productores y radiodifusores de contenido supervisar y efectuar controles de calidad durante la producción del contenido, verificar la utilización de metadatos y garantizar la interoperabilidad con otros elementos de la cadena de producción,

recomienda

1 que los métodos de reproducción descritos en el Anexo 1 se consideren la referencia para la interpretación de los metadatos ADM especificados en la Recomendación UIT-R BS.2076-1 y de las señales de audio acompañantes;

2 que la Nota 1 siguiente se considere parte de esta Recomendación.

NOTA 1 – La observancia de esta Recomendación es voluntaria. Ahora bien, la Recomendación puede contener ciertas disposiciones obligatorias (para asegurar, por ejemplo, la aplicabilidad o el interfuncionamiento), por lo que la observancia se consigue con el cumplimiento exacto y puntual de todas las disposiciones obligatorias. El término «deberá» o cualquier otra palabra que conlleve la idea de obligatoriedad, como «tendrá que», así como los equivalentes correspondientes de negación se emplean para formular los requisitos. El hecho de que se utilice esta formulación no entraña la observancia parcial o total de la presente Recomendación.

Anexo 1

Especificación del reproductor ADM para sistemas de sonido avanzados

ÍNDICE

Página

Anexo 1 – Especificación del reproductor ADM para sistemas de sonido avanzados 2

1 Introducción 5

1.1 Abreviaturas/glosario 5

2 Convenios 5

2.1 Notación 5

2.2 Sistema de coordenadas 6

3 Estructura 7

3.1 Comportamiento del entorno objetivo 7

Página

4 Interfaz ADM-XML 8
4.1 AudioBlockFormat 9
4.2 Subelementos de posición 9
4.3 TypeDefinition 9
5 Elementos reproducidos 9
5.1 Estructuras de metadatos 9
5.2 Determinación de los elementos reproducidos 12
5.3 Procesamiento de elementos reproducidos 21
6 Componentes del reproductor compartidos 23
6.1 Panoramizador de fuente puntual polar 23
6.2 Determinar si el ángulo está dentro del margen de tolerancia 31
6.3 Determinar si un canal es un canal LFE a partir de sus metadatos de frecuencia 32
6.4 Canal de procesamiento de bloques 33
6.5 Interpretación genérica de los metadatos de temporización 34
6.6 Interpretación de `TrackSpecs` 35
6.7 Ángulo relativo 36
6.8 Transformación de coordenadas 36
7 Reproducción de elementos con typeDefinition==Objects 37
7.1 Estructura 37
7.2 InterpretObjectMetadata 37
7.3 Calculador de ganancia 39
7.4 Filtros de descorrelación 67
8 Reproducción de elementos con typeDefinition==DirectSpeakers 68
8.1 Reglas de correspondencia 68
8.2 Determinación de LFE 69
8.3 Correspondencia de etiquetas de altavoz 69
8.4 Fijación de bordes de pantalla 69
8.5 Correspondencia de límites 70
9 Reproducción de elementos con typeDefinition==HOA 70
9.1 Formatos HOA soportados 70

Página

9.2 Subelementos no soportados 71
9.3 Reproducción de señales HOA por altavoces 71
10 Conversión de metadatos 73
10.1 Conversión de *position* 74
10.2 Conversión de extensión 77
10.3 Conversión objectDivergence 79
11 Estructuras de datos y cuadros 79
11.1 Estructuras de metadatos internas 79
11.2 Posiciones de altavoces alocéntricas 81
11.3 Correspondencia de datos para DirectSpeakers 86
Bibliografía 91
Adjunto 1 del Anexo 1 (informativo) – Tabla de referencia a las partes correspondientes de la especificación de metadatos ADM 92
A1.1 Metadatos ADM en el reproductor ADM del UIT-R 92
Adjunto 2 del Anexo 1 (informativo) – Configuración de altavoces virtuales alternativa 93
A2.1 Especificación de una configuración de altavoces virtuales alternativa 93

1 Introducción

En esta Recomendación se describe un reproductor de audio que proporciona una interpretación completa de los metadatos del modelo de definición de audio (ADM, *audio definition model*), especificado en la Recomendación UIT-R BS.2076-1. Se recomienda la utilización de metadatos ADM para describir formatos de audio utilizados en la producción de programas para sistemas de sonido avanzados (AdvSS, *advanced sound systems*), también conocidos como sistemas de audio de la próxima generación (NGA, *next-generation audio*). Este reproductor puede reproducir señales de audio con todas las configuraciones de altavoz especificadas en la Recomendación UIT-R BS.2051-2.

Esta especificación se acompaña de una implementación de referencia de código abierto, escrita en Python para el procesamiento de ADM por ficheros, disponible en:

https://www.itu.int/dms_pub/itu-r/oth/0a/07/R0A0700003E0001ZIPE.zip.

En esta especificación se describe el código de referencia.

1.1 Abreviaturas/glosario

ADM	Modelo de definición de audio (*audio definition model*)
BMF	Formato de intercambio de metadatos de radiodifusión (*broadcast metadata exchange format*)
BW64	Formato de onda de radiodifusión 64 (*broadcast wave 64 format*)
BWF	Formato de onda de radiodifusión (*broadcast wave format*)
HOA	Ambisonía de alto orden (*higher-order ambisonics*)
NGA	Audio de la próxima generación (*next generation audio*)
PSP	Panoramizador de fuente puntual (*point source panner*)
VBAP	Panoramización en amplitud por vectores (*vector base amplitude panning*)
XML	Lenguaje de marcaje extensible (*extensible markup language*)

2 Convenios

2.1 Notación

En esta Recomendación se utilizan los siguientes convenios:

– van en cursiva los elementos, subelementos, parámetros o atributos ADM de la Recomendación UIT-R BS.2076-1: *audioObject*;

– se utilizan caracteres monoespaciados para el código fuente (variables, funciones, clases) de la implementación de referencia: `core.point_source.PointSourcePanner`. Cabe señalar que, por mor de legibilidad, se omite el prefijo `iar.`;

– se utilizan las mayúsculas en negrita para las matrices: **X**;

– se utilizan las minúsculas en negrita para los vectores: **x**;

– se indican en subíndice los elementos de un vector. Por ejemplo, x_n denota el enésimo elemento de un vector **x**;

– se resaltan en color porciones de texto monoespaciado para describir estructuras de datos:

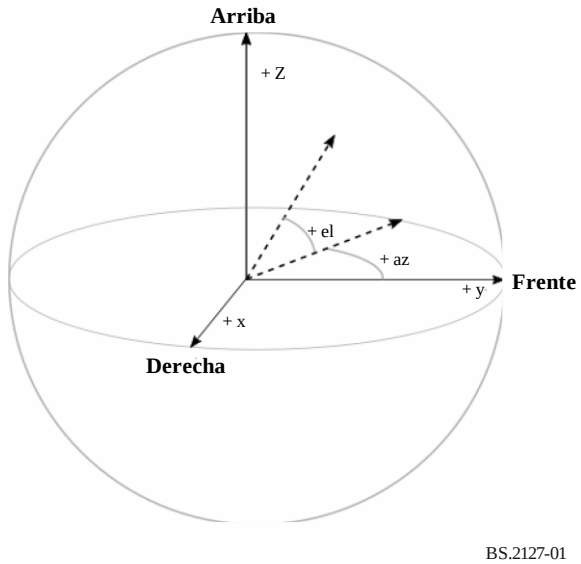
```
struct PolarPosition : Position {
  float azimuth, elevation, distance = 1;
};
```

2.2 Sistema de coordenadas

A lo largo de esta Recomendación se utilizan coordenadas cartesianas y polares.

FIGURA 1

Sistema de coordenadas



Las coordenadas polares se especifican de conformidad con la Recomendación UIT-R BS.2076-1 de la siguiente manera:

– Acimut, denotado por φ, es el ángulo en el plano horizontal de 0 grados al frente y valores positivos en sentido levógiro.

– Elevación, denotada por θ, es el ángulo por encima del plano horizontal de cero grados al frente y valores positivos hacia arriba.

Las coordenadas cartesianas se especifican de conformidad con la Recomendación UIT-R BS.2076-1 de la siguiente manera:

– el eje Y positivo apunta al frente;

– el eje X positivo apunta a la derecha;

– el eje Z positivo apunta hacia arriba.

El descodificador de ambisonía de alto orden (HOA, *high order ambisonics*) especificado en el § 9 utiliza el sistema de coordinadas y la notación HOA especificados en la Recomendación UIT-R BS.2076-1, donde:

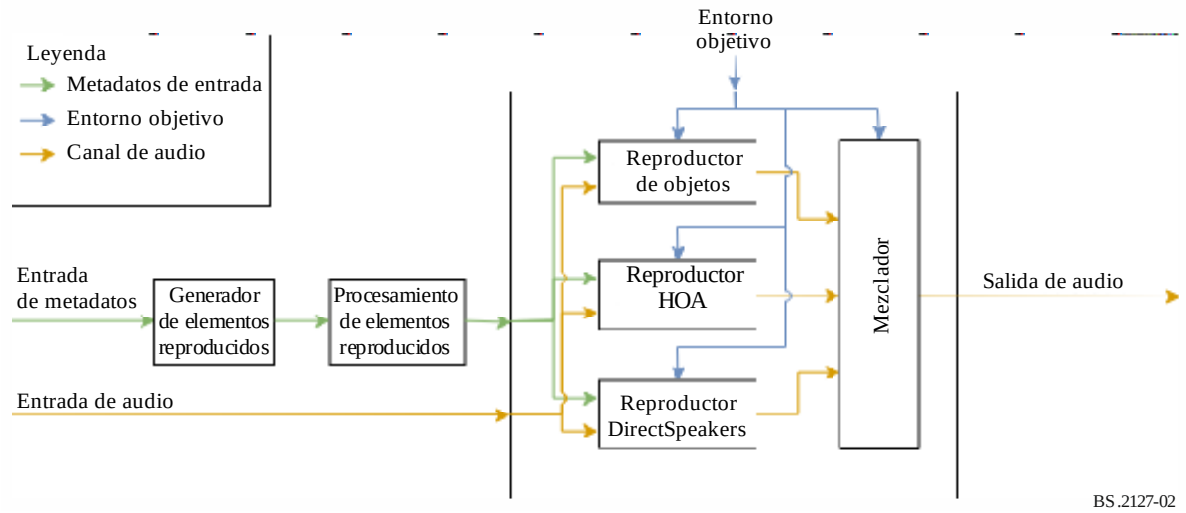
– Elevación, denotada por θ, es el ángulo, en radianes, con respecto al eje Z positivo.

– Acimut, denotado por φ, es el ángulo en el plano horizontal, en radianes, con valor 0 al frente y valores positivos en sentido levógiro.

3 Estructura

FIGURA 2

Ilustración global de la arquitectura



La arquitectura global está compuesta por varios componentes fundamentales y etapas de procesamiento, que se describen en los capítulos siguientes de esta Recomendación:

– La transformación de datos ADM en un conjunto de elementos reproducibles se describe en el § 5.2.

– En el § 5.3 se expone el procesamiento optativo para aplicar la emulación de importancia y conversión a los elementos reproducidos.

– La reproducción en sí se divide en subcomponentes basados en el tipo (*typeDefinition*) del elemento:

  • la reproducción del contenido por objetos se describe en el § 7;

  • la reproducción de las señales DirectSpeaker se describe en el § 8;

  • la reproducción HOA se describe en el § 9;

  • en el § 6 se describen las partes comunes a todos los componentes.

No se muestra en la Figura el procesamiento tipo *Matrix*, pues este tipo interviene durante la creación de los elementos reproducidos y como parte de los reproductores para otros tipos.

3.1 Comportamiento del entorno objetivo

En el momento de inicialización el usuario puede seleccionar una configuración de altavoz de entre las especificadas en la Recomendación UIT-R BS.2051.

La posición nominal de cada altavoz (`polar_nominal_position`) es la especificada en la Recomendación UIT-R BS.2051. Los acimutes nominales de `M+SC` y `M−SC` serán 15° y −15°.

El usuario puede especificar la posición real de cada altavoz (`polar_position`). De desconocerse este dato, se utilizará la posición nominal. Se verificará la conformidad de las posiciones reales con las gamas especificadas en UIT-R BS.2051; de no ser conformes, se emitirá un error. Además, el acimut absoluto tanto de `M+SC` como de `M−SC` debe situarse entre 5° y 25° o entre 35° y 60°.

4 Interfaz ADM-XML

ADM es un modelo de metadatos genérico que puede representarse naturalmente como un documento XML. En las siguientes subcláusulas se explica la correspondencia entre ADM y las estructuras de datos internos que se utilizan a lo largo de esta Recomendación y son conformes con las estructuras de datos utilizadas por la implementación de referencia.

Cabe señalar que, a pesar de que XML sea la forma habitual y común de representar los metadatos ADM, el reproductor no se limita a esa representación.

La correspondencia entre ADM y las estructuras de datos internas sigue una serie de normas simples, que se describen a continuación. Además de reglas, también hay algunas excepciones, que se describen en las subcláusulas siguientes.

– Todos los elementos ADM se representan como una subclase derivada de `ADMElement` con la firma:

```
class ADMElement {
  string id;
  ADM adm_parent;
  bool is_common_definition;
};
```

– Cada clase de elemento ADM se ampliará con todos los atributos y subelementos ADM, que se corresponden con atributos de clase.

– Si un subelemento contiene más de un valor, es una clase en sí mismo. Por ejemplo, el subelemento *jumpPosition* es una clase con la firma:

```
class JumpPosition {
  bool flag;
  float interpolationLength;
};
```

– Durante el análisis sintáctico de XML, las referencias a otros elementos ADM se almacenan como ID simples utilizando el nombre del subelemento como nombre de atributo (por ejemplo, `AudioObject.audioPackFormatIDRef`). Para simplificar el acceso ulterior, estas referencias se resuelven en un paso posterior, donde los elementos resueltos se añaden directamente a cada estructura de datos (`AudioObject.audioPackFormats`).

Siguiendo estas reglas, la firma completa del elemento `AudioContent` se representa de la siguiente manera:

```
class AudioContent : ADMElement {
  string audioContentName;
  string audioContentLanguage;
  LoudnessMetaData;
  int dialogue;
  vector<AudioObject*> audioObjects;
  vector<string> audioObjectIDRef;
};
```

Los elementos ADM principales y sus clases propias se implementan en `fileio.adm.elements.main_elements`. La resolución de la referencia se implementa en cada clase (en `ADM` y en cada elemento ADM principal) como el método `lazy_lookup_references`.

El análisis sintáctico y la escritura del ADM se implementan en `fileio.adm.xml`.

4.1 AudioBlockFormat

audioBlockFormat difiere de otros elementos ADM en que sus subelementos y atributos son distintos en función de *typeDefiniton*. Para reflejar esta particularidad, `AudioBlockFormat` se divide en múltiples clases, una para cada *typeDefinition* soportado: `AudioBlockFormatObjects`, `AudioBlockFormatDirectSpeakers` y `AudioBlockFormatHoa`.

Esto se implementa en `fileio.adm.elements.block_formats`.

4.2 Subelementos de posición

Las posiciones se representan mediante múltiples subelementos *position* en el ADM. Para simplificar el tratamiento interno, los valores de estos subelementos se combinan en un único atributo dentro de la representación de `AudioBlockFormat`.

Para *typeDefinition==Objects* se trata de `ObjectPolarPosition` o de `ObjectCartesianPosition`, en función del sistema de coordenadas utilizado. Para *typeDefinition==DirectSpeakers* se trata de `DirectSpeakerPolarPosition` o de `DirectSpeakerCartesianPosition`.

4.3 TypeDefinition

Los atributos *typeDefinition* y *typeLabel* describen una única propiedad. Por ese motivo, a nivel interno sólo se utilizará una entidad para representarlos.

```
enum TypeDefinition {
  DirectSpeakers = 1;
  Matrix = 2;
  Objects = 3;
  HOA = 4;
  Binaural = 5;
};

enum FormatDefinition {
  PCM = 1;
};
```

5 Elementos reproducidos

`RenderingItem` es una representación de un elemento ADM que se ha de reproducir y contiene toda la información necesaria para ello. Por consiguiente, un elemento representará un único *audioChannelFormat* o un grupo de *audioChannelFormats*. Dado que cada *typeDefinition* tiene sus propios requisitos, es necesario disponer de distintas estructuras de metadatos para que cada *typeDefinition* las adapte a sus necesidades específicas.

En la siguiente cláusula se describen más detalladamente las estructuras de metadatos utilizadas.

5.1 Estructuras de metadatos

`RenderingItems` se construye a partir de las siguientes clases base:

– `TypeMetadata` para contener todos los parámetros (posiblemente variables en el tiempo) necesarios para la reproducción del elemento;

– `MetadataSource` para contener una serie de objetos `TypeMetadata`; y

– `RenderingItem` para asociar `MetadataSource` con una fuente de muestras de audio e información adicional, no necesariamente exigida por el reproductor.

Dado que cada *typeDefinition* tiene requisitos distintos, `TypeMetadata` y `RenderingItem` han de ser subclases para que cada *typeDefinition* los adapte a sus necesidades específicas. `MetadataSource` es independiente de *typeDefinition*. Los datos comunes se aúnan en `ExtraData`:

```
struct ExtraData {
  optional<duration> object_start;
  optional<duration> object_duration;
  ReferenceScreen reference_screen;
  Frequency channel_frequency;
};
```

Los datos de importancia se deben almacenar en una estructura `ImportanceData`:

```
struct ImportanceData {
  optional<int> audio_object;
  optional<int> audio_pack_format;
};
```

Las referencias a las muestras de audio de entrada se encapsularán en estructuras `TrackSpec` para permitir la especificación de pistas silenciosas y el procesamiento de Matrix. `DirectTrackSpec` especifica que las muestras deben leerse directamente de la pista de entrada indicada. `SilentTrackSpec` especifica que las muestras deben ser todas cero.

```
struct TrackSpec {};

struct DirectTrackSpec : TrackSpec {
  int track_index;
};

struct SilentTrackSpec : TrackSpec {
};
```

Se proporcionan dos tipos de `TrackSpec` para soportar *typeDefinition==DirectSpeakers*. `MatrixCoefficientTrackSpec` especifica que los parámetros especificados en `coefficient` (de un elemento *audioBlockFormat* `coefficient` Matrix) se aplican a todas las muestras de `input_track`, mientras que `MixTrackSpec` especifica que se han de mezclar las muestras de múltiples `TrackSpecs`.

```
struct MatrixCoefficientTrackSpec : TrackSpec {
  TrackSpec input_track;
  MatrixCoefficient coefficient;
};

struct MixTrackSpec : TrackSpec {
  vector<TrackSpec> input_tracks;
};
```

La implementación se efectúa en `core.utils.metadata_input`. En las siguientes subcláusulas se describen más detalladamente las implementaciones específicas de cada *typeDefinition*.

5.1.1 DirectSpeakers

Para *typeDefinition==DirectSpeakers* el `TypeMetadata` contendrá *audioBlockFormat,* la lista de *audioPackFormat*s que llevan al *audioChannelFormat* contenedor, además de los datos comunes aunados en `ExtraData`.

```
struct DirectSpeakersTypeMetadata : TypeMetadata {
  AudioBlockFormatDirectSpeakers block_format;
  vector<AudioPackFormat> audioPackFormats;
  ExtraData extra_data;
};
```

Dado que cada *audioChannelFormat* con *typeDefinition==DirectSpeakers* puede procesarse de manera independiente, `RenderingItem` contiene un único `TrackSpec`.

```
struct DirectSpeakersRenderingItem : RenderingItem {
  TrackSpec track_spec;
  MetadataSource metadata_source;
  ImportanceData importance;
};
```

5.1.2 Matrix

typeDefinition==Matrix se soportará utilizando el mecanismo `TrackSpec` al reproducir elementos para otros tipos, por lo que no se necesitan clases `MatrixTypeMetadata` o `MatrixRenderingItem` explícitas.

5.1.3 Objetos

`ObjectTypeMetadata` contendrá un *audioBlockFormat* además de los datos comunes aunados en `ExtraData`.

```
struct ObjectTypeMetadata : TypeMetadata {
  AudioBlockFormatObjects block_format;
  ExtraData extra_data;
};
```

Dado que cada *audioChannelFormat* con *typeDefinition==Objects* puede procesarse de manera independiente, `RenderingItem` contendrá sólo un único `TrackSpec`.

```
struct ObjectRenderingItem : RenderingItem {
  TrackSpec track_spec;
  MetadataSource metadata_source;
  ImportanceData importance;
};
```

5.1.4 HOA

Para *typeDefinition==HOA* la situación es distinta que para *typeDefinition==DirectSpeakers* y *typeDefinition==Objects*, porque se han de procesar varios *audioChannelFormats* juntos. Por ese motivo, `HOATypeMetadata` no contiene un `audioBlockFormat` además de `ExtraData`, sino que la información necesaria se extrae de *audioBlockFormats* y se almacena directamente en `HOATypeMetadata`.

```
struct HOATypeMetadata : TypeMetadata {
  vector<int> orders;
  vector<int> degrees;
  optional<string> normalization;
  optional<float> nfcRefDist;
  bool screenRef;
  ExtraData extra_data;
  optional<duration> rtime;
  optional<duration> duration;
};
```

Por ese mismo motivo, el caso de `HOARenderingItem` es diferente. En este caso, `HOARenderingItem` no contiene sólo un `TrackSpec`, sino un vector de `TrackSpecs`.

```
struct HOARenderingItem : RenderingItem {
  vector<TrackSpec> track_specs;
  MetadataSource metadata_source;
  vector<ImportanceData> importances;
};
```

5.1.5 Binaural

Como *typeDefinition==Binaural* no se soporta, no existen las clases `BinauralTypeMetadata` o `BinauralRenderingItem`.

5.2 Determinación de los elementos reproducidos

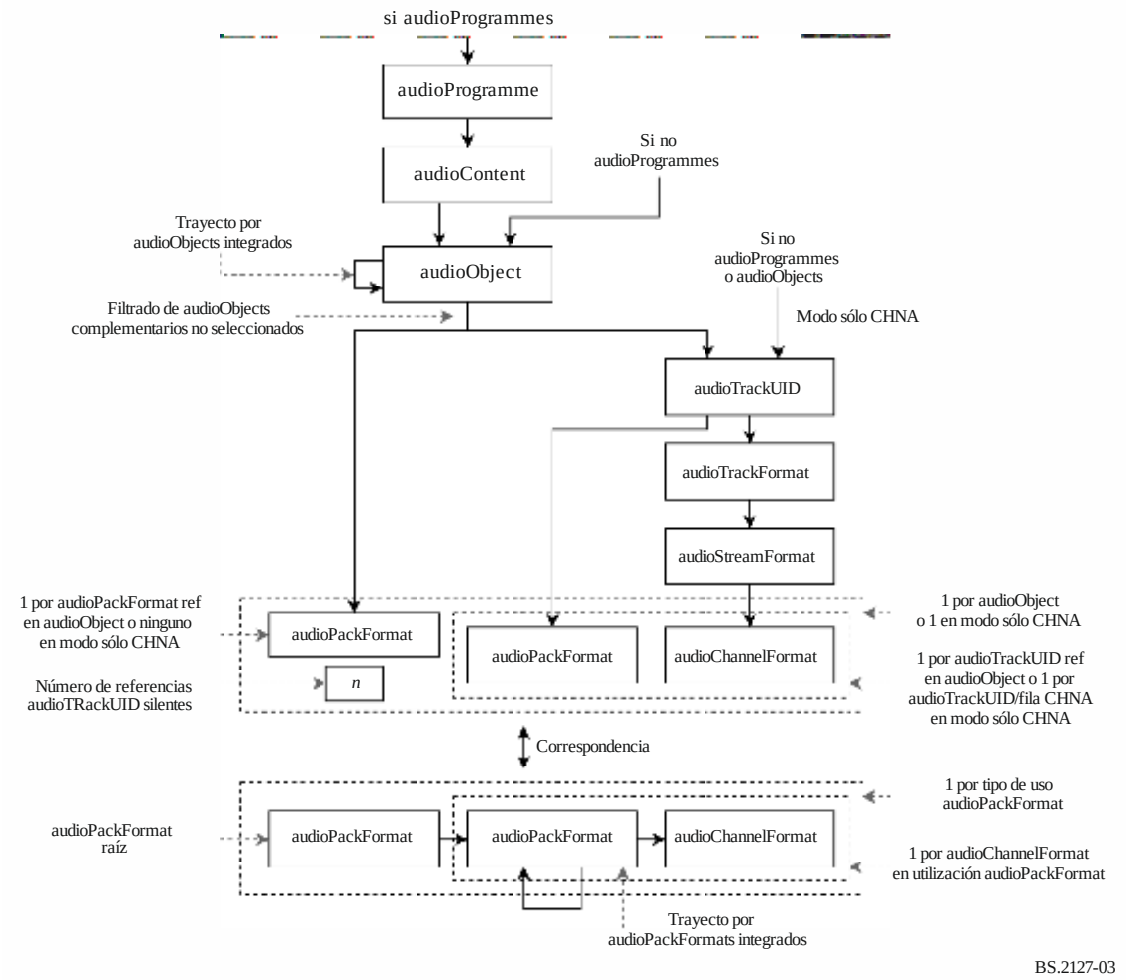
Para determinar `RenderingItems`, es necesario analizar la estructura ADM. En la Fig. 3 se ilustra este procedimiento.

El estado del proceso de selección del elemento atraviesa varios componentes en un único objeto, denominado «estado de selección de elemento», que, cuando está completo, representa todos los componentes que forman un único `RenderingItem`. Cada componente acepta un único estado de selección de elemento y devuelve copias (entre cero y muchas) del mismo con más entradas completas. Estos pasos se reúnen en `select_rendering_items`, un bucle integrado a lo largo de los estados cada vez que un componente lo modifica.

La implementación se efectúa en `core.select_items`.

FIGURA 3

Análisis de la estructura ADM para determinar RenderingItems



5.2.1 Inicio

La selección de un elemento reproducido puede iniciarse en múltiples puntos de la estructura ADM, en función de los elementos incluidos en el fichero.

Si hay elementos *audioProgramme*, se selecciona un único *audioProgramme*. Si hay elementos *audioObject*, se seleccionarán todos los *audioObject*s. Si no, se seleccionarán todos los *audioTrackUID*s (filas CHNA) (esto se denomina «modo sólo CHNA»).

5.2.2 Selección de audioProgramme

Sólo se selecciona un *audioProgramme*. El usuario puede seleccionar el programa que se va a utilizar, de no seleccionarse un *audioProgramme*, se seleccionará el que tenga el ID numéricamente más bajo.

5.2.3 Selección de audioContent

Se seleccionan todos los *audioContent*s referenciados por el *audioProgramme* seleccionado.

5.2.4 Selección de audioObject

`audioObjects` se pondrá a todos los trayectos posibles por la jerarquía *audioObject* a partir del *audioContent* seleccionado (tras los enlaces *audioObject*).

5.2.5 Tratamiento de *audioObject* complementarios

Las referencias *audioComplementaryObject* se interpretarán como grupos definitorios de *audioObjects*, de los cuales sólo se reproducirá un *audioObject*.

Un grupo se describe mediante referencias *audioComplementaryObject* a partir del *audioObject* por defecto del grupo hasta los *audioObject*s no por defecto del grupo. El usuario puede proporcionar un conjunto de *audioObject*s que seleccionar, lo que prima sobre los valores por defecto. A partir de ahí, se determina el conjunto de *audioObject*s que ignorar y se descartan los estados si en ese conjunto está alguno de los *audioObject*s de trayecto *audioObject*.

5.2.5.1 Selección de *audioObjects* complementarios que ignorar

En primer lugar se aumentará el conjunto de *audioObject*s seleccionado por el usuario con los valores por defecto de cada grupo. Para cada *audioObject* raíz (un *audioObject* con referencias *audioComplementaryObject*), si ninguno de los *audioObject*s del grupo definido por el *audioObject* raíz de ese grupo está en el conjunto, se añadirá el *audioObject* raíz (por defecto).

Por tanto, el conjunto de *audioObject*s que ignorar es el conjunto de todos los audioObjects complementarios (es decir, los *audioObject*s con una referencia *audioComplementaryObject* y audioObjects a los que apunta una referencia *audioComplementaryObject*) menos el conjunto aumentado de *audioObject*s seleccionado por el usuario.

Si se seleccionan *audioObject*s que no pertenecen a ningún grupo complementario o se seleccionan múltiples *audioObject*s de un único grupo *audioObject* (ya los seleccione el usuario o se seleccionen como resultado del solapamiento de grupos), se generará un error.

5.2.6 Correspondencia de *audioPackFormat*

El siguiente paso consiste en la correspondencia entre la información de un *audioObject* (la lista de *audioPackFormat*s, *audioTrackUID*s y el número de pistas silentes o simplemente la lista de todos los *audioTrackUID*s en modo sólo CHNA) y las estructuras *audioPackFormat* y *audioChannelFormat*.

Esto se especifica como un problema de correspondencia/búsqueda, más que como trayectos específicos por las estructuras de referencia que se han de resolver, porque hay múltiples elementos en ambos lados que han de corresponder y no entrar en conflicto para hallar una solución válida.

La correspondencia sólo se considera válida si se encuentra exactamente una solución. De no encontrarse soluciones, los metadatos son contradictorios y se generará un error. De encontrarse múltiples soluciones, los metadatos son ambiguos y se generará un error. En ambos casos, se ejecutará un diagnóstico para mostrar al usuario todas las causas posibles de error.

5.2.6.1 Packs para la correspondencia

Los *audioPackFormat*s con los que se ha de establecer la correspondencia se especifican como una lista de estructuras `AllocationPack`:

```
struct AllocationChannel {
  AudioChannelFormat channel_format;
  vector<AudioPackFormat> pack_formats;
};

struct AllocationPack {
  AudioPackFormat root_pack;
  vector<AllocationChannel> channels;
};
```

Cada una especificará el *audioPackFormat* raíz (`root_pack`, *audioPackFormat* de nivel superior que referencia todos los canales por atribuir) y una lista de los canales para la correspondencia con

ese canal. Cada canal es una combinación de una referencia *audioChannelFormat* y una lista de los posibles *audioPackFormat*s a los que se puede asociar ese canal.

Para cada *audioPackFormat* `pack` en que *typeDefinition != Matrix*, se crea un objeto `AllocationPack` donde:

– `root_pack` es `pack`.

– `channels` tiene una entrada para cada *audioChannelFormat* accesible desde `pack` (recursivamente siguiendo los enlaces *audioPackFormat*), donde `pack_formats` contiene todos los *audioPackFormat*s del trayecto entre `pack` y *audioChannelFormat* (incluido `pack`).

Aunque se trata de una ligera simplificación de la estructura *audioPackFormat* y *audioChannelFormat*, la ventaja de esta representación es su capacidad para representar las estructuras de referenciación *audioPackFormat* y *audioChannelFormat* utilizadas con el contenido *Matrix*, descrito a continuación.

5.2.6.1.1 Tratamiento de Matrix

Los *audioPackFormats* Matrix pueden referenciarse de múltiples maneras, dependiendo del efecto que se desee producir. Estas estructuras de referencia se reflejan en los siguientes `AllocationPacks`, que se producen para cada `pack` audioPackFormat con *typeDefinition==Matrix*:

– Si `pack` es una matriz directa o de descodificación, la matriz debe aplicarse si un *audioObject* referencia tanto el `pack` como un conjunto de *audioTrackUID*s que, a su vez, referencian el `pack` y los canales del *audioPackFormat* de entrada o de codificación del `pack`:

  • `root_pack` es un pack.

  • `channels` contiene un valor por cada canal *audioChannelFormat* en el *audioPackFormat* de entrada del pack (ya sea el *encodePackFormat* o el *inputPackFormat,* en función del tipo) donde `channel_format` es `channel` y `pack_formats` es [`pack`].

– Si `pack` es una matriz directa o de descodificación, se ha de considerar que la matriz se ha aplicado previamente a las muestras del fichero si un *audioObject* referencia tanto `pack` como un conjunto de *audioTrackUID*s que, a su vez, referencian el `pack` (o los subpacks) y los canales de `pack`:

  • `root_pack` es `pack`.

  • `channels` contiene un valor por *audioChannelFormat* `channel` del `pack`, donde `channel_format` es `channel` y `pack_formats` contiene todos los *audioPackFormat*s del trayecto entre `pack` y `channel`.

– Si `pack` es una matriz de descodificación, es posible aplicar su *encodePackFormat* seguido de `pack` si un *audioObject* referencia `pack` y un conjunto de *audioTrackUID*s que, a su vez, referencian *encodePackFormat* y los canales del *inputPackFormat* de *encodePackFormat*:

  • `root_pack` es `pack`.

  • `channels` contiene un valor por cada audioChannelFormat `channel` del inputPackFormat del encodePackFormat de `pack`, donde `channel_format` es `channel` y `pack_formats` contiene todos los audioPackFormats del trayecto entre inputPackFormat y `channel`.

El «tipo» de una matriz *audioPackFormat* se determina utilizando las reglas siguientes:

– Si tiene tanto una referencia *inputPackFormat* como *outputPackFormat*, es una matriz directa.

– Si tiene una referencia *inputPackFormat*, pero no una referencia *outputPackFormat*, es una matriz de codificación.

– Si tiene una referencia *outputPackFormat*, pero no una referencia *inputPackFormat*, es una matriz de descodificación.

– Si no tiene ni referencia *inputPackFormat* ni referencia *outputPackFormat*, se genera un error.

5.2.6.2 Pistas y referencias *audioPackFormat* para la correspondencia

Las pistas cuya correspondencia con `AllocationPacks` se ha de establecer se especifican por medio de tres valores:

– `tracks`, una lista de `AllocationTracks`, cada una de las cuales representa un *audioTrackUID* (o fila CHNA):

```
class AllocationTrack {
  AudioChannelFormat channel_format;
  AudioPackFormat pack_format;
};
```

`channel_format` se obtiene de un *audioTrackUID* siguiendo las referencias *audioTrackFormat*, *audioStreamFormat* y *audioChannelFormat*, mientras que `pack_format` está directamente referenciado por el *audioTrackUID*.

– `pack_refs`, lista opcional de referencias *audioPackFormat* de un *audioObject*.

`num_silent_tracks`, número de pistas «silentes» por atribuir, representado en las referencias de un *audioObject* como `ATU_00000000`.

Al determinar estas estructuras para un *audioObject*:

– `tracks` contiene una entrada para cada *audioTrackUID* (no silente) referenciado del *audioObject*.

– `pack_refs` es una lista de referencias *audioPackFormat* del *audioObject*.

– `num_silent_tracks` es el número de *audioTrackUID*s silentes referenciados (correspondiente a las referencias a `ATU_00000000` del *audioObject*).

Mientras que en el modo sólo CHNA:

– `tracks` contiene una entrada para cada *audioTrackUID* (o fila CHNA) del fichero.

– `pack_refs` es `None`.

– `num_silent_tracks` es 0.

5.2.6.3 Correspondencia

Una solución de correspondencia se especifica como una lista de objetos `AllocatedPack`:

```
struct AllocatedPack {
  AllocationPack pack;
  vector<tuple<AllocationChannel,
               optional<AllocationTrack>>> allocation;
};
```

Cada uno asocia uno de los *audioChannelFormat* de `pack` con una `track`, o una pista silente si no se especifica `AllocationTrack`.

Una solución válida tiene las siguientes propiedades:

1) Para cada `AllocatedPack`, cada canal del `AllocationPack` ocurre exactamente una vez en `allocation`.

2) Cada pista de `tracks` ocurre exactamente una vez en la salida.

3) El número de pistas silentes en la salida es igual a `num_silent_tracks`.

4) Para cada `AllocationChannel channel` y `AllocationTrack track` asociados, `track.channel_format` es `channel.channel_format` y `track.pack_format` está en `channel.pack_formats`.

5) Si `pack_refs` no es `None`, hay una correspondencia biunívoca entre `pack_refs` y los valores de `pack.pack.root_pack` para cada `AllocatedPack pack`.

Las soluciones idénticas, excepto por el orden de los `AllocationPacks` o las `allocations` que conllevan, se consideran equivalentes.

Puede utilizarse cualquier método que pueda enumerar todas las soluciones válidas y únicas (no equivalentes). En la implementación de referencia las soluciones se encuentran considerando las propiedades enumeradas como un problema de satisfacción restrictivo y enumerando todas las soluciones efectuando una búsqueda de rastreo.

5.2.6.3.1 Ejemplos

La correspondencia de formato de packs se ilustra en los siguientes ejemplos.

En primer lugar, se definen las estructuras utilizadas en los ejemplos. `c1`, `c2`, etc. y `p1`, `p2`, etc. representan las referencias a *audioChannelFormat*s y *audioPackFormat*s (pero pueden ser cualquier objeto, pues `allocate_packs` sólo utiliza información en las estructuras `Allocation...`, comparando esas referencias por identidades).

Un pack mono y la pista que lo referencia:

```
ac1 = AllocationChannel(c1, [p1])
ap1 = AllocationPack(p1, [ac1])
at1 = AllocationTrack(c1, p1)
```

Un pack de dos canales con dos pares de pistas que lo referencian:

```
ac2 = AllocationChannel(c2, [p2])
ac3 = AllocationChannel(c3, [p2])
ap2 = AllocationPack(p2, [ac2, ac3])

at2 = AllocationTrack(c2, p2)
at3 = AllocationTrack(c3, p2)

at4 = AllocationTrack(c2, p2)
at5 = AllocationTrack(c3, p2)
```

La resolución de una única pista mono en un *audioObject* resulta en una solución única con un único pack atribuido:

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[at1],
    pack_refs=[p1],
    num_silent_tracks=0,
) == [[AllocatedPack(pack=ap1, allocation=[(ac1, at1)])]]
```

La resolución de una única pista mono en modo sólo CHNA da como resultado la misma estructura:

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[at1],
    pack_refs=None,
    num_silent_tracks=0,
) == [[AllocatedPack(pack=ap1, allocation=[(ac1, at1)])]]
```

La resolución de una única pista silente da como resultado la misma estructura, excepto que la referencia a la pista se sustituye por `None`:

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[],
    pack_refs=[p1],
    num_silent_tracks=1,
) == [[AllocatedPack(pack=ap1, allocation=[(ac1, None)])]]
```

Si hay más pistas que canales disponibles en las referencias del pack, no habrá solución alguna, porque la regla 2 entra en conflicto con la regla 5:

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[at1],
    pack_refs=[],
    num_silent_tracks=0,
) == []
```

Si hay más pistas silentes que canales disponibles en las referencias del pack, no habrá solución alguna, porque la regla 2 entra en conflicto con la regla 5:

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[],
    pack_refs=[ap1],
    num_silent_tracks=2,
) == []
```

Si hay una correspondencia errónea entre las referencias del pack y la información de canal/pack en las pistas, no habrá solución alguna, pues entran en conflicto las reglas 1, 4 y 5:

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[at1, at1],
    pack_refs=[p2],
    num_silent_tracks=0,
) == []
```

Si hay múltiples instancias de un pack multicanal en un *audioObject*, la asignación de las pistas a los packs es ambigua, por lo que hay múltiples soluciones:

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[at2, at3, at4, at5],
    pack_refs=[p2, p2],
    num_silent_tracks=0,
) == [
    [AllocatedPack(pack=ap2, allocation=[(ac2, at2), (ac3, at3)]),
     AllocatedPack(pack=ap2, allocation=[(ac2, at4), (ac3, at5)])],
```

```
        [AllocatedPack(pack=ap2, allocation=[(ac2, at2), (ac3, at5)]),
         AllocatedPack(pack=ap2, allocation=[(ac2, at4), (ac3, at3)])],
    ]
```

5.2.6.4 Procesamiento ulterior de las soluciones

Cabe señalar que los resultados de la correspondencia se especifican en términos de estructuras de entrada (`AllocationPack`, `AllocationChannel`, `AllocationTrack`), y no en referencias subyacentes a estructuras ADM. Esto se hace así para permitir la correspondencia arbitraria entre las referencias de *audioPackFormat* y *audioChannelFormat* (de *audioObject* y *audioTrackUID*) y la información facilitada al reproductor, pues no hay correspondencias simples cuando se utiliza *typeDefinition==Matrix*.

En el caso de un `AllocatedPack pack` no matriz, la correspondencia es directa. `output_pack` es `pack.pack.root_pack` y hay una correspondencia biunívoca entre las atribuciones de `pack.allocation` y la atribución real de los canales: `AllocationChannel channel` se corresponde con `channel.channel_format`, `AllocationTrack track` se corresponde con `DirectTrackSpec` para el índice de pistas de *audioTrackUID* (o fila CHNA) asociado con `track`, y la `AllocationTrack` que falta se corresponde con `SilentTrackSpec`.

En el caso de un `AllocatedPack pack` matriz, la correspondencia es más compleja:

`pack.root_pack` siempre es un pack directo o de descodificación (véase el § 5.2.6.1.1), por lo que `output_pack` es `pack.root_pack.outputPackFormat`.

El canal de salida a la atribución de pistas contiene una entrada por cada *audioChannelFormat* `matrix_channel` de `root_pack`. Estos canales tienen una correspondencia biunívoca con los *audioChannelFormats* de `output_pack` establecidos por las referencias de *outputChannelFormat*.

audioChannelFormat es matrix_channel.block_formats[0].outputChannelFormat.

`TrackSpec` se construye siguiendo recursivamente las referencias de *inputChannelFormat* desde `matrix_channel` a los *audioChannelFormat*s referenciados en `pack.allocation`, integrando `MatrixCoefficientTrackSpecs` y `MixTrackSpecs` para aplicar el procesamiento especificado en los elementos *coefficient* y mezclando múltiples canales de entrada:

– Si `matrix_channel` está referenciado en `pack.allocation`, se devuelve un `DirectTrackSpec` o `SilentTrackSpec` correspondiente al `AllocationTrack` asociado (véase más arriba).

– En caso contrario, se devuelve un `MixTrackSpec` con un `MatrixCoefficientTrackSpec` para cada elemento *coefficient* `c` de `matrix_channel.block_formats[0].matrix`, que aplica el procesamiento especificado en `c` a la pista especificada para `c.inputChannelFormat`, determinado recursivamente.

En la implementación de referencia, este proceso se implementa en dos subclases de `AllocationPack`, que tienen métodos para cuestionar al *audioPackFormat* y una atribución de canal a disposición del reproductor. La asociación entre `AllocationTracks` y sus correspondientes *audioTrackUID*s se mantiene, del mismo modo, utilizando una subclase de `AllocationTrack`.

5.2.7 Elementos reproducidos de salida

Una vez determinado el *audioPackFormat* raíz y asignado un `TrackSpec` a cada uno de sus canales, toda la información encontrada se traduce en uno o más `RenderingItems`.

El proceso correspondiente depende del tipo de *audioPackFormat* raíz.

5.2.7.1 Componentes compartidos

Algunos de los datos de los elementos reproducidos son comunes a varios tipos y, por tanto, se derivan de la misma manera.

5.2.7.1.1 Importancia

Debe derivarse un objeto `ImportanceData` del estado de selección de elemento con los siguientes valores:

– `audio_object` es la importancia mínima especificada en todos los *audioObject*s del trayecto.

– `audio_pack_format` es la importancia mínima especificada en cualquier *audioPackFormat* a lo largo del trayecto entre el *audioPackFormat* raíz y el *audioChannelFormat*.

En ambos casos se considera que `None` (importancia no especificada) es la importancia más elevada.

5.2.7.1.2 Datos extra

Debe derivarse un objeto `ExtraData` del estado de selección de elemento con los siguientes valores:

– `object_start` es el tiempo *start* del último *audioObject* del trayecto (`None` en modo sólo CHNA).

– `object_duration` es la *duration* del último *audioObject* del trayecto (`None` en modo sólo CHNA).

– `reference_screen` es el *audioProgrammeReferenceScreen* del *audioProgramme* seleccionado (`None` si no se selecciona ninguno).

– `channel_frequency` es el elemento *frequency* del *audioChannelFormat* seleccionado (o `None` si no se ha seleccionado ninguno, como cuando se crea un elemento reproducido HOA).

5.2.7.2 Elementos reproducidos de salida para typeDefinition==Objects o DirectSpeakers

El proceso para determinar los elementos reproducidos para *Objects* y *DirectSpeakers* es similar, sólo cambian los tipos implicados y la selección de los parámetros.

Se produce un elemento reproducido por *audioChannelFormat* y Par `track_spec` en la atribución de canal.

Se crea un `MetadataSource`, que produce un `RenderingItem` (del tipo adecuado) por cada *audioBlockFormat* del *audioChannelFormat* seleccionado, donde el campo `extra_data` se determina como se ha expuesto anteriormente y el campo `audioPackFormats` contiene todos los *audioPackFormat*s del trayecto entre el *audioPackFormat* raíz y el *audioChannelFormat*. Esto se encapsula en un objeto `RenderingItem` (también del tipo adecuado) con `track_spec` e `importance` determinados como en el caso anterior.

5.2.7.3 Elementos reproducidos de salida para *typeDefinition==HOA*

Se produce un `HOARenderingItem` por cada atribución de pack con toda la información necesaria para reproducir un grupo de canales, que forman un flujo HOA. Esta información se reparte por múltiples *audioChannelFormats* y *audioPackFormats* (cuando están integrados), que deben ser coherentes.

Los *audioChannelFormat*s HOA deben contener únicamente un solo elemento *audioBlockFormat,* pues, en caso contrario, se genera un error.

Un único objeto `NHOATypeMetadata` se crea con los parámetros derivados conforme al Cuadro 1.

CUADRO 1

Propiedades de los parámetros `HOATypeMetadata`

parámetro `HOATypeMetadata`	parámetro *audioBlockFormat*	parámetro *audioPackFormat*	Número
`rtime`	*rtime*		Único
`duration`	*duration*		Único
`orders`	*order*		Por canal
`degrees`	*order*		Por canal
`normalization`	*normalization*	*normalization*	Único
`nfcRefDist`	*nfcRefDist*	*nfcRefDist*	Único
`screenRef`	*screenRef*	*screenRef*	Único

En primer lugar se han de determinar todos los parámetros para cada *audioChannelFormat* del *audioPackFormat* raíz. Para los parámetros que tienen tanto parámetros *audioBlockFormat* como *audioPackFormat*, el parámetro puede fijarse sólo en el *audioBlockFormat* del *audioChannelFormat* o en cualquier *audioPackFormat* del trayecto entre el *audioPackFormat* raíz y el *audioChannelFormat*. Si se encuentran múltiples copias de un parámetro para un *audioChannelFormat* determinado, éstos deberán tener el mismo valor o se generará un error. Si no se encuentran valores para un determinado parámetro y *audioChannelFormat*, se aplicarán los valores por defecto especificados en la Recomendación UIT-R BS.2076-1.

Una vez hallado *nfcRefDist* para un *audioChannelFormat* concreto, un valor 0 se traducirá a `None`, lo que implica que no se aplicará NFC. Este paso se realiza en este momento (en lugar de durante el análisis sintáctico XML) para que se considere que *nfcRefDist==0.0* entra en conflicto con *nfcRefDist==1.0,* por ejemplo.

Para los parámetros que sólo tienen un valor único (todos excepto `orders` y `degrees`), los parámetros determinados para todos los *audioChannelFormats* serán iguales o se generará un error.

`extra_data` se determina como se ha indicado para todo el *audioPackFormat*.

Se producirá un `HOARenderingItem` con una entrada en `track_specs` e `importances` por cada elemento en la atribución de canal (como se ha indicado), además de un `MetadataSource` con sólo el objeto `HOATypeMetadata` anterior.

5.3 Procesamiento de elementos reproducidos

Algunas funcionalidades del reproductor se implementan modificando la lista de elementos reproducidos seleccionados. En el § 5.3.1 se describe cómo eliminar el contenido de acuerdo con el nivel de importancia especificado y en el § 5.3.3 se explica cómo pueden emularse los efectos de la conversión de metadatos descendente.

5.3.1 Emulación de importancia

Los parámetros *importance* definidos en la Recomendación UIT-R BS.2076-1 permiten a un reproductor descartar elementos que no alcancen un cierto nivel de importancia por motivos indeterminados específicos de la aplicación.

ADM especifica los tres parámetros *importance* que se han de utilizar:

– *importance* como atributo audioObject;

– *importance* como atributo audioPackFormat;

– *importance* como atributo audioBlockFormat para typeDefinition==Object.

La diferencia más notable entre esos atributos *importance* es que la importancia *audioBlockFormat* es dependiente del tiempo, es decir, que puede variar a lo largo del tiempo, mientras que la importancia *audioObject* y *audioPackFormat* son estáticas.

Puede utilizarse un umbral distinto para cada atributo *importance*. La determinación de los valores umbral deseados se considera muy dependiente de la aplicación y el caso de uso concretos, por lo que queda fuera del alcance de una especificación de reproductor de producción. En su lugar, el reproductor ofrece medios para simular el efecto que tendría aplicar un determinado umbral de importancia a ADM, lo que permite a los productores de contenido estudiar los efectos de los valores de *importance* en la reproducción. Por consiguiente, la emulación de la importancia no forma parte del proceso de reproducción en sí, sino que se aplica a `RenderingItems` como una fase de posproducción.

5.3.1.1 Valores de importancia de `RenderingItems`

Cada elemento reproducido puede tener su propio conjunto de valores *importance* efectivos, pues *audioObjects* y *audioPackFormats* pueden estar integrados. Así, para cada `RenderingItem` se tienen en cuenta todos los *audioObjects* y *audioPackFormats* referentes implicados en la determinación del `RenderingItem`.

Se aplican las siguientes reglas:

– Si un *audioObject* tiene un valor *importance* inferior al umbral, también se descartarán todos los *audioObjects* referenciados. Para ello, se utilizará el valor *importance* más bajo de todos los *audioObjects* que llevan a un `RenderingItem` como *audioObject importance* para ese `RenderingItem`.

– Si un *audioPackFormat* tiene un valor *importance* inferior al umbral, también se descartarán todos los *audioPackFormats* referenciados. Para ello, se utilizará el valor *importance* más bajo de todos los *audioPackFormats* que llevan a un `RenderingItem` como *audioPackFormat importance* para ese `RenderingItem`.

– Al determinar la *importance* de un `RenderingItem` no se tendrán en cuenta los *audioObject* sin valor *importance*.

– Al determinar la *importance* de un `RenderingItem` no se tendrán en cuenta los *audioPackFormat* sin valor *importance*.

La implementación se efectúa en `fileio.utils.RenderingItemHandler`.

5.3.1.2 Tratamiento de la importancia estática

Dado un `RenderingItem` con `ImportanceData`, el elemento se suprimirá de la lista de elementos por reproducir si cualquiera de los valores de importancia estática (*audioObject*, *audioPackFormat*) es inferior al umbral correspondiente definido por el usuario:

$$\begin{aligned} & \texttt{importance.audio\_object} && < \texttt{audio\_object\_threshold} \\ \vee\ & \texttt{importance.audio\_pack\_format} && < \texttt{audio\_pack\_format\_threshold} \end{aligned}$$

La implementación se efectúa en `core.importance.filter_audioObject_by_importance` y `core.importance.filter_audioPackFormat_by_importance`.

5.3.1.3 Tratamiento de la importancia variable en el tiempo

No es posible tratar la importancia a nivel de *audioBlockFormat* (*typeDefinition==Object*) filtrando los `RenderingItems`, pues los elementos pueden encontrarse bajo el umbral sólo durante un tiempo. Para emular el descarte de elementos reproducidos en este caso concreto, el `RenderingItem` se silenciará efectivamente por la duración del *audioBlockFormat*. En este contexto «silenciar un *audioBlockFormat*» equivale a asumir un `bf.gain` igual a cero durante un *audioBlockFormat* `bf`.

La implementación se efectúa en `core.importance.MetadataSourceImportanceFilter`.

5.3.2 Emulación de conversión

Es posible aplicar la emulación de metadatos a los elementos reproducidos. Se puede desactivar la emulación de conversión, definir la conversión de metadatos al formato polar o definir la conversión de metadatos al formato cartesiano.

De activarse la emulación de conversión, se seleccionará la función adecuada del § 10 y se aplicará a todos los *audioBlockFormats* con *typeDefinition==Objects* en los elementos reproducidos seleccionados.

6 Componentes del reproductor compartidos

En esta sección se describen los componentes compartidos entre los subreproductores para las distintas typeDefinitions.

6.1 Panoramizador de fuente puntual polar

El componente panoramizador de fuente puntual es el núcleo del reproductor que, con la información sobre la disposición de los altavoces y una dirección 3D dadas, produce una ganancia por altavoz que, al aplicarse a una forma de onda mono/señal digital, tras reproducirse por los altavoces, debe causar al oyente la impresión de que el sonido emana de la dirección deseada.

El panoramizador de fuente puntual se utiliza en todo el reproductor, pues se utiliza para reproducir fuentes puntuales especificadas por los metadatos de objetos y como parte del sistema de reproducción extenso, como opción de reserva para el reproductor *DirectSpeakers* y como parte del proceso de diseño del descodificador HOA.

El panoramizador de fuente puntual de este reproductor se basa en la formulación VBAP [2] y contiene varias mejoras que lo hacen más adaptado para su utilización en entornos de radiodifusión:

– Además de las tripletas de altavoces, como en VBAP, el panoramizador de fuente puntual soporta cuadriláteros atómicos de altavoces, lo que resuelve los mismos problemas que la utilización de altavoces virtuales en otros sistemas, pero ofrece una función de panoramización general más paulatina.
– La triangulación de la disposición de los altavoces se realiza a partir de las posiciones nominales de los altavoces y se deforma para corresponder con las posiciones reales de los altavoces, lo que garantiza que la panoramización siempre es coherente aunque se adapte una disposición concreta.
– Se utilizan altavoces virtuales y el mezclado descendente para modificar la reproducción en ciertos casos a fin de corregir efectos perceptuales observados y producir el comportamiento deseado en disposiciones dispersas.
– Para evitar complicar el diseño cuando se utilizan disposiciones de altavoces extremadamente restringidas, el caso 0+2+0 se trata como un caso especial.

6.1.1 Arquitectura

El panoramizador de fuente puntual contiene una lista de objetos en la interfaz `RegionHandler`; cada objeto regional será responsable de producir la ganancia de los altavoces en un espacio concreto.

Para producir ganancias en una dirección dada, el panoramizador de fuente puntual cuestionará consecutivamente a cada región, que devolverán un vector de ganancia, si pueden tratar esa dirección, o un resultado nulo en caso contrario. Se utiliza el vector ganancia de la primera región que pueda tratar esa dirección.

Todo panoramizador de fuente puntual válido ha de cumplir dos condiciones:

– al menos una región puede tratar cualquier dirección dada;
– todas las regiones que pueden tratar una dirección dada generan ganancias similares (con cierto margen);
– dentro de cualquier región, la ganancia producida es lisa con respecto a la dirección deseada.

Todas estas propiedades garantizan que las ganancias producidas por un panoramizador de fuente puntual están bien definidas para todas las direcciones y siempre son, con cierto margen, lisas con respecto a la dirección.

Los tipos `RegionHandler` disponibles y el proceso de configuración utilizado para generar la lista de regiones para una disposición dada se describen en las siguientes cláusulas.

Este comportamiento se implementa en `core.point_source.PointSourcePanner`.

Además, se implementa una clase `PointSourcePannerDownmix` con la misma interfaz. Cuando se le cuestiona una posición, recurre a otro `PointSourcePanner` para obtener un vector ganancia, al que aplica una matriz de mezclado descendente y una normalización de potencia. Esto se utiliza en el § 6.1.3.1 para recartografiar los altavoces virtuales.

6.1.2 Tipos de región

La mayoría de regiones producen ganancias para un subconjunto de canales de salida. La correspondencia entre este subconjunto de canales y el vector completo de canales se implementa en `core.point_source.RegionHandler.handle_remap`.

6.1.2.1 Tripleta

Se trata de una región triangular esférica formada por tres altavoces en la implementación básica VBAP.

Esta región debe inicializarse con las posiciones 3D de los tres altavoces:

$$\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]^T$$

Las tres ganancias de salida, **g,** para una dirección D dada son tales que:

– $\mathbf{g} \cdot \mathbf{P} = s\mathbf{d}$ para algunos $s > 0$, con un pequeño margen.

– $g_i \geq 0 \; \forall \; i \in \{1,2,3\}$

– $\| \mathbf{g} \|_2 = 1$

Este tipo `RegionHandler` se implementa en `core.point_source.Triplet`.

6.1.2.2 VirtualNgon

Se trata de una región formada por n altavoces reales, que se divide en triángulos al añadir un único altavoz virtual. Cada triángulo está formado por dos altavoces reales adyacentes y el altavoz virtual, que se somete al mezclado descendente con los altavoces reales utilizando los coeficientes de mezclado descendente dados.

Por ejemplo, si se utilizan cuatro posiciones de altavoces reales $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4\}$ y una posición de altavoz virtual, $\mathbf{p}_v$, se crearán los siguientes triángulos:

– $\{\mathbf{p}_v, \mathbf{p}_1, \mathbf{p}_2\}$

– $\{\mathbf{p}_v, \mathbf{p}_2, \mathbf{p}_3\}$

– $\{\mathbf{p}_v, \mathbf{p}_3, \mathbf{p}_4\}$

– $\{\mathbf{p}_v, \mathbf{p}_4, \mathbf{p}_1\}$

Cuando se pide a este tipo `RegionHandler` una posición, se probará consecutivamente cada triángulo hasta que uno devuelva ganancias válidas, al igual que ocurre con el panoramizador de fuente puntual de nivel superior. Se crea así un vector de n ganancias para los altavoces reales, $\mathbf{g} = \{g_1, \ldots, g_n\}$, y la ganancia para el altavoz virtual, g_v, que se somete al mezclado descendente con los altavoces reales utilizando los coeficientes de mezclado descendente dados, $\mathbf{w}_{\mathrm{dmx}}$:

$$\mathbf{g}' = \mathbf{g} + \mathbf{W}_{\mathrm{dmx}} \quad g_v$$

Por último, se normaliza la potencia, lo que ofrece unas ganancias finales de:

$$\mathbf{g}'' = \frac{\mathbf{g}'}{\|\mathbf{g}'\|_2}$$

Este tipo `RegionHandler` se implementa en `core.point_source.VirtualNgon`.

6.1.2.3 QuadRegion

Se trata de una región cuadrilateral esférica formada por cuatro altavoces.

Las ganancias se calculan para cada altavoz dividiendo, en primer lugar, la posición en dos componentes, x e y. x puede considerarse la posición horizontal dentro del cuadrilátero, situándose 0 en el borde izquierdo y 1 en el borde derecho, e y puede considerarse la posición vertical, situándose 0 en el borde inferior y 1 en el borde superior.

Se establece la correspondencia entre los valores x e y y una ganancia de cada altavoz utilizando las ecuaciones (1) y (2). Los valores x e y (y, por tanto, las ganancias de los altavoces) que resultan en un vector velocidad dado pueden determinarse resolviendo las ecuaciones (1) a (3).

La solución a este problema es de complejidad similar a la de VBAP y ofrece la misma ganancia que VBAP en los bordes del cuadrilátero, lo que permite su utilización con otros tipos `RegionHandler` en un único panoramizador de fuente puntual conforme a las reglas del § 6.1.1.

Las ganancias resultantes son muy diferenciables en función de la posición dentro de la región, lo que produce resultados comparables a la panoramización por pares entre altavoces virtuales en situaciones comunes.

Este tipo `RegionHandler` se implementa en `core.point_source.QuadRegion`.

6.1.2.3.1 Formulación

Dada la posición cartesiana de cuatro altavoces, $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]$, en sentido levógiro desde el punto de vista del oyente, el vector ganancia $\mathbf{g}$ se calcula para una dirección fuente $\mathbf{d}$ de la siguiente manera:

$$\mathbf{g}' = [(1-x)(1-y), x(1-y), xy, (1-x)y] \tag{1}$$

$$\mathbf{g} = \frac{\mathbf{g}'}{\|\mathbf{g}'\|_2} \tag{2}$$

Escogiéndose los valores x e y de tal manera que el vector velocidad $\mathbf{g} \cdot \mathbf{P}$ tiene la dirección deseada $\mathbf{d}$. La magnitud del vector velocidad r es irrelevante, pues las ganancias se normalizan en potencia:

$$\mathbf{g} \cdot \mathbf{P} = r\mathbf{d} \tag{3}$$

Para algunos $r > 0$.

6.1.2.3.2 Solución

Dado un valor x, todos los vectores velocidad $\mathbf{d}$ con este valor x están un plan o formado por el origen del sistema de coordenadas y dos puntos a cierta distancia a lo largo de los bordes superior e inferior del cuadrilátero:

$$(1-x)\mathbf{p}_1 + x\mathbf{p}_2$$

$$(1-x)\mathbf{p}_4 + x\mathbf{p}_3$$

Por consiguiente:

$$(((1-x)\mathbf{p}_1 + x\mathbf{p}_2) \times ((1-x)\mathbf{p}_4 + x\mathbf{p}_3)) \cdot \mathbf{d} = 0 \tag{4}$$

Esta ecuación puede resolverse para encontrar x para una dirección fuente $\mathbf{d}$ dada.

Tómense los términos x:

$$[(\mathbf{p}_1 + x(\mathbf{p}_2 - \mathbf{p}_1)) \times (\mathbf{p}_4 + x(\mathbf{p}_3 - \mathbf{p}_4))] \cdot \mathbf{d} = 0$$

Amplíese el producto cruzado y tómense los términos:

$$\begin{aligned}&[(\mathbf{p}_1 \times \mathbf{p}_4)\\&+x\,((\mathbf{p}_1 \times (\mathbf{p}_3 - \mathbf{p}_4)) + ((\mathbf{p}_2 - \mathbf{p}_1) \times \mathbf{p}_4))\\&+x^2\,((\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_4))\\&] \cdot \mathbf{d} = 0\end{aligned}$$

Por último, multiplíquese por **D**:

$$\begin{aligned}&[(\mathbf{p}_1 \times \mathbf{p}_4) \cdot \mathbf{d}]\\+x\quad&[((\mathbf{p}_1 \times (\mathbf{p}_3 - \mathbf{p}_4)) + ((\mathbf{p}_2 - \mathbf{p}_1) \times \mathbf{p}_4)) \cdot \mathbf{d}]\\+x^2\quad&[((\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_4)) \cdot \mathbf{d}]\\&= 0\end{aligned}$$

Por consiguiente, la solución de x es la raíz de un polinomio, que puede resolverse con los métodos convencionales.

Sustituyendo **P** por $\mathbf{P}'$ en las ecuaciones anteriores, también puede determinarse y:

$$\mathbf{P}' = [\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_1]$$

Entonces pueden calcularse las ganancias **g** utilizando las ecuaciones 1 y 2. Dado que en la ecuación (4) se ignora la magnitud de **d**, pueden hallarse soluciones que produzcan un vector velocidad directamente opuesto al deseado, lo que puede verificarse si:

$$\mathbf{gP} \cdot \mathbf{d} > 0$$

6.1.2.4 StereoPanDownmix

Las señales de salida de una fuente puntual para estéreo (0+2+0) se obtienen utilizando un método basado en una mezcla descendente de 0+5+0 a 0+2+0. Este método se implementa de manera independiente.

El procedimiento es el siguiente:

– Se panoramiza la dirección de entrada utilizando un panoramizador de fuente puntual configurado para 0+5+0 a fin de producir un vector de cinco ganancias, $\mathbf{g}'$, en el orden M+030, M-030, M+000, M+110, M-110.

– Se aplica una matriz de conversión de formato de 0+5+0 a 0+2+0 para generar ganancias estéreo, $\mathbf{G}''$**,** en el orden M+030, M-030:

$$\mathbf{g}'' = \begin{bmatrix} 1 & 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{2}} & 0 \\ 0 & 1 & \sqrt{\frac{1}{3}} & 0 & \sqrt{\frac{1}{2}} \end{bmatrix} \cdot \mathbf{g}'$$

– Se normaliza en potencia $\mathbf{g}''$ a un valor determinado por la balanza entre los altavoces frontal y posterior en $\mathbf{g}'$, de manera que no se atenúan las fuentes entre M+030 y M-030, mientras que las fuentes entre M-110 y M+110 se atenúan en 3 dB.

$$\begin{aligned} a_{\text{front}} &= \max\{g'_1, g'_2, g'_3\} \\ a_{\text{rear}} &= \max\{g'_4, g'_5\} \\ r &= \frac{a_{\text{rear}}}{a_{\text{front}}+a_{\text{rear}}} \\ \mathbf{g} &= \mathbf{g}'' \frac{1\frac{r}{2}}{\|\mathbf{g}''\|_2} \end{aligned}$$

Este tipo `RegionHandler` se implementa en `core.point_source.StereoPanDownmix`.

NOTA – **g** de (0+5+0) a (0+2+0) corresponde totalmente a los coeficientes de mezcla descendente especificados en la Recomendación UIT-R BS.775 como se muestra a continuación:

$$\mathbf{g} = \begin{bmatrix} 1 & 0 & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 \\ 0 & 1 & \sqrt{\frac{1}{2}} & 0 & \sqrt{\frac{1}{2}} \end{bmatrix}$$

6.1.3 Procedimiento de configuración

El procedimiento de configuración crea un panoramizador de fuente puntual con los tipos `RegionHandler` indicados para una disposición dada. El procedimiento de configuración toma un objeto `Layout` (definido en el § 11.1.3) y produce un `PointSourcePanner`.

En un primer momento, el procedimiento de configuración selecciona el comportamiento en función del atributo `Layout::name`. Si el atributo `Layout::name` es `0+2+0`, la configuración se trata con la función de configuración especial para estéreo descrita en el § 6.1.3.2. Todos los demás casos se tratan con la función genérica expuesta en el § 6.1.3.1.

El procedimiento de configuración se implementa en `core.point_source.configure`.

6.1.3.1 Procedimiento para disposiciones genéricas

Para configurar un `PointSourcePanner` para disposiciones de altavoces genéricas se utiliza el siguiente procedimiento:

1) Se actualiza el acimut de las posiciones nominales de los altavoces con etiqueta `M+SC` o `M−SC` para garantizar la correcta triangulación con altavoces de pantalla muy espaciados. Si el acimut real (`polar_position.azimuth`) es φ, el acimut nominal, φ_n (`polar_nominal_position.azimuth`), es:

$$\varphi_n = \text{sgn}(\varphi) \times \begin{cases} 45 & |\varphi| > 30 \\ 15 & \text{en caso contrario} \end{cases}$$

2) Se determina el conjunto de altavoces virtuales recartografiados como se describe a continuación. Estos altavoces se añaden al conjunto de altavoces de la disposición y se tratan como los altavoces reales.

3) Se crean dos listas de posiciones de altavoces cartesianas normalizadas, que se utilizarán en las fases siguientes. La primera contiene las posiciones nominales de los altavoces (para triangular la disposición de los altavoces) y la segunda contiene las posiciones reales de los altavoces (para crear las regiones). Las posiciones nominales de los altavoces son las especificadas en la Recomendación UIT-R BS.2051-2, mientras que las posiciones reales de los altavoces son las posiciones efectivamente utilizadas por el sistema de reproducción en cuestión.

4) A cada lista de posiciones de altavoces se añaden uno o dos altavoces virtuales, que se convertirán en el altavoz virtual central de un `VirtualNgon`:

   - $0{,}0{,}-1$ (por debajo del oyente) siempre se añade, pues ninguna de las disposiciones de altavoces definidas en la Recomendación UIT-R BS.2051-2 cuenta con un altavoz en esta posición.
   - $0{,}0{,}1$ (por encima del oyente) se añade si en la disposición no hay altavoces con la etiqueta `T+000` o `UH+180`. Este altavoz no se utiliza cuando hay un `UH+180` porque, cuando se utiliza en la disposición `3+7+0` definida en la Recomendación UIT-R BS.2051-2, esa posición puede coincidir con la del altavoz virtual, lo que crea un cambio de etapa en la función de panoramización.

5) Se toma la envolvente convexa de las posiciones nominales de los altavoces. Si se implementa este algoritmo con aritmética de punto flotante, pueden darse errores que provoquen la división de algunas facetas de la envolvente convexa: las facetas se fusionan dentro de un margen de tolerancia de manera que el resultado es idéntico al que se obtendría si se implementase el algoritmo con aritmética exacta.

6) Se crea un `PointSourcePannerDownmix` con la siguientes regiones:

   - Para cada faceta de la envolvente convexa que no contiene uno de los altavoces virtuales añadidos en el paso 3:
     – Si la faceta tiene tres bordes, se crea un `Triplet` con las posiciones reales de los altavoces correspondientes a los vértices de la faceta.
     – Si la faceta tiene cuatro bordes, se crea un `QuadRegion` con las posiciones reales de los altavoces correspondientes a los vértices de la faceta.
   - Para cada altavoz virtual añadido en el paso 3, se crea un `VirtualNgon` con las posiciones reales de los altavoces adyacentes (todos los altavoces que comparten una faceta de la envolvente convexa con el altavoz virtual) en el borde, la posición del altavoz virtual en el centro y todos los coeficientes de mezcla descendente puestos a $\frac{1}{\sqrt{n}}$, siendo n el número de altavoces adyacentes.

     Téngase en cuenta que ninguna de las disposiciones definidas en la Recomendación UIT-R BS.2051-2 genera facetas con más de cuatro bordes.

     Los coeficientes de mezcla descendente establecen la correspondencia entre los altavoces virtuales y los altavoces físicos como se describe a continuación.

La implementación se efectúa en `core.point_source._configure_full`.

6.1.3.1.1 Determinación de altavoces virtuales con mezcla descendente directa

Por cada altavoz de capa media se añade un altavoz virtual en las capas inferior y superior con el mismo acimut que el altavoz real, si no hay altavoces en la capa superior o inferior en esa zona. Estos altavoces virtuales tendrán coeficientes de mezcla que hacen corresponder su salida directamente con la del altavoz de nivel medio correspondiente.

Como ocurre con los altavoces reales, los altavoces virtuales tienen posiciones reales y nominales; la posición real se deriva de las posiciones reales de los altavoces reales y la posición nominal se deriva de las posiciones nominales de los altavoces reales. La conveniencia de incluir o no un altavoz virtual depende de las posiciones nominales de los altavoces reales, de manera que para una determinada disposición siempre se utilizará el mismo conjunto de altavoces virtuales.

Para determinar el conjunto de altavoces virtuales para una disposición determinada se utiliza el siguiente procedimiento:

– Para cada $i \in [1, N]$, siendo $N = \texttt{len(layouts.channels)}$, el número de canales, se definen:

$$\begin{aligned}
\varphi_{i,r} &= \texttt{layouts.channels[i].polar_position.azimuth} \\
\varphi_{i,n} &= \texttt{layouts.channels[i].polar_nominal_position.azimuth} \\
\theta_{i,r} &= \texttt{layouts.channels[i].polar_position.elevation} \\
\theta_{i,n} &= \texttt{layouts.channels[i].polar_nominal_position.elevation}
\end{aligned}$$

– Se definen tres conjuntos de índices de canal, que identifican los canales en las capas superior, media e inferior de la disposición:

$$\begin{aligned}
S_u &= \{i \mid 30° \le \theta_{i,n} \le 70°\} \\
S_m &= \{i \mid -10° \le \theta_{i,n} \le 10°\} \\
S_l &= \{i \mid -70° \le \theta_{i,n} \le -30°\}
\end{aligned}$$

– Los altavoces virtuales tienen los mismos acimutes reales y nominales que los correspondientes altavoces reales. La elevación real es la elevación media de los altavoces reales en la capa, de haberla, o $-30°$ o $30°$ para las capas inferior y superior, de no haberla. La elevación nominal es siempre $-30°$ o $30°$ para las capas inferior y superior.

Se definen dos elevaciones nominales:

$$\theta'_{u,n} = 30°$$
$$\theta'_{l,n} = -30°$$

Se definen dos elevaciones reales:

$$\theta'_{u,r} = \begin{cases} 30° & |S_u| = 0 \\ \frac{\sum_{j \in S_u} \varphi_{j,r}}{|S_u|} & \text{en caso contrario} \end{cases}$$

$$\theta'_{l,r} = \begin{cases} 30° & |S_u| = 0 \\ \frac{\sum_{j \in S_l} \varphi_{j,r}}{|S_l|} & \text{en caso contrario} \end{cases}$$

– Sólo se crean altavoces en una capa si el acimut nominal absoluto del altavoz de capa media correspondiente es igual o superior al acimut nominal absoluto máximo de los altavoces reales de la capa más 40°. Estos límites acimutales se definen de la siguiente manera:

$$L_u = \begin{cases} 0 & |S_u| = 0 \\ \max\limits_{j \in S_u} |\varphi_{j,n}| + 40° & \text{en caso contrario} \end{cases}$$

$$L_l = \begin{cases} 0 & |S_l| = 0 \\ \max\limits_{j \in S_l} |\varphi_{j,n}| + 40° & \text{en caso contrario} \end{cases}$$

– Por cada j en S_m:

  • Se crea un altavoz superior virtual si $\varphi_{j,n} \geq L_u$, identificado por una estructura `channel Channel`, con:

$$\begin{aligned} \texttt{channel.polar\_position.azimuth} &= \varphi_{j,r} \\ \texttt{channel.polar\_position.elevation} &= \theta'_{u,r} \\ \texttt{channel.polar\_nominal\_position.azimuth} &= \varphi_{j,n} \\ \texttt{channel.polar\_nominal\_position.elevation} &= \theta'_{u,n} \end{aligned}$$

  • Se crea un altavoz inferior virtual si $\varphi_{j,n} \geq L_l$, identificado por una estructura `channel Channel`, con:

$$\begin{aligned} \texttt{channel.polar\_position.azimuth} &= \varphi_{j,r} \\ \texttt{channel.polar\_position.elevation} &= \theta'_{l,r} \\ \texttt{channel.polar\_nominal\_position.azimuth} &= \varphi_{j,n} \\ \texttt{channel.polar\_nominal\_position.elevation} &= \theta'_{l,n} \end{aligned}$$

  Ambos tienen coeficientes de mezcla que encaminan las ganancias de este altavoz a las del correspondiente altavoz de capa media, j.

La implementación se efectúa en `core.point_source.extra_pos_vertical_nominal`.

6.1.3.2 Procedimiento para `0+2+0`

Para 0+2+0, se devuelve un `PointSourcePanner` con una única región `StereoPanDownmix`.

La implementación se efectúa en `core.point_source._configure_stereo`.

6.2 Determinar si el ángulo está dentro del margen de tolerancia

Se utiliza una función `inside_angle_range` para comparar ángulos con las gamas angulares dadas, permitiendo la especificación de gamas con la parte posterior del sistema de coordenadas. Esto se utiliza en la zona de exclusión y los componentes *DirectSpeakers* de los § 7.3.12.1 y 8.4.

La firma es:

```
bool inside_angle_range(float x, float start, float end, float tol=0.0);
```

Esto devuelve verdadero si el ángulo `x` está dentro del arco circular que empieza en `start` y va en sentido levógiro hasta `end`, ampliado por `tol`. Todos los ángulos se dan en grados.

En el caso común en que:

$$-180 \leq \texttt{start} \leq \texttt{end} \leq 180$$

Esta función es equivalente a:

$$\texttt{start} - \texttt{tol} \leq \texttt{x}' \leq \texttt{end} + \texttt{tol}$$

siendo $\texttt{x}' = \texttt{x} + 360 \times i$ para algunos i de manera que $-180 < \texttt{x}' \leq 180$.

En otros casos, el comportamiento es más sutil. Por ejemplo, si $\texttt{start} = 90$ y $\texttt{end} = -90$, se especifica la mitad posterior del sistema de coordenadas:

$$x' \leq -90 \vee x' \geq 90$$

En el Cuadro 2 se dan algunos ejemplos de gamas y su expresión equivalente.

CUADRO 2

Expresiones equivalentes a `inside_angle_range(x, start, end, tol)`

`start`	`end`	`tol`	**Expresión equivalente**
-90	90	0	$-90 \leq x' \leq 90$
-90	90	5	$-95 \leq x' \leq 95$
90	-90	0	$x' \leq -90 \vee x' \geq 90$
90	-90	5	$x' \leq -85 \vee x' \geq 85$
0	0	0	$x' = 0$
180	180	0	$x' = 180$
-180	-180	0	$x' = 180$
180	180	5	$x' \leq -175 \vee x' \geq 175$
-180	180	0	Verdadero

Esta función se implementa en `core.geom.inside_angle_range`.

6.3 Determinar si un canal es un canal LFE a partir de sus metadatos de frecuencia

Los metadatos de frecuencia, que pueden estar presentes como subelementos *frequency* de *audioChannelFormats*, pueden utilizarse para determinar si un canal es efectivamente un canal LFE.

Para representar los metadatos de frecuencia se utiliza la siguiente estructura de datos:

```
struct Frequency {
  optional<float> lowPass;
  optional<float> highPass;
};
```

La función con la firma

```
bool is_lfe(Frequency frequency)
```

evalúa

$$\texttt{frequency.lowPass} \wedge \neg\texttt{frequency.highPass} \wedge (\texttt{frequency.lowPass} \leq 120\,\text{Hz})$$

y devuelve `True` si se supone que el canal es un canal LFE y `False` en caso contrario.

La implementación se efectúa en `core.renderer_common.is_lfe`.

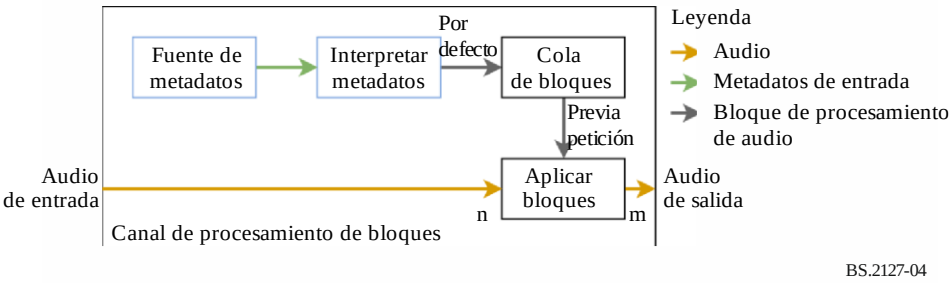
NOTA – En la Recomendación UIT-R BS.2127-0, la frecuencia para seleccionar los canales LFE se establecía en 200 Hz o menos.

6.4 Canal de procesamiento de bloques

Al reproducir metadatos ADM temporizados se requiere algún tipo de funcionalidad idéntica para todos los valores *typeDefinition;* para un determinado subconjunto de canales de entrada se aplica un procesamiento entre límites temporales, generando canales de altavoz en la salida.

FIGURA 4

Estructura utilizada para procesar los canales conexos. Los componentes en azul proceden del exterior



En la Fig. 4 se muestra la estructura de este procedimiento. La interfaz con este componente es la siguiente:

```
class BlockProcessingChannel {
    BlockProcessingChannel(MetadataSource metadata_source, Callable
interpret_metadata);
    void process(int sample_rate, int start_sample,
        ndarray<float> input_samples, ndarray<float> &output_samples);
};
```

El sistema ofrece `MetadataSource` como mecanismo para insertar los metadatos en el reproductor y su interfaz es la siguiente:

```
class MetadataSource {
    optional<TypeMetadata> get_next_block();
};
```

Mediante la llamada repetitiva a `get_next_block`, el canal de procesamiento de bloques recibe una secuencia de bloques `TypeMetadata`, como se describe en el § 5, correspondiente a bloques temporizados de los metadatos necesarios durante la reproducción.

Estos bloques de metadatos se interpretan con la función `interpret_metadata`, que facilita el reproductor para cada *typeDefinition*. Estas funciones aceptan un `TypeMetadata` y devuelven una lista de objetos `ProcessingBlock`, que encapsula el procesamiento de audio temporizado necesario para implementar el `TypeMetadata` dado. La interpretación de *typeDefinition==Objects* se describe detalladamente en el § 7.2. Para *typeDefinition==HOA* y *typeDefinition==DirectSpeakers* se devuelve un único `ProcessingBlock`.

Los objetos `ProcessingBlock` tienen la siguiente interfaz externa:

```
class ProcessingBlock {
    Fraction start_sample, end_sample;
    int first_sample, last_sample;

    void process(int in_out_samples_start,
        ndarray<float> input_samples, ndarray<float> &output_samples);
}
```

Se supone que las muestras que pasan a `process` son un subconjunto de muestras del fichero de entrada/salida, de manera que `input_samples`[i] y `output_samples`[i] representan todas las

muestras de entrada y salida `in_out_samples_start` + i. Los atributos `first_sample` y `last_sample` definen la gama de números de muestras totales, s, que pueden verse afectadas por `process`:

$$\texttt{first\_sample} \leq s \leq \texttt{last\_sample}$$

`start_sample` y `end_sample` son las fracciones de las muestras de inicio y fin, que se utilizan para determinar los atributos `first_sample` y `last_sample`, y pueden utilizar las implementaciones de subclase `ProcessingBlock`.

Los objetos `BlockProcessingChannel` almacenan una cola de `ProcessingBlock`, que se rellena con los bloques solicitantes de `metadata_source` y los pasan por `interpret_metadata`. `BlockProcessingChannel.process` aplica los bloques de procesamiento de la cola a las muestras que se le pasan, utilizando `first_sample` y `last_sample` para determinar cuándo pasar al siguiente bloque.

Esta estructura permite el desdoblamiento de los componentes del reproductor. Las muestras de audio pueden procesarse en fragmentos, independientemente del tamaño del bloque de metadatos, permitiendo al mismo tiempo el procesamiento de metadatos con precisión de muestra y sin complicar los reproductores con problemas propios de la temporización.

La decisión de permitir que el reproductor solicite los bloques de metadatos deja la interpretación de los metadatos de temporización dentro del reproductor. Si los metadatos se insertasen en el reproductor, el componente de inserción debería saber cuándo se necesita el bloque siguiente, lo que depende de la información de temporización que contiene.

Esta funcionalidad de implementa en `core.renderer_common`.

6.4.1 Tipos `ProcessingBlock` implementados

Los tres tipos de bloques de procesamiento comunes son:

`FixedGains` toma un único canal de entrada y aplica n ganancias, sumando la salida en n canales de salida.

`FixedMatrix` toma N canales de entrada y aplica una matriz de ganancia NxM para formar M canales de salida.

`InterpGains` toma un único canal de entrada y aplica n ganancias interpoladas linealmente, sumando la salida en n canales de salida. Se facilitan dos vectores ganancia `gains_start y gains_end`, que son las ganancias que se han de aplicar en `start_sample y end_sample`. La ganancia $g(i, s)$ que se aplica al canal i en la muestra s se obtiene con la siguiente fórmula:

$$p(s) = \frac{s - \texttt{start\_sample}}{\texttt{end\_sample} - \texttt{start\_sample}}$$

$$g(i, s) = (1 - p(s)) \times \texttt{gains\_start}[i] + p(s) \times \texttt{gains\_end}[i]$$

6.5 Interpretación genérica de los metadatos de temporización

La determinación de los tiempos de inicio y fin de los bloques se comparte entre reproductores para distintos *typeDefinitions*. Para un objeto `block TypeMetadata`, se utiliza el procedimientos siguiente:

– El tiempo de inicio y fin del objeto que contiene el bloque se determina a partir de `block.extra_data.object_start` y `block.extra_data.object_duration`. Si `object_start` es Ninguno, se

supone que el objeto empieza en el tiempo 0. Si `object_duration` es Ninguna, se supone que se extiende hasta el infinito.

– Los tiempos de inicio y fin del bloque se determinan a partir de los atributos `rtime` y `duration`:

  - Si `rtime` y `duration` no son `None`, se supone que el tiempo de inicio del bloque es el tiempo de inicio del objeto más `rtime` y que el tiempo de fin del bloque es el tiempo de inicio del bloque más `duration`.
  - Si `rtime` y `duration` son `None`, se supone que el bloque se extiende del tiempo de inicio del objeto al tiempo de fin del objeto.
  - Se considera que otras constelaciones de `rtime` y `duration` son un **error**. Para múltiples objetos *audioBlockFormat* dentro de un *audioChannelFormat* se han de facilitar tanto *rtime* como *duration*, mientras que para un único bloque que abarque todo el *audioObject*, no es necesario facilitar *rtime* o *duration*. En caso contrario, el comportamiento es indefinido.

Se ha de verificar la coherencia de los tiempos. No deben permitirse bloques que terminen después del tiempo de fin del objeto ni el solapamiento de bloques en una secuencia, pues se considera un error. Un error implica que los implementadores deben considerar que hay algo mal en los datos de entrada. El procedimiento correcto consiste en reparar el sistema que los ha producido. En la implementación de referencia los errores se tratan deteniendo el proceso de reproducción y comunicando el error al usuario. Otras implementaciones pueden tratar los errores de distinta manera, en función de su entorno de aplicación objetivo.

La implementación se efectúa en `core.renderer_common.InterpretTimingMetadata`.

6.6 Interpretación de `TrackSpecs`

El audio entra en el reproductor a través de un bus multicanal leído directamente del fichero de entrada. Los metadatos de entrada en forma de `RenderingItems` incluyen objetos `TrackSpec`, que son instrucciones para la extracción de canales de este bus, incluida la aplicación de un preprocesamiento Matrix que mezcla múltiples canales.

El procesamiento de cada tipo `TrackSpec` se implementa en `core.track_processor`.

Dado un `TrackSpec`, puede crearse un objeto `TrackProcessor`, que tiene un único método `process(sample_rate, input_samples)`, que aplica el procesamiento especificado a `input_samples` y devuelve el resultado de canal único (a la velocidad de muestras dada).

6.6.1 SilentTrackSpec

Para n muestras de entrada, `process` para un `SilentTrackSpec` devuelve n muestras de valor cero.

6.6.2 DirectTrackSpec

`process` para un `DirectTrackSpec track_spec` devuelve las muestras de entrada en la pista especificada en `track_spec.track_index` (utilizando una indización con base cero).

6.6.3 MixTrackSpec

`process` para un `MixTrackSpec track_spec` devuelve la suma de los resultados de llamar a `process` en un `TrackProcessor` para cada subpista de `track_spec.input_tracks`.

6.6.4 MatrixCoefficientTrackSpec

`process` para un `MatrixCoefficientTrackSpec track_spec` aplica el procesamiento matriz especificado en `track_spec.coefficient` (que representa los parámetros de un único elemento *coefficient* de matriz) a un único canal especificado por `track_spec.input_track`.

Si `track_spec.coefficient.gain` no es `None`, las muestras se multiplican por `gain`.

Si `track_spec.coefficient.delay` no es `None`, las muestras se retrasan n muestras, `delay` mseg, redondeado a la muestra más cercana (con relación discontinua a 0):

$$n = \left\lceil \frac{\texttt{sample\_rate} \times \texttt{delay}}{1000} - \frac{1}{2} \right\rceil$$

Algunos parámetros no se soportan. Si `gainVar`, `delayVar`, `phaseVar` o `phase` no son `None`, o `delay` es negativo, se genera un error.

6.7 Ángulo relativo

`relative_angle`(x, y) se utiliza para encontrar un ángulo equivalente a y que sea mayor o igual a x. Esto se realiza para evitar casos límite al trabajar con arcos circulares.

`relative_angle`(x, y) devuelve $y' = y + 360n$, siendo n el entero más pequeño para que $y' \geq x$

6.8 Transformación de coordenadas

La función `cart` se define para traducir las posiciones polares a posiciones cartesianas, de acuerdo con el § 2.2:

$$cart(\varphi, \theta, d) = \{x, y, z\}$$

siendo:

$$\begin{aligned} x &= \text{sen}\left(-\frac{\pi}{180}\varphi\right)\cos\left(\frac{\pi}{180}\theta\right)d \\ y &= \cos\left(-\frac{\pi}{180}\varphi\right)\cos\left(\frac{\pi}{180}\theta\right)d \\ z &= \text{sen}\left(\frac{\pi}{180}\theta\right)d \end{aligned}$$

También se define la transformación inversa para obtener el acimut y la elevación a partir de una posición cartesiana:

$$\begin{aligned} \texttt{azimuth}(\{x, y, z\}) &= -\frac{180}{\pi}\text{atan2}(x, y) \\ \texttt{elevation}(\{x, y, z\}) &= \frac{180}{\pi}\text{atan2}\left(z, \sqrt{x^2 + y^2}\right) \end{aligned}$$

La función `local_coordinate_system` produce una matriz de rotación que establece la correspondencia entre $\{0,1,0\}$ y un acimut y una elevación determinados:

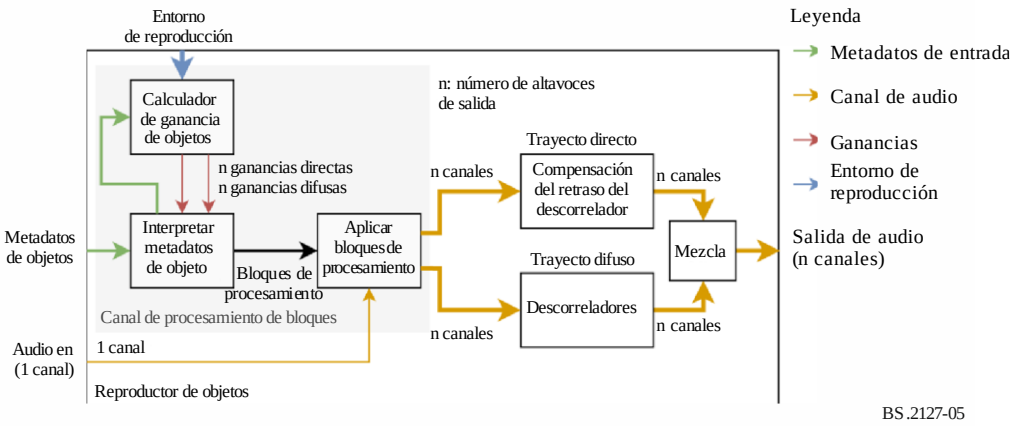
$$\texttt{local\_coordinate\_system}(\varphi, \theta) = \begin{bmatrix} \texttt{cart}(\varphi - 90,0,1) \\ \texttt{cart}(\varphi, \theta, 1) \\ \texttt{cart}(\varphi, \theta + 90,1) \end{bmatrix}$$

7 Reproducción de elementos con typeDefinition==Objects

7.1 Estructura

FIGURA 5

Estructura del reproductor de objetos



En la Fig. 5 se muestra la estructura del reproductor para *typeDefinition==Objects*. En esta Figura se muestra el procesamiento aplicado a un único elemento reproducido. En la reproducción de múltiples elementos la estructura se duplica para cada elemento y se mezclan las salidas.

Los metadatos entran en el reproductor en forma de objeto `ObjectRenderingItem`, que contiene un índice de pistas, y una fuente de objetos `ObjectTypeMetadata` que representan los parámetros de reproducción temporizados de la pista identificada.

El método descrito en el § 7.2 se aplica a cada objeto `ObjectTypeMetadata` para interpretar los metadatos de temporización, y se calculan los vectores de ganancia con el calculador de ganancia descrito en el § 7.3. Esto genera objetos `ProcessingBlock`, que aplican un procesamiento de señal temporizada al audio de entrada para producir un bus directo y un bus difuso, cada uno de ellos con un canal por altavoz. Este método, y la clase `BlockProcessingChannel` que lo encapsula, se describen en el § 6.4.

El bus difuso pasa por un banco de filtro de descorrelación por canal y el bus directo se retrasa el tiempo correspondiente antes de mezclarlos para formar la salida. Los filtros de descorrelación y los retrasos se describen en el § 7.4.

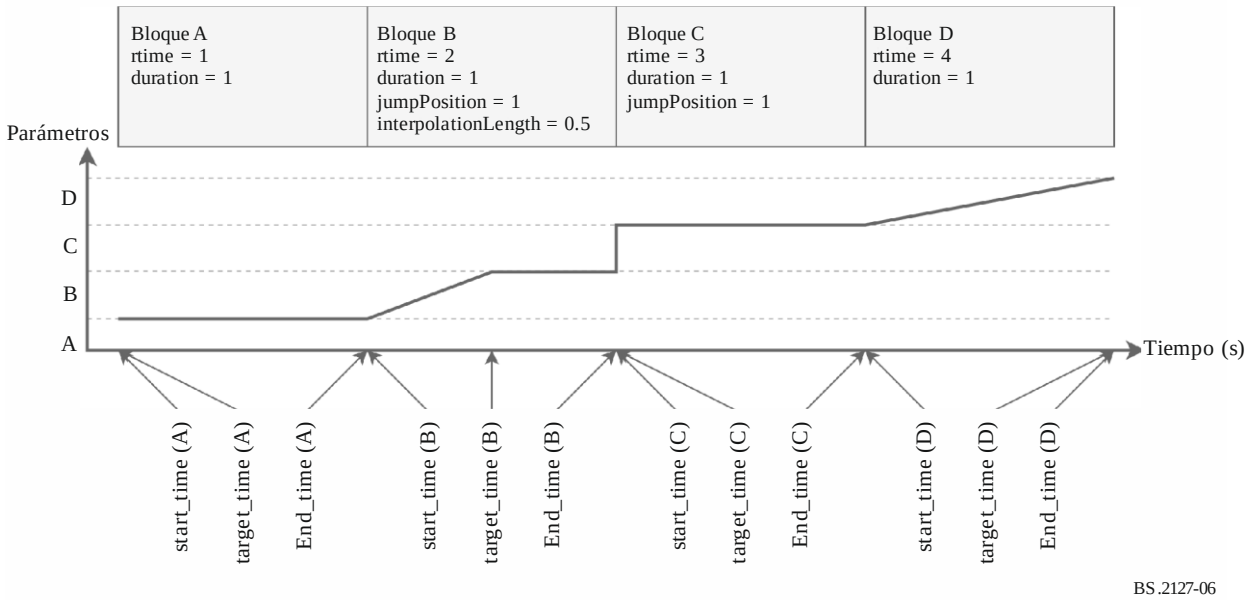
Esta estructura se implementa en `core.objectbased.renderer.ObjectRenderer`.

7.2 InterpretObjectMetadata

Los metadatos de temporización de objeto se interpretan en la clase `InterpretObjectMetadata`, que encaja en la estructura de canal de procesamiento de bloques.

FIGURA 6

Ejemplos de audioBlockFormats y curvas de interpolación interpretada



Para cada `ObjectTypeMetadata` de entrada el procedimiento aplicado es el siguiente:

– Los tiempos de inicio y fin, `start_time` y `end_time`, del bloque se determinan como se indica en el § 6.5.

– El tiempo en que termina la interpolación de este bloque, `target_time`, se determina en función de los siguientes casos, que se ilustran en los bloques correspondientes de la Fig. 6:

A

Si es el primer bloque, o si el `end_time` del bloque anterior es inferior al `start_time` del bloque presente:

$$\texttt{target\_time} = \texttt{start\_time}$$

B

Si `bf.jumpPosition.flag` está configurado y `bf.jumpPosition.interpolationLength` no es `None`:

$$\texttt{target\_time} = \texttt{start\_time} + \texttt{bf.jumpPosition.interpolationLength}$$

C

Si `bf.jumpPosition.flag` está configurado y `bf.jumpPosition.interpolationLength` es `None`:

$$\texttt{target\_time} = \texttt{start\_time}$$

D

Si `bf.jumpPosition.flag` no está configurado, la interpolación se realiza en todo el bloque:

$$\texttt{target\_time} = \texttt{end\_time}$$

– El vector ganancia `interp_to` se calcula utilizando una instancia `GainCalculator` para el bloque presente. `interp_from` es el vector ganancia calculado para el bloque anterior.

– Si `start_time` < `target_time`, se crea un `InterpGains ProcessingBlock`, que interpola de `interp_from a interp_to` entre `start_time` y `target_time`.

– Si `target_time` < `end_time`, se crea un `FixedGainsProcessingBlock`, que aplica interp_to entre start_time y target_time.

La implementación se efectúa en `core.objectbased.renderer.InterpretObjectMetadata`.

7.3 Calculador de ganancia

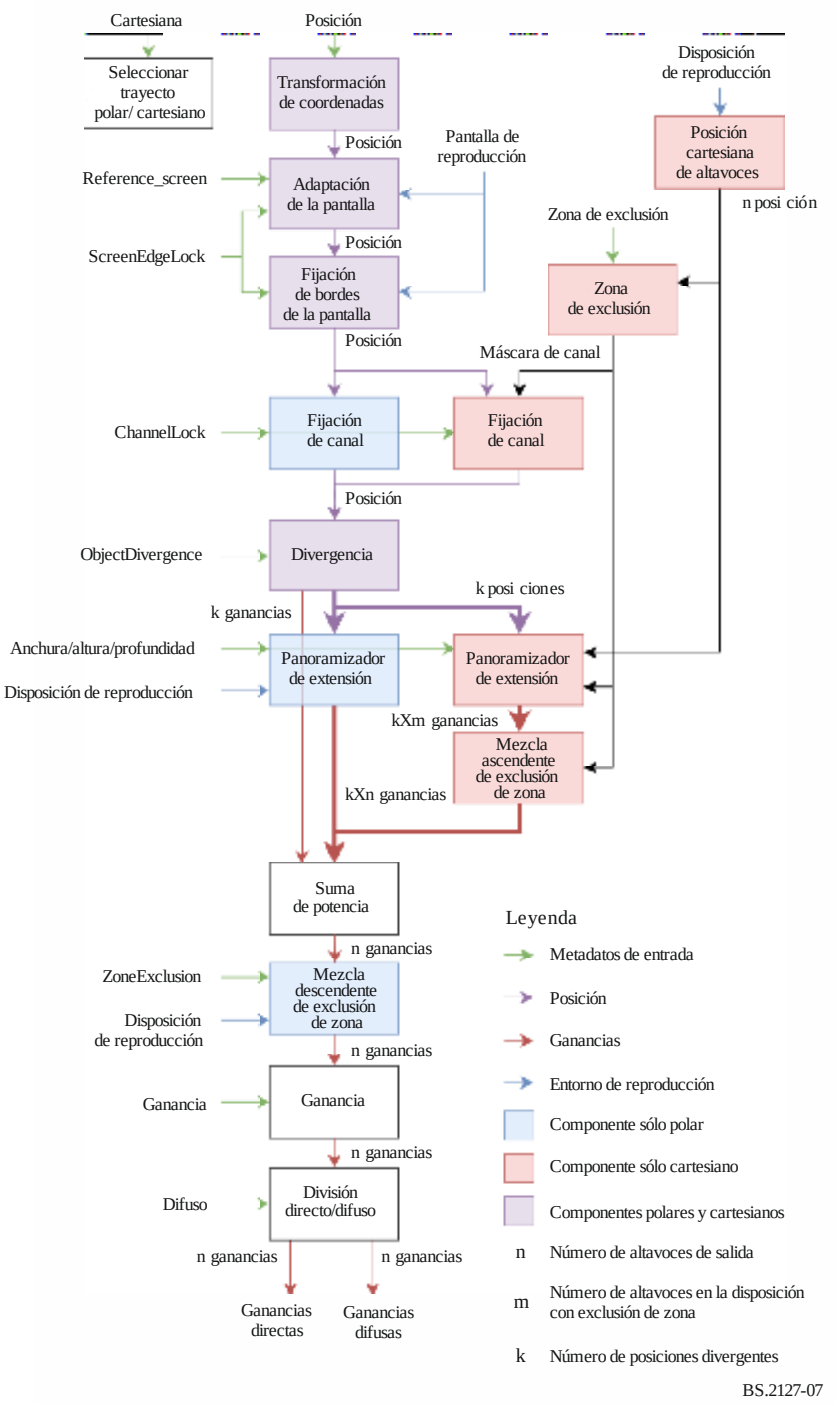
Si tomamos un objeto `ObjectTypeMetadata`, este objeto calcula una ganancia para cada altavoz en los trayectos directo y difuso. La interfaz con este componente es:

```
struct      DirectDiffuseGains {
    vector<float> direct;
    vector<float> diffuse;
};

class GainCalc {
    GainCalc(Layout layout);

    DirectDiffuseGains render(ObjectTypeMetadata otm);
};
```

7.3.1 Estructura

FIGURA 7

Estructura del calculador de ganancia para *typeDefinition==Objects*



Este componente es básicamente una mezcla de los subcomponentes enumerados en esta cláusula. En la Fig. 7 se muestra un diagrama del flujo que sigue la señal entre estos componentes. El comportamiento de un `ObjectTypeMetadata otm` con un atributo `bf block_format` es el siguiente:

– Se aplica la transformación de coordenadas descrita en el § 7.3.2 a `bf.position` para obtener un objeto `position CartesianPosition`.

– La adaptación de pantalla se efectúa con el método descrito en el § 7.3.3 y los parámetros `position`, `bf.screenRef`, `otm.extra_data.reference_screen` y `bf.cartesian`, lo que actualiza `position`. Este componente se inicializa con la pantalla de reproducción (`layout.screen`) y la disposición de reproducción (`layout`).

– La fijación de los bordes de la pantalla se efectúa con el método descrito en el § 7.3.4 y los parámetros `position`, `bf.position.screenEdgeLock` y `bf.cartesian`, y se actualiza `position` con el resultado. Este componente se inicializa con la pantalla de reproducción (`layout.screen`) y la disposición de reproducción (`layout`).

– Si `bf.cartesian`:

  • Se determina la posición alocéntrica de cada altavoz en `layout.without_lfe` de acuerdo con el § 7.3.9, obteniendo una matriz de `allo_channel_positions`.

  • Se aplica el algoritmo de exclusión de zona del § 7.3.5 a `allo_channel_positions` y `bf.zone_exclusion`, y se obtiene una máscara booleana de altavoces que excluir, `excluded`.

  • Se aplica la fijación de canal a la configuración alocéntrica descrita en el § 7.3.6 con los parámetros `position`, `bf.channelLock` y `excluded`, lo que actualiza `position`.

  Alternativamente:

  • Se aplica la fijación de canal a la configuración egocéntrica descrita en el § 7.3.6 con los parámetros `position` y `bf.channelLock`, lo que actualiza `position`.

– Se aplica la divergencia con el método descrito en el § 7.3.7 y los parámetros `position`, `bf.objectDivergence` y `bf.cartesian`. El resultado es hasta tres fuentes ampliadas cuyas ganancias y posiciones se almacenan en `diverged_gains` y `diverged_positions`.

– Si `bf.cartesian`:

  • Se aplica el panoramizador de extensión del § 7.3.11 a cada `p` de `diverged_positions`, con los parámetros `channel_positions`, `p`, `bf.width`, `bf.height`, `bf.depth`, lo que resulta en vectores ganancia para la matriz de altavoces no excluidos. `channel_positions` es una lista de posiciones de canal no excluidas seleccionadas a partir de `allo_channel_positions[i]` siendo `excluded[i] False`.

  • Estos vectores ganancia se mezclan en ascendente de acuerdo con `excluded`, y se obtiene una ganancia para cada altavoz `i` cuando `excluded[i]` es `False`, y cero cuando `excluded[i]` es `True`, que se almacena en `gains_for_each_pos`.

  Alternativamente:

  • Se aplica el panoramizador de extensión del § 7.3.8 a cada `p` de `diverged_positions`, con los parámetros `p`, `bf.width`, `bf.height`, `bf.depth`, lo que resulta en un vector ganancia por altavoz, que se almacena en `gains_for_each_pos`.

– Las ganancias de `gains_for_each_pos` se mezclan con una potencia determinada por `diverged_gains`:

$$\texttt{gains}[i] = \sqrt{\sum_j \texttt{diverged\_gains}\,[j] \times \texttt{gains\_for\_each\_pos}[j,i]^2}$$

– Si `bf.cartesian` no está configurado, se aplica la zona de exclusión descrita en el § 7.3.12 a `gains` y `bf.zoneExclusion`, con lo que se obtiene un nuevo vector `gains`. Este componente se inicializa con `layout.without_lfe`.

– `gains` se amplía añadiendo ganancias de canal LFE de valor 0 para producir `gains_full`, con un valor por cada altavoz de `layout`.

– `gains_full` se divide en un vector directo y un vector difuso para controlar los trayectos directo y difuso, en función del parámetro `bf.diffuse`. Estos se devuelven como `DirectDiffuseGains` con atributos:

$$\begin{aligned} \texttt{direct} &= \texttt{gains\_full} \times \sqrt{1 - \texttt{bf.diffuse}} \\ \texttt{diffuse} &= \texttt{gains\_full} \times \sqrt{\texttt{bf.diffuse}} \end{aligned}$$

7.3.1.1 Discusión (Informativo)

La estructura del calculador de ganancia está influida por los dos siguientes principios:

– Si los parámetros son escasos (es decir, sólo se utiliza un pequeño número de los campos de metadatos posibles), es preferible conservar la interpretación obvia de esos parámetros.

– Cuando se utilizan combinaciones de parámetros, se escoge la opción que da al usuario la mayor cantidad de posibilidades para los distintos comportamientos.

Por ejemplo:

– Fijación de canal se implementa como una modificación de posición – si se utiliza fijación de canal por sí misma (con la adecuada *maxDistance*), la fuente se fijará a un canal a causa del comportamiento del panoramizador de fuente puntual. Sin embargo, la fijación de canal también puede utilizarse con parámetros de extensión para producir una fuente ampliada centrada en torno a un altavoz concreto, por ejemplo.

– La difusividad no está vinculada a la extensión – puede obtenerse una fuente difusa plenamente extendida configurando adecuadamente los parámetros de extensión, pero esto también permite utilizar el filtrado de descorrelación con extensiones incompletas.

7.3.2 Transformación de coordenadas

La transformación de coordenadas simple se implementa en `core.objectbased.gain_calc.coord_trans`, que se utiliza para convertir posiciones entrantes en coordenadas cartesianas uniformes. La firma es la siguiente:

```
CartesianPosition coord_trans(ObjectPosition position);
```

`position` se convierte en primer lugar a un vector cartesiano, **p**.

Si `position` es un `ObjectCartesianPosition`, los elementos de **p** se recortan para que coincidan con la gama $[-1,1]$ antes de devolverlos:

$$\texttt{clip}(\mathbf{p}, -1, 1)$$

En caso contrario, se devuelve **p** sin modificar.

`clip` se define para números reales como:

$$\texttt{clip}(x, a, b) = \begin{cases} a & x \le a \\ x & a \le x \le b \\ b & b \le x \end{cases}$$

y se aplica comúnmente a cada elemento de un vector:

$$\texttt{clip}(\{x, y, z\}, a, b) = \{\texttt{clip}(x, a, b), \texttt{clip}(y, a, b), \texttt{clip}(z, a, b)\}$$

7.3.3 Adaptación de pantalla

El componente adaptación de pantalla deforma las posiciones fuente para compensar las diferencias de geometría de la pantalla entre los entornos de producción y de reproducción. La interfaz a este componente es:

```
class ScreenScaleHandler {
  ScreenScaleHandler(Screen reproduction_screen);
  CartesianPosition handle(
    CartesianPosition position,
    bool screenRef,
    Screen reference_screen,
    bool cartesian
  );
};
```

Se utilizan dos definiciones de pantalla:

Pantalla de referencia

El `audioProgrammeReferenceScreen` enumerado en el elemento `audioProgramme` o el tamaño de pantalla polar por defecto, si no se facilita, se trata de la geometría de pantalla utilizada durante la producción de los metadatos.

Pantalla de reproducción

Geometría de pantalla en el entorno de reproducción en que se escuchará la salida del reproductor.

Se deforman las posiciones de la pantalla de referencia para que aparezcan en las posiciones correspondientes de la pantalla de reproducción.

7.3.3.1 Representación de pantalla interna

La información sobre ambas pantallas puede facilitarse en coordenadas polares o cartesianas (objetos `PolarScreen` o `CartesianScreen`). A diferencia de lo que ocurre con las posiciones de fuente de objetos, no hay una equivalencia obvia entre ambas, pero para simplificar la implementación se necesita una representación de pantalla única que pueda representar a ambos tipos de pantalla. Tal es el objetivo de la estructura `PolarEdges`, que almacena los acimutes de los bordes izquierdo y derecho de la pantalla y las elevaciones de los bordes superior e inferior de la pantalla:

```
struct PolarEdges {
    float left_azimuth;
    float right_azimuth;
    float bottom_elevation;
    float top_elevation;
};
```

Se crea un objeto `PolarEdges` a partir de un objeto `PolarScreen` o `CartesianScreen` dado, transformando, en primer lugar, la pantalla en una posición central cartesiana y dos vectores (para las direcciones x y z) que definen la superficie de la pantalla y, a continuación, se hallan el acimut y la elevación de cada uno de los bordes.

Para un `PolarScreen screen` donde:

$$\begin{aligned} \varphi &= \texttt{screen.centrePosition.azimuth} \\ \theta &= \texttt{screen.centrePosition.elevation} \\ d &= \texttt{screen.centrePosition.distance} \\ w &= \texttt{screen.widthAzimuth} \\ a &= \texttt{screen.aspectRatio} \end{aligned}$$

Se utiliza el siguiente procedimiento:

– la posición central es una conversión cartesiana simple de la posición central:

$$centre = cart(\varphi, \theta, d)$$

– se calculan la anchura y la altura cartesianas:

$$\begin{aligned} \texttt{width} &= d \cdot \tan\left(\frac{\pi}{180}\frac{w}{2}\right) \\ \texttt{height} &= \frac{\texttt{width}}{a} \end{aligned}$$

– se utiliza `local_coordinate_system` para hallar los vectores x y z de la pantalla:

$$\begin{aligned} \begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix} &= \texttt{local\_coordinate\_system}(\varphi, \theta) \\ v_x &= \texttt{width} \times l_x \\ v_z &= \texttt{height} \times l_z \end{aligned}$$

Para un `CartesianScreen screen` donde:

$$\begin{aligned} w &= \texttt{screen.widthX} \\ a &= \texttt{screen.aspectRatio} \end{aligned}$$

Se utiliza el siguiente procedimiento:

– se utiliza directamente la posición central:

$$centre = screen.centrePosition$$

– se calculan la anchura y la altura:

$$\begin{aligned} \texttt{width} &= \frac{w}{2} \\ \texttt{height} &= \frac{\texttt{width}}{a} \end{aligned}$$

– se definen los vectores x y z de la pantalla:

$$\begin{aligned} v_x &= \{\texttt{width}, 0, 0\} \\ v_z &= \{0, 0, \texttt{height}\} \end{aligned}$$

Para ambos tipos de pantalla, es posible crear un objeto `PolarEdges` con:

$$\begin{aligned} \texttt{left\_azimuth} &= \texttt{azimuth}(\texttt{centre} - v_x) \\ \texttt{right\_azimuth} &= \texttt{azimuth}(\texttt{centre} + v_x) \\ \texttt{bottom\_elevation} &= \texttt{elevation}(\texttt{centre} - v_z) \\ \texttt{top\_elevation} &= \texttt{elevation}(\texttt{centre} + v_z) \end{aligned}$$

7.3.3.2 Compensación de posición

En algunas disposiciones de salida, cuando `cartesian==true`, la panoramización vertical frente al oyente puede estar deformada. Esto se compensa con la función `core.screen_common.compensate_position`:

$$\texttt{compensate\_position}(\varphi, \theta, \texttt{layout}) = \begin{cases} \{\varphi', \theta\} & \text{"U} + 045\text{"} \in \texttt{layout.channel\_names} \\ \{\varphi, \theta\} & \text{en caso contrario} \end{cases}$$

donde:

– φ_r se forma mediante interpolación lineal individual de θ entre:

$$\{-90,0,30,90\}$$

y:

$$\left\{30,30,30\frac{30}{45},30\right\}$$

– φ' se forma mediante interpolación lineal individual de φ entre:

$$\{-180,-30,30,180\}$$

y:

$$\{-180,-\varphi_r,\varphi_r,180\}$$

7.3.3.3 Deformación de dirección

La deformación de las posiciones se define en `core.screen_scale.PolarScreenScaler.scale_az_el`, que deforma de manera independiente los valores de acimut y elevación. Dados los `PolarEdges ref` de la pantalla de referencia y los `PolarEdges rep` de la pantalla de reproducción, la deformación se efectúa de la siguiente manera:

– se aplica una interpolación lineal individual al acimut estableciendo una correspondencia entre los valores

$$\{-180, \texttt{ref.right\_azimuth}, \texttt{ref.left\_azimuth}, 180\}$$

y

$$\{-180, \texttt{rep.right\_azimuth}, \texttt{rep.left\_azimuth}, 180\}$$

– se aplica una interpolación lineal individual a la elevación estableciendo una correspondencia entre los valores

$$\{-90, \texttt{ref.bottom\_elevation}, \texttt{ref.top\_elevation}, 90\}$$

y

$$\{-90, \texttt{rep.bottom\_elevation}, \texttt{rep.top\_elevation}, 90\}$$

Esto se encapsula en `core.screen_scale.PolarScreenScaler.scale_position`, que aplica `scale_az_el` a las componentes acimut y elevación de un vector cartesiano, sin modificar la distancia.

7.3.3.4 Interpretación de metadatos

Si `screenRef` está configurado y se facilita la pantalla de reproducción, la posición se transmite mediante `PolarScreenScaler.scale_direction` con la configuración de la pantalla de referencia y la pantalla de reproducción. En caso contrario, se devuelve la posición sin modificar.

Si `screenRef` no está configurado o no se facilita la pantalla de reproducción, la posición se devuelve sin modificar. En caso contrario, el comportamiento depende de la bandera `cartesian`:

– Si `cartesian` está configurado, se aplican la adaptación polar y la compensación utilizando la conversión descrita en el § 10.1, lo que da una nueva posición $\{x', y', z'\}$:

$$\{\varphi, \theta, d\} = \texttt{point\_cart\_to\_polar}(\texttt{position.x}, \texttt{position.y}, \texttt{position.z})$$
$$\{\varphi_s, \theta_s\} = \texttt{scale\_az\_el}(\varphi, \theta)$$
$$\{\varphi_{sc}, \theta_{sc}\} = \texttt{compensate\_position}(\varphi_s, \theta_s, \texttt{layout})$$
$$\{x', y', z'\} = \texttt{point\_cart\_to\_polar}(\varphi_{sc}, \theta_{sc}, d)$$

– En caso contrario, se aplica `scale_az_el` a las componentes acimut y elevación de la posición.

7.3.4 Fijación de bordes de pantalla

El componente fijación de bordes de pantalla deforma las posiciones fuente para situar la fuente en el borde indicado de la pantalla. La interfaz es la siguiente:

```
class ScreenEdgeLockHandler {
  ScreenEdgeLockHandler(Screen reproduction_screen);

  CartesianPosition handle_vector(
    CartesianPosition position,
    ScreenEdgeLock screen_edge_lock,
    cartesian=False
  );

  tuple<float, float> handle_az_el(
    float azimuth,
    float elevation,
    ScreenEdgeLock screen_edge_lock
  );
};
```

En la inicialización, este componente transforma `reproduction_screen` en un objeto `polar_edges PolarEdges`, como se especifica en el § 7.3.3.1.

`handle_az_el` modifica independientemente el acimut y la elevación, lo que da como resultado un nuevo acimut y una nueva elevación:

– Si `screen_edge_lock.horizontal` es `LEFT`, el acimut se pone a `polar_edges.left_azimuth`; si es `RIGHT`, el acimut se pone a `polar_edges.right_azimuth`; en caso contrario, el acimut no se modifica.

– Si `screen_edge_lock.vertical` es `TOP`, la elevación se pone a `polar_edges.top_elevation`; si es `BOTTOM`, la elevación se pone a `polar_edges.bottom_elevation`; en caso contrario, la elevación no se modifica.

Si no se da `reproduction_screen`, no se modifica la posición.

El procesamiento se realiza en el dominio polar, por lo que es necesario convertir en primer lugar las posiciones cartesianas. Si se utiliza el método `handle_vector` en lugar del método `handle_az_el`, se aplica una conversión de ida y vuelta.

– Si `cartesian` está configurado, se aplican la adaptación polar y la compensación utilizando la conversión descrita en el § 10.1, lo que da como resultado una nueva posición $\{x', y', z'\}$:

$$\{\varphi, \theta, d\} = \texttt{point\_cart\_to\_polar}(\texttt{position.x}, \texttt{position.y}, \texttt{position.z})$$
$$\{\varphi_s, \theta_s\} = \texttt{handle\_az\_el}(\varphi, \theta, \texttt{screen\_edge\_lock})$$
$$\{\varphi_{sc}, \theta_{sc}\} = \texttt{compensate\_position}(\varphi_s, \theta_s, \texttt{layout})$$
$$\{x', y', z'\} = \texttt{point\_cart\_to\_polar}(\varphi_{sc}, \theta_{sc}, d)$$

– En caso contrario, se aplica `handle_az_el` a las componentes acimut y elevación de la posición.

Este componente se implementa en `core.screen_edge_lock.ScreenEdgeLockHandler`.

7.3.5 Exclusión de zona cartesiana

El algoritmo de exclusión de zona cartesiana empieza con la disposición de reproducción completa como `channel_positions` y procesa los objetos `ExclusionZone` para identificar qué altavoces se han de suprimir. Este procedimiento es posterior al definido en el § 7.3.12.1. Todos los altavoces dentro de cualquiera de las regiones definidas por un objeto `ExclusionZone` se suprimen, reduciendo la fila de altavoces (que comparten las mismas coordenadas y y z) a un único altavoz, y se suprimen todos los altavoces de la fila para mantener las propiedades básicas que necesita el panoramizador de fuente puntual del § 7.3.10.

Si el procedimiento de exclusión de zona hace que se supriman todos los altavoces, no se suprime ningún altavoz.

Por último, se crea una matriz de mezclado ascendente que establece la correspondencia entre los canales de la disposición reducida y los canales de la disposición completa con ganancia unitaria.

La implementación se efectúa en `core.allocentric.apply_zone_exclusion`.

7.3.6 Fijación de canal

La fijación de canal se implementa como una transformación de posición. Si *channelLock* está configurado y hay un altavoz dentro de la gama especificada en *maxDistance*, la posición se transformará en la posición del altavoz más cercano a la posición original. En ausencia de metadatos de divergencia, extensión, exclusión de zona y difusividad, la fuente se reproducirá directamente con el altavoz seleccionado.

La firma de `objectbased._gain_calc.ChannelLockHandlerBase` es la siguiente:

```
class ChannelLockHandlerBase {
  ChannelLockHandlerBase(Layout layout);
  CartesianPosition handle(
    CartesianPosition position,
    optional<ChannelLock> channelLock,
    vector<bool> excluded,
  );
};
```

`excluded` es una máscara de exclusión de canal que indica qué altavoces se han de ignorar y se utiliza únicamente en el trayecto alocéntrico, pues sólo ahí se efectúa la fijación de canal tras la exclusión de zona.

Para el trayecto egocéntrico `ChannelLockHandlerBase` se configura en `core.objectbased._gain_calc.EgoChannelLockHandler`.
Para el trayecto alocéntrico `ChannelLockHandlerBase` se configura en `core.objectbased._gain_calc.AlloChannelLockHandler`.

En modo egocéntrico las posiciones de altavoz se consideran las posiciones de altavoz real normalizadas en `layout`, mientras que en el modo alocéntrico se trata de las posiciones conformes con `core.allocentric.positions_for_layout(layout)`, como se describe en el § 7.3.9.

Para aplicar los metadatos de fijación de canal se sigue este procedimiento:

– Si `excluded` no es `None`, no considerar los altavoces cuando `excluded[n] == True` (siendo `n` el enésimo altavoz) en los siguientes pasos.

– Si `channelLock` es `None`, devolver la `position` original.

– Si `channelLock.maxDistance` no es `None`, calcular la distancia ℓ_2 entre cada posición de altavoz y `position`, e identificar todos los altavoces (con cierto margen) cuando la distancia es inferior a `channelLock.maxDistance` con respecto a los posibles altavoces.

– Si no se identifica ningún altavoz posible, devolver `position`.

– Dentro del conjunto de altavoces posibles, identificar los altavoces más cercanos a `position`. En la configuración egocéntrica, se utiliza la distancia ℓ_2 entre `position` y cada altavoz; en la configuración alocéntrica, se utiliza la distancia ponderada entre `position` y cada altavoz. La distancia ponderada se calcula de la siguiente manera:

$$dw_i = \sqrt{w_x \times \left(x_o - x_{spkr_i}\right)^2 + w_y \times \left(y_o - y_{spkr_i}\right)^2 + w_z \times \left(z_o - z_{spkr_i}\right)^2}$$

siendo:

$$\begin{aligned} w_x &= \frac{1}{16} \\ w_y &= 4 \\ w_z &= 32 \end{aligned}$$

– Si no hay un único altavoz más cercano (con cierto margen), se escoge el altavoz con mayor prioridad del conjunto de altavoces más cercanos. El orden de prioridad de los altavoces se determina por comparación lexicográfica de la tupla:

$$\{|\theta|, \theta, |\varphi|, \varphi\}$$

siendo φ y θ el acimut y la elevación reales del altavoz. Las tuplas inferiores tienen mayor prioridad – los altavoces con las menores elevaciones absolutas tienen la mayor prioridad. Si no se puede determinar en función de la elevación, se recurre al acimut absoluto y, en su defecto, al acimut.

– Devolver la posición del altavoz elegido.

7.3.7 Divergencia

La divergencia se implementa añadiendo dos posiciones fuente adicionales, $\mathbf{p}_l$ y $\mathbf{p}_r$, a la izquierda y a la derecha de la posición fuente original, $\mathbf{p}_c$. Cada posición fuente está asociada a un valor ganancia: g_l, g_c y g_r.

Los metadatos de divergencia se interpretan en `core.objectbased.gain_calc.diverge`, con la siguiente firma:

```
tuple<vector<float>, vector<CartesianPosition>> diverge(
  CartesianPosition position,
  ObjectDivergence objectDivergence,
  bool cartesian
);
```

Esta función acepta una posición 3D (en este caso, la salida de la función fijación de canal) y aplica los metadatos de divergencia incluidos en `objectDivergence`. Se producen tres posiciones fuente con sus ganancias asociadas y cada una de ellas se transmite al panoramizador de extensión para la reproducción.

A continuación se describe el cálculo de esas ganancias y posiciones.

7.3.7.1 Cálculo de ganancias

Para un `objectDivergence.value` x dado, se calculan tres ganancias de la siguiente manera:

$$g_c = \frac{1-x}{x+1}$$

$$g_l = g_r = \frac{x}{x+1}$$

Se satisfacen así los siguientes requisitos:

– $\forall x,\ g_l + g_r + g_c = 1$

– $x = 0 \Longrightarrow g_l = g_r = 0 \wedge g_c = 1$

– $x = \frac{1}{2} \Longrightarrow g_l = g_r = g_c = \frac{1}{3}$

– $x = 1 \Longrightarrow g_l = g_r = 0{,}5 \wedge g_c = 0$

7.3.7.2 Cálculo de posiciones

Las posiciones producidas dependen de la bandera `cartesian` en el formato de bloques. Se genera una alerta si `azimuthRange` y `cartesian` están configurados o si `positionRange` está configurado y `cartesian` no lo está.

7.3.7.2.1 Comportamiento cuando `cartesian == true`

Para una `position` de valor $\mathbf{p}$ y un `objectDivergence.positionRange` de valor x, la posición central se desplaza simplemente a izquierda y derecha x a lo largo del eje x y se limita a $[-1,1]$:

$$\begin{aligned} \mathbf{p}_c &= \mathtt{clip}(\mathbf{p}, -1, 1) \\ \mathbf{p}_l &= \mathtt{clip}(\mathbf{p} - \{x, 0{,}0\}, -1, 1) \\ \mathbf{p}_r &= \mathtt{clip}(\mathbf{p} + \{x, 0{,}0\}, -1, 1) \end{aligned}$$

`clip` se define en el § 7.3.2.

7.3.7.2.2 Comportamiento cuando `cartesian == false`

Las posiciones se calculan para un `objectDivergence.azimuthRange` a dado, de manera que desde la perspectiva del oyente las fuentes izquierda y derecha están a grados a la izquierda y la derecha del centro, y las tres fuentes están en línea recta.

Esto se consigue definiendo tres posiciones centradas en torno al eje $+y$ a una distancia $d = \| \mathbf{p}_c \|_2$, siendo $\mathbf{p}_c$ la posición fuente original:

$$'_l = \mathtt{cart}(a, 0, d)$$

$$P'_r = \mathtt{cart}(-a, 0, d)$$

$$P'_c = \mathtt{cart}(0,0, d)$$

A continuación se rotan en torno a la dirección fuente original utilizando la matriz de rotación **M**, que se define de tal manera que $\mathbf{p}_c{}'$ corresponde a la posición fuente original $\mathbf{p}_c$:

$$[\mathbf{p}_l, \mathbf{p}_r, \mathbf{p}_c]^T = \mathbf{M} \cdot [\mathbf{p'}_l, \mathbf{p'}_r, \mathbf{p'}_c]^T$$

7.3.8 Panoramizador de extensión polar

Los parámetros de extensión polar ADM se tratan en `core.objectbased.gain_calc.PolarExtentHandler`, que utiliza los módulos siguientes para producir un vector ganancia para una determinada posición y unos determinados parámetros de extensión.

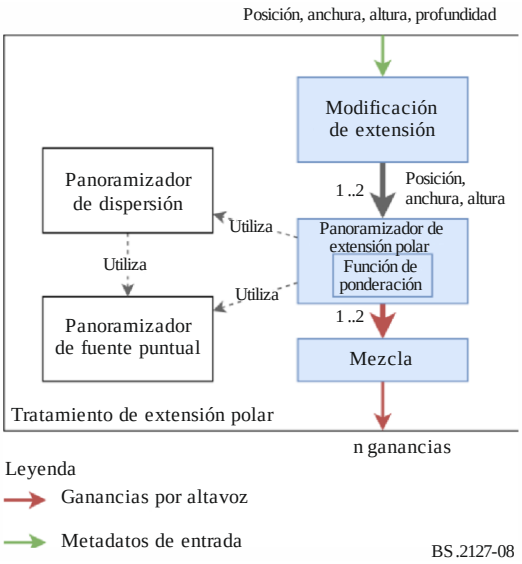
La interfaz con esta clase es:

```
class PolarExtentHandler {
  PolarExtentHandler(PointSourcePanner psp);

  vector<float> handle(
    CartesianPosition position,
    float width,
    float height,
    float depth);
};
```

La estructura de la clase `PolarExtentHandler` se muestra en la Fig. 8.

FIGURA 8

Estructura del tratamiento de extensión



A nivel interno, este objeto contiene una referencia a un `PolarExtentPanner`, como se describe en el § 7.3.8.2, que utiliza para calcular los vectores ganancia.

Los parámetros `width`, `height` y `position` se duplicarán y modificarán para tratar el parámetro `depth` y la componente distancia de `position`. Estos parámetros pasan por el panoramizador de extensión polar para generar un vector ganancia de altavoz para cada uno y, por último, se mezclan esos vectores ganancia. Este procedimiento se describe en el § 7.3.8.2.

A continuación se describen los modos de reproducción de extensión polar utilizando el panoramizador de dispersión para generar ganancias de altavoz.

7.3.8.1 Panoramizador de dispersión

La forma de las fuentes extendidas en el reproductor se define en términos de una función de ponderación, que, dada una dirección 3D, puede calcular una ponderación para esa dirección. Esta ponderación puede considerarse la cantidad en que un determinado objeto debe reproducirse en una dirección dada. Por ejemplo, para una fuente en frente del oyente más ancha que alta puede aplicarse una función de ponderación como la que se muestra en la Fig. 10.

Al generar ganancias por altavoz que reflejan esta función de ponderación, aplicar estas ganancias a la forma de onda mono de un objeto y aplicar el filtrado de descorrelación a los canales resultantes, puede lograrse una impresión de fuente sonora extendida o difusa con los parámetros de extensión deseados.

Para calcular un vector ganancia para una función de ponderación dada se utiliza la clase `SpreadingPanner`.

El conjunto de 1 652 posiciones de fuentes virtuales utilizadas en el panoramizador de dispersión se determina como sigue:

Para cada elevación θ entre –90° y 90°, inclusive, en cinco pasos, calcular el número de puntos *n* que deben espaciarse uniformemente alrededor de un círculo en esa elevación, para lograr una densidad aproximadamente uniforme en la superficie de la esfera unitaria:

$$n' = \frac{360}{5}\cos\theta$$

$$n = \max(\text{round}(n'),\ 1)$$

Luego, para cada *i* en el rango 0 a $n - 1$ inclusive, calcular el azimut φ:

$$\varphi = 360\,\frac{i}{n}$$

Esto resulta en la cesta de puntos (φ, θ, 1).

Los objetos de este tipo contienen un conjunto de posiciones fuente virtuales y un vector ganancia de altavoz para cada una de esas posiciones.

Durante el inicio, el panoramizador de fuente puntual se utiliza con los vectores ganancia calculados para cada posición.

Para calcular el vector ganancia de una función de ponderación dada, ésta se aplica a las posiciones fuente virtuales. El vector ganancia por fuente virtual resultante se multiplica por los vectores ganancia por altavoz calculados para obtener un único vector ganancia por altavoz. A continuación se aplica la normalización en potencia para obtener el vector ganancia final.

La implementación se efectúa en `core.objectbased.extent.SpreadingPanner`.

7.3.8.2 Extensión polar de reproducción

El procedimiento utilizado para calcular las ganancias de altavoz para los parámetros `position`, `width`, `height` y `depth` en modo polar es el siguiente:

– El parámetro `depth` se interpreta como dos fuentes extendidas con la misma dirección, pero distancias diferentes. Las dos distancias son:

$$d_1 = \max\left\{0, \| \texttt{position} \|_2 + \frac{\texttt{depth}}{2}\right\}$$
$$d_2 = \max\left\{0, \| \texttt{position} \|_2 - \frac{\texttt{depth}}{2}\right\}$$

– Para cada distancia se utiliza el panoramizador de extensión polar para calcular los vectores ganancia, $\mathbf{g'}_1$ y $\mathbf{g'}_2$, de `position`, y `width` y `height` modificados por la función de modificación de extensión polar, como se describe a continuación.

– Los vectores ganancia se mezclan para producir el vector ganancia de salida, **g**, siendo $\mathbf{g}_i$ la ganancia para el altavoz i:

$$\mathbf{g}_i = \sqrt{\frac{\mathbf{g'}_{1,i}{}^2 + \mathbf{g'}_{2,i}{}^2}{2}}$$

7.3.8.2.1 Función de modificación de extensión polar

La función de modificación de extensión se utiliza para modificar los parámetros anchura y altura con un parámetro distancia dado.

Sus propiedades son las siguientes:

– A $\texttt{distance} = 0$, la extensión es siempre 360°.

– A $\texttt{distance} = 1$, se utiliza la extensión original.

– A $\texttt{distance} > 1$, la extensión disminuye a medida que aumenta la distancia.

– Cuando $0 < \texttt{distance} < 1$, la extensión cambia más pronunciadamente en torno a $\texttt{distance} = 0$ para las extensiones más pequeñas.

La función de modificación de extensión para `extent` y `distance` se define de la siguiente manera:

– Se establece una correspondencia lineal entre la extensión en grados y la extensión a lo largo del eje x, con un tamaño mínimo de:

$$\texttt{min\_size} = 0{,}2$$

$$\texttt{size} = \texttt{min\_size} + \frac{(1-\texttt{min\_size})\times\texttt{extent}}{360°}$$

– Se forma un triángulo rectángulo donde el borde adyacente es la distancia y el borde opuesto es la distancia. El ángulo formado se utiliza para determinar una nueva extensión, que se calcula para una distancia de 1 y `distance`:

$$e_1 = 4 \times \frac{180}{\pi} \times \text{atan2}(\texttt{size}, 1)$$
$$e_d = 4 \times \frac{180}{\pi} \times \text{atan2}(\texttt{size}, \texttt{distance})$$

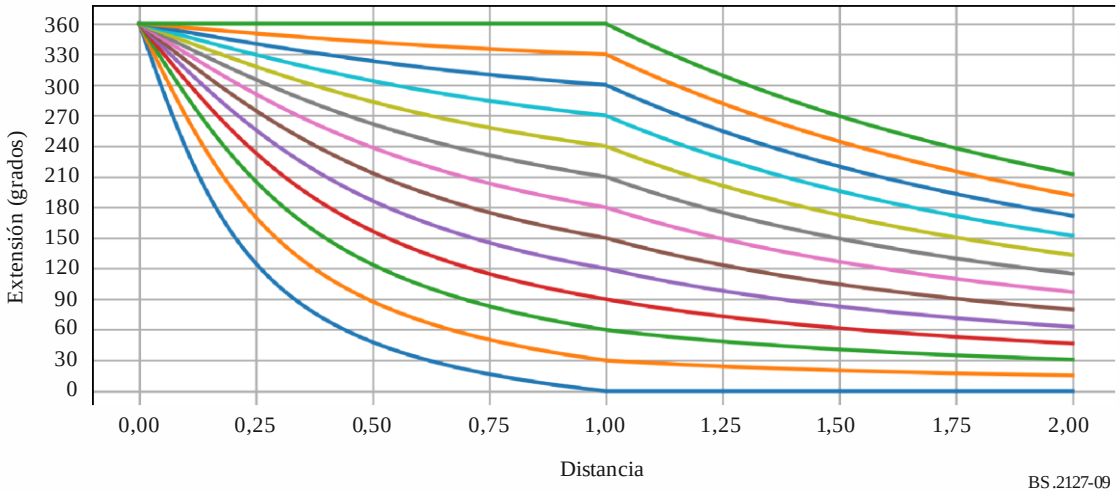
– Se aplica una interpolación lineal individual para establecer la correspondencia entre e_d y la extensión original cuando $e_d = e_1$:

$$\texttt{extent\_mod} = \begin{cases} \texttt{extent} \times \frac{e_d}{e_1} & e_d < e_1 \\ \texttt{extent} + (360° - \texttt{extent}) \times \frac{e_d - e_1}{360° - e_1} & e_d \geq e_1 \end{cases}$$

La implementación se efectúa en `core.objectbased.gain_calc.PolarExtentHandler.extent_mod`. La forma de la función de modificación de extensión se muestra en la Fig. 9.

FIGURA 9

Función de modificación de extensión para fuentes extendidas polares



NOTA – Cada línea muestra cómo varía la extensión de salida con la distancia para una determinada extensión de entrada. La extensión no se modifica cuando $\texttt{distance} = 1$, de modo que, por ejemplo, la línea inferior muestra cómo varía la extensión modificada con la distancia para una extensión de entrada 0.

7.3.8.2.2 Panoramizador de extensión polar

Para tratar toda la gama de posiciones y extensiones permitidas en ADM, el tamaño se ha de modificar antes de poder aplicar la función de ponderación polar. Se siguen estos pasos:

– Se calculan una anchura y una altura modificadas como $max\{width, 5°\}$ y $max\{height, 5°\}$, que se utilizan con el panoramizador de dispersión descrito en el § 7.3.8.1 y la función de ponderación polar descrita a continuación para obtener un vector ganancia de dispersión, g_s.

– La posición se transmite al panoramizador de fuente puntual para obtener un vector ganancia de fuente puntual, g_p.

Los dos vectores se mezclan para generar un vector g tal que para una anchura y una altura cero, sólo se utilizan las ganancias de fuente puntual, mientras que, si la altura o la anchura son superiores a 5 grados, se utilizan únicamente las ganancias de dispersión:

$$g_i = \sqrt{p g_{s,i}^2 + (1-p) g_{p,i}^2}$$

donde:

$$p = clip\left(\frac{max(width, height)}{5}, 0{,}1\right)$$

Esto es necesario para soportar extensiones pequeñas – aquí la parte no cero de la función de dispersión debe ser lo suficientemente amplia para abarcar múltiples puntos de muestreo a fin de producir ganancias paulatinas, y para ello se necesita una cantidad de dispersión mínima, que puede ser superior a la cantidad deseada.

La implementación se efectúa en `core.objectbased.extent.PolarExtentPanner.calc_pv_spread`.

7.3.8.2.3 Función de ponderación polar

La función de ponderación para la reproducción de extensiones polares se parametriza con un vector cartesiano 3D `position` y ángulos `width` y `height` en grados. Dado que no se utiliza la componente distancia de la posición, ésta puede considerarse como una dirección.

La función de ponderación es la siguiente:

– Se calcula una matriz de rotación que establece la correspondencia entre la posición {0,1,0} (directamente frente al oyente) con la posición de la fuente. Esta matriz de rotación toma la forma de una rotación en torno a {1,0,0} seguida de una rotación en torno a {0,0,1}. La implementación se efectúa en `core.objectbased.extent.calc_basis`.

– Si la altura es superior a la anchura, el sistema de coordenadas se cambia para simplificar el cálculo, pues la función de ponderación para una fuente con anchura w y altura h debe ser idéntica a la función de ponderación para una fuente con anchura h y algura w, rotada 90° en torno a la posición fuente. Esto se consigue intercambiando las variables anchura y altura e intercambiando las filas x y z de la matriz de rotación. Véanse, por ejemplo, las Figs. 10 y 11, que tienen la misma forma, pero están rotadas 90 grados (ignorando la deformación causada por la proyección utilizada).

– La función de ponderación aproximada es ahora 1 dentro de un rectángulo `width` × `height` redondeado al máximo (estado) en un espacio acimut-elevación con algunas modificaciones:

- Los bordes redondeados son circulares en espacio cartesiano, pues la ponderación se calcula en función del ángulo de dos vectores en su centro. Cuando `width = height`, la función de ponderación es circular.
- A `width` > 180°, la anchura se incrementa de manera que, cuando la anchura alcanza 360°, las partes redondeadas se solapan por completo, formando una «banda», donde la función de ponderación tiene el mismo valor para todas las posiciones de idéntica elevación. Véanse las Figs. 12 y 13.
- Se añade un desvanecimiento al borde de la función de ponderación; la ponderación cae de 1 a 0 a medida que la distancia angular desde la extensión alcanza 10 grados.

Esta función se implementa en `core.objectbased.extent.PolarExtentPanner.get_weight_func`.

FIGURA 10

Función de ponderación polar para `width` = 90° y `height` = 30°

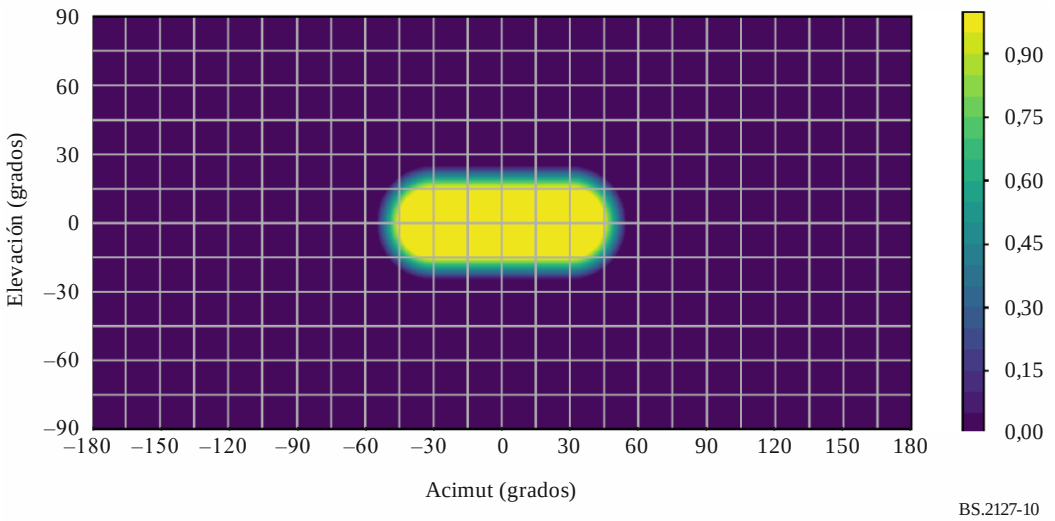


FIGURA 11

Función de ponderación polar para `width` = 30° y `height` = 90°

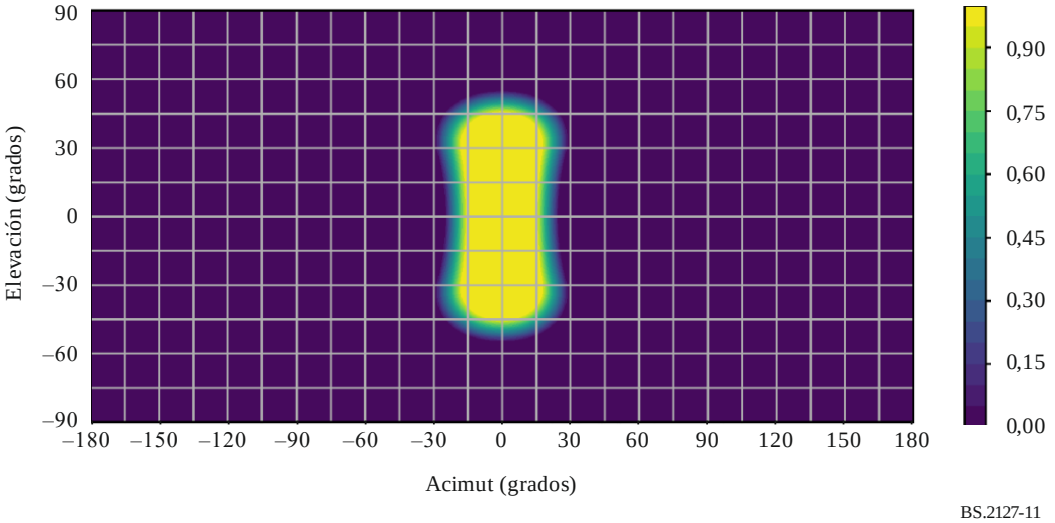


FIGURA 12

Función de ponderación polar para `width` = 300° y `height` = 30°

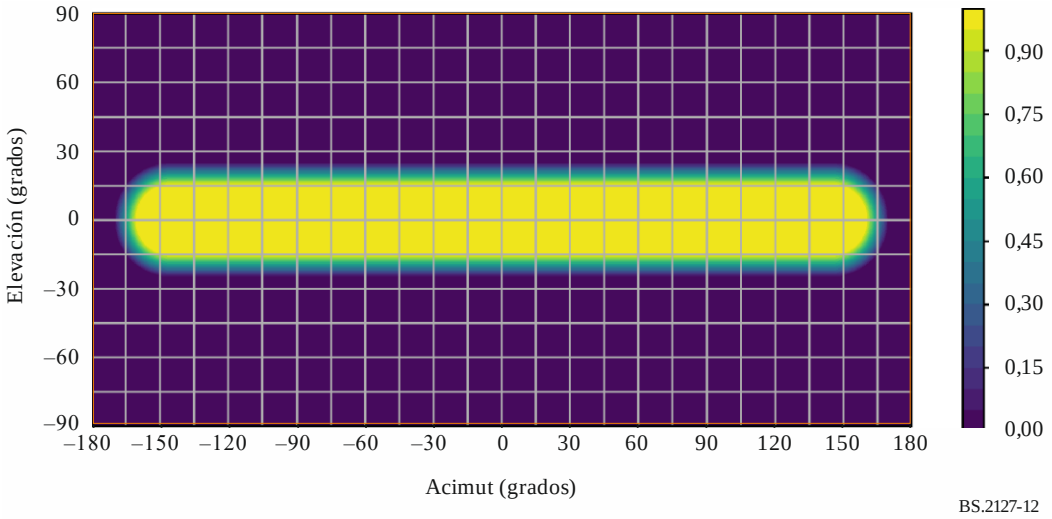
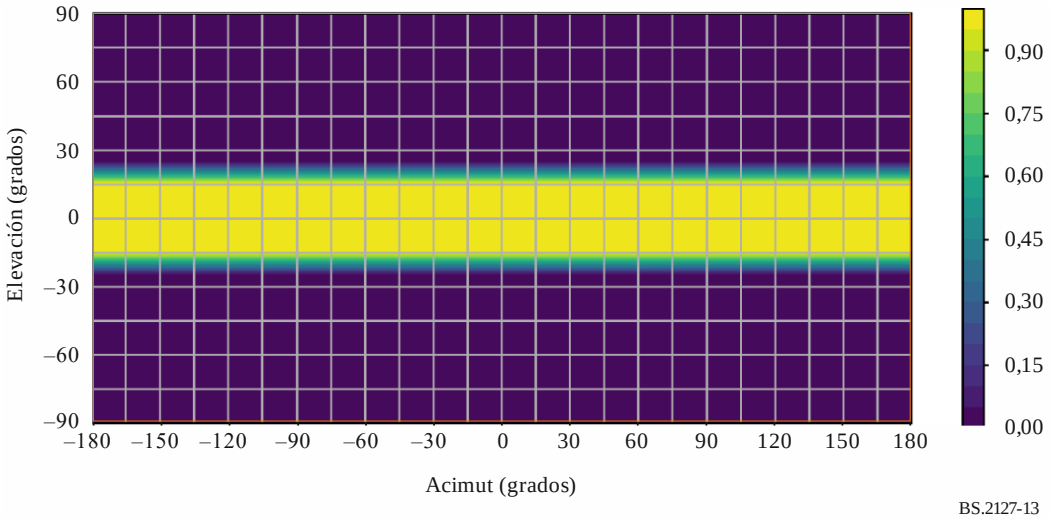


FIGURA 13

Función de ponderación polar para `width` = 360° y `height` = 30°



7.3.9 Posiciones de altavoz cartesianas

Para utilizar el panoramizador de fuente puntual cartesiano especificado en el § 7.3.10, se ha de encontrar una posición cartesiana para cada altavoz en la disposición.

La interfaz con este componente es la siguiente:

```
vector<CartesianPosition> positions_for_layout(Layout layout)
```

En primer lugar, el cuadro de posiciones correspondientes a `layout.name` puede encontrarse en el § 11.2.

Para cada `channel` de `layout.channels`, los parámetros `x`, `y` y `z` de una `CartesianPosition` de salida se determinan de la siguiente manera:

– Si `channel.name` es `M+SC` o `M-SC`:

$$\{x, y, z\} = \texttt{point\_polar\_to\_cart}(\texttt{channel.polar\_position.azimuth}, 0, 1)$$

Téngase en cuenta que se supone una precisión infinita en `point_polar_to_cart`. En la práctica, las posiciones se deben modificar de manera que:

- $z = 0$;
- las coordenadas y de ambos altavoces de pantalla sean idénticas;
- las coordenadas x de ambos altavoces de pantalla sean exáctamente simétricas en torno a 0.

– En caso contrario, los valores son los de la fila `channel.name`.

La implementación se efectúa en `core.allocentric`.

7.3.10 Panoramizador de fuente puntual cartesiano

El algoritmo de panoramización puntual cartesiano está formado por una extensión 3D del panoramizador de «balance dual», que se utiliza ampliamente en la producción de sonido ambiental en canal 5.1 y 7.1.

Las entradas del panoramizador son una posición de objeto $[p_{ox}, p_{oy}, p_{oz}]$ y las posiciones de N altavoces de salida, todas ellas en coordenadas cartesianas. Supongamos que $[p_{sx}(j), p_{sy}(j), p_{sz}(j)]$ denota la posición del altavoz j.

En lo que respecta a la disposición de altavoces, el panoramizador de fuente puntual necesita que se cumplan las siguientes condiciones para poder situar con precisión una imagen fantasma del objeto en cualquier punto de la sala:

– Los altavoces deben estar agrupados en uno o más planos discretos en la dimensión z.

– Los altavoces de cada plano deben estar agrupados en una o más filas discretas en la dimensión y.

– En cualquier fila en que $-1 < y < 1$ (es decir, cualquier fila que no tenga intersección con las paredes frontal o posterior de la sala), debe haber altavoces en $x = 1$ y $x = -1$.

– Todos los altavoces deben estar situados en la superficie de la sala cúbica, es decir, en el suelo, las paredes o el techo.

– Las posiciones que cumplen esas condiciones pueden hallarse con el procedimiento del § 7.3.9.

Aproximadamente, las ganancias de altavoz para una posición fuente dada se hallan de la siguiente manera:

– Se hallan las capas de altavoces por encima y por debajo de la fuente y se calcula una ganancia z para ambas capas en función de la posición z de las capas y la fuente.

– En cada una de las capas halladas, se halla una fila de altavoces en frente y detrás de la posición fuente y se calcula una ganancia y para cada una de esas filas en función de la posición y de las capas y la fuente.

– En cada una de las filas halladas, se halla un par de altavoces a la izquierda y la derecha de la posición fuente y se calcula una ganancia x para cada uno de esos altavoces en función de la posición x de los altavoces y la fuente.

Se habrán seleccionado hasta ocho altavoces: cada uno de ellos tendrá una ganancia de $x \times y \times z$; los demás altavoces tendrán una ganancia cero.

La especificación exacta del algoritmo se muestra a continuación, calculando una ganancia $g^{point}(j_x, j_y, j_z)$ para cada altavoz j. Habida cuenta de que cada eje es separable, también conviene observar que $g^{point}(x, y, z) = g^{point_x}(x) \times g^{point_y}(y) \times g^{point_z}(z)$, y que las tres ganancias independientes pueden utilizarse como valores intermedios en el algoritmo.

```
epsilon = 0.001 //small positive constant

//simplification: Use object-centric coordinates, so that object is
//always at the origin.
for (j = 1 to N)
{
  p_sx(j) -= p_ox
  p_sy(j) -= p_oy
  p_sz(j) -= p_oz
}

for (j = 1 to N)
{
  //Z-gain
  z_this = p_sz(j)
  //find loudspeakers in other plane, on other side of object
  if (z_this >= 0) {
    z_other = max({p_sz : p_sz < z_this})
  } else {
    z_other = min({p_sz : p_sz > z_this})
  }
  if (isempty(z_other)) {
    gz = 1.0
  } else if (sign(z_other) == sign(z_this)) {
    gz = 0.0
  } else {
    gz = cos(z_this / (z_other - z_this) * pi /2)
  }

  //Y-gain
  //from among loudspeakers in this plane...
  p_sx_plane = p_sx({i:abs(p_sz(i) - z_this) < epsilon})
  p_sy_plane = p_sy({i:abs(p_sz(i) - z_this) < epsilon})
  y_this = p_sy(j)
  //...find loudspeakers in closest row, on other side of object
  if (y_this >= 0) {
    y_other = max({p_sy_plane : p_sy_plane < y_this})
  } else {
    y_other = min({p_sy_plane : p_sy_plane > y_this})
  }
  if isempty(y_other) {
    gy = 1.0
  } else if (sign(y_other) == sign(y_this)) {
    gy = 0.0
  } else {
    gy = cos(y_this / (y_other - y_this) * pi /2)
  }

  //X-gain
  //Among loudspeakers in this plane and row...
  p_sx_row = p_sx_plane({i:abs(p_sy_plane(i) - y_this) < epsilon})
  x_this = p_sx(j)
  //find loudspeakers in the closest column
  if (x_this >= 0) {
    x_other = max({p_sx_row : p_sx_row < x_this})
  } else {
    x_other = min({p_sx_row : p_sx_row > x_this})
  }
```

```
    if (isempty(x_other)) {
      gx = 1.0
    } else if (sign(x_other) == sign(x_this)) {
      gx = 0.0
    } else {
      gx = cos(x_this / (x_other - x_this) * pi /2)
    }
    g_point(j) = gx * gy * gz
  }
```

Obsérvese que, como máximo, habrá ocho altavoces con ganancia no cero y que la suma de los cuadrados de las ganancias de altavoz siempre será 1, por lo que la operación de panoramización conserva la energía.

La implementación se efectúa en `core.point_source.AllocentricPanner`.

7.3.11 Panoramizador de extensión cartesiano

El objetivo del panoramizador de extensión es calcular un coeficiente de ganancia para cada altavoz de la disposición de altavoces de salida a partir de una posición de objeto y de extensiones de objeto. El objetivo de la extensión es que el objeto aparezca más grande de manera que, cuando está al máximo, el objeto llene la sala y, cuando esté a cero, el objeto se reproduzca como un objeto puntual.

Para ello, el panoramizador de extensión considera una red de múltiples fuentes virtuales en la sala. Cada fuente virtual activa los altavoces exactamente de la misma manera que lo haría cualquier objeto reproducido con el panoramizador de fuente puntual. Cuando se le da una posición de objeto y extensiones de objeto, el panoramizador de extensión determina cuáles (y cuántas) de esas fuentes virtuales contribuirán.

Para calcular las ganancias de un objeto con extensión se han de aplicar los pasos siguientes, que se explican más detalladamente en las siguientes subcláusulas.

1) Preadaptación de los parámetros de extensión.
2) Cálculo de ganancias puntuales para todas las fuentes virtuales.
3) Combinación de todas las ganancias de fuentes virtuales dentro de la sala para producir ganancias de extensión internas.
4) Combinación de todas las ganancias de fuentes virtuales en los límites de la sala para producir ganancias de extensión limítrofes.
5) Combinación de las ganancias de extensión internas y limítrofes para producir las ganancias de extensión finales.
6) Combinación de las ganancias de extensión finales con las ganancias puntuales del objeto.

El panoramizador de extensión cartesiano se implementa en `core.objectbased.allo_extent.get_gains`.

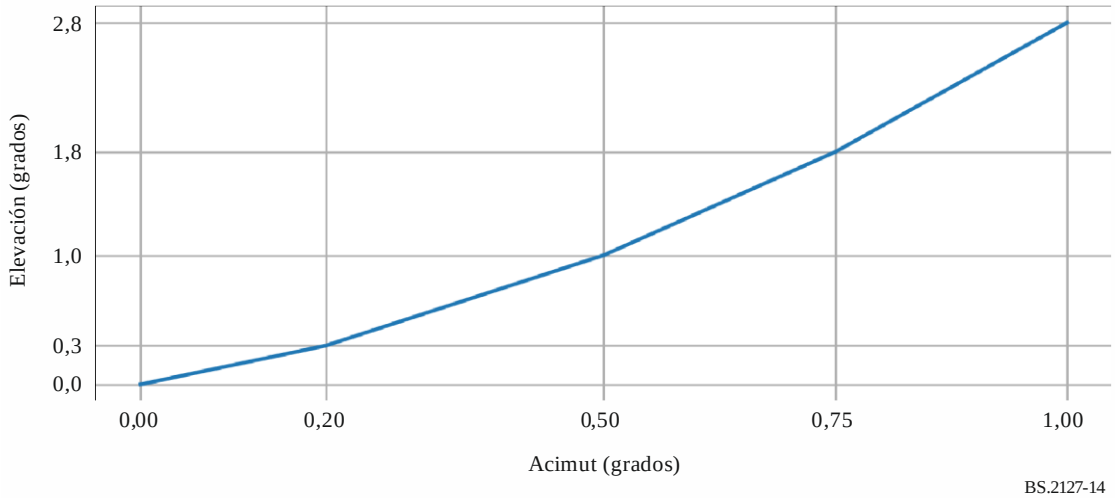
7.3.11.1 Preadaptación de los parámetros de extensión

Antes de calcular las ganancias se adaptan los valores de los parámetros para que la función de ponderación de fuente se comporte de manera más intuitiva. El usuario se expone a unos valores $s \in [0,1]$, cuya correspondencia se establece con la extensión real utilizada por el algoritmo dentro de la gama $[0,2.8]$. La correspondencia se efectúa con una función lineal individual definida por los pares de valores $(0,0), (0.2,0.3), (0.5,1.0), (0.75,1.8), (1,2.8)$, que se muestra en la Fig. 14. El valor máximo de 2,8 garantiza que, cuando la extensión está puesta al máximo (1.0), verdaderamente ocupa

toda la sala. A continuación las variables $\hat{s}_x, \hat{s}_y, \hat{s}_z$ se refieren a los valores de extensión de entrada una vez efectuada la correspondencia.

FIGURA 14

Correspondencia lineal individual entre parámetros de extensión ADM y valores de extensión internos del algoritmo



Para mantener el comportamiento deseado cuando los valores de extensión son extremos, se aplican valores mínimos a $\hat{s}_x, \hat{s}_y, \hat{s}_z$ como se indica a continuación:

$$s_x = max\left(\hat{s}_x, \frac{2}{N_x - 1}\right), s_y = max\left(\hat{s}_y, \frac{2}{N_y - 1}\right), s_z = max\left(\hat{s}_z, \frac{2}{N_z - 1}\right)$$

Estos valores s_x, s_y, s_z restringuidos se utilizan en todo el algoritmo.

7.3.11.2 Cálculo de las ganancias de fuente virtual

La cuadrícula de fuentes virtuales se define como una cuadrícula uniforme rectangular estática de $N_x \times N_y \times N_z$ puntos. La cuadrícula abarca toda la gama de posiciones $[-1,1]$ en todas las dimensiones. La densidad se ha de configurar de manera que incluya unas pocas fuentes entre altavoces en una disposición típica. Las pruebas empíricas demuestran que $N_x = N_y = N_z = 40$ crea una cuadrícula de fuentes virtuales adecuada[1]. La notación (x_s, y_s, z_s) se utilizará para denotar las coordenadas posibles de las fuentes virtuales. Cada fuente virtual crea un conjunto de ganancias $g_j^{point}(x_s, y_s, z_s)$ para cada altavoz $j = 1, \ldots, N_j$ de la disposición, de acuerdo con el algoritmo de panoramización de fuente puntual cartesiana descrito en el § 7.3.10. Téngase en cuenta que, de haberse excluido altavoces de la disposición a causa del objeto exclusión de zona (véase el § 7.3.5), para calcular las ganancias se utiliza la disposición de altavoces reducida.

[1] En las disposiciones de altavoces sin altavoces en la capa inferior, la gama de fuentes virtuales en la dimensión Z se limita a $[0,1]$ y el valor recomendado de N_z es 20.

7.3.11.3 Combinación de ganancias de fuente virtual dentro de la sala

La posición y la extensión del objeto $(x_o, y_o, z_o, s_x, s_y, s_z)$ se utilizan para calcular un conjunto de ponderaciones que determinan cuánto contribuirá cada fuente virtual a las ganancias finales[2]. Las ponderaciones de cada fuente virtual se denotan mediante $w(x_s, y_s, z_s, x_o, y_o, z_o, s_x, s_y, s_z)$ y se utilizan para adaptar las ganancias puntuales de cada fuente virtual. Una vez ponderadas, se suman todas las ganancias de fuente virtual para obtener las ganancias de extensión internas:

$$g_j^{inside}(x_o, y_o, z_o, s_x, s_y, s_z) = \sum_{x_s,y_s,z_s} w\,(x_s, y_s, z_s, x_o, y_o, z_o, s_x, s_y, s_z) \times g_j^{point}(x_s, y_s, z_s)$$

Sin embargo, el algoritmo de extensión combina las ganancias de fuente virtual de manera variable en función de la extensión del objeto. En general puede describirse como:

$$g_j^{inside}(x_o, y_o, z_o, s_x, s_y, s_z) = \left[\sum_{x_s,y_s,z_s} \left[w(x_s, y_x, z_s, x_o, y_o, z_o, s_x, s_y, s_y) \times g_j^{point}(x_s, y_s, z_s)\right]^p\right]^{\frac{1}{p}}$$

El exponente dependiente de la extensión, p, controla la progresividad de las ganancias entre altavoces. Garantiza el crecimiento homogéneo del objeto cuando s es pequeño y corrige la distribución de energía en todas las direcciones cuando s es grande. Para calcular p, en primer lugar se ordenan $\{s_x, s_y, s_z\}$ en orden descendente y se etiqueta el trío ordenado resultante: $\{s_1, s_2, s_3\}$. El trío puede entonces combinarse para obtener la extensión efectiva:

$$s_{eff} = \frac{6}{9}s_1 + \frac{2}{9}s_2 + \frac{1}{9}s_3$$

En disposiciones con un único plano de altavoces, como `0+5+0`, o si la exclusión de zona hace que la disposición se reduzca a un único plano, en primer lugar se ordenan $\{s_x, s_y\}$ en orden descendente y se etiqueta el dúo ordenado resultante: $\{s_1, s_2\}$, con lo que se obtiene:

$$s_{eff} = \frac{3}{4}s_1 + \frac{1}{4}s_2$$

Para la disposición `0+2+0` (estéreo) o si la exclusión de zona reduce el conjunto de altavoces a una única fila, $s_{eff} = s_x$.

La extensión efectiva se utiliza entonces para calcular un exponente definido individual:

$$p = \begin{cases} 6 & s_{eff} \le 0{,}5 \\ 6 - 4 \times \frac{s_{eff} - 0{,}5}{s_{max} - 0{,}5} & en\ caso\ contrario \end{cases}$$

siendo $s_{max} = 2{,}8$, de manera que, cuando s está al máximo, $p = 2$.

---

[2] En las disposiciones de altavoces en las que no hay altavoces en la capa inferior, el algoritmo de extensión utiliza $z_o = max(p_{oz}, 0)$ como posición de los objetos en la dimensión Z. En caso contrario, utiliza $z_o = p_{oz}$. En todas las disposiciones de altavoces, el algoritmo de extensión utiliza las mismas posiciones X e Y que el panoramizador de fuente puntual (es decir, $y_o = p_{oy}, x_o = p_{ox}$).

La función de ponderación también puede tratar cada eje por separado y el cálculo de toda la extensión se simplifica si se utilizan funciones de ponderación separables:

$$w(x_s, y_s, z_s, x_o, y_o, z_o, s_x, s_y, s_z) = w_x(x_s, x_o, s_x) w_y(y_s, y_o, s_y) w_z(z_s, z_o, s_z)$$

Las funciones escogidas tienen una forma entre circular y cuadrada (o esférica y cúbica en 3D):

$$\begin{aligned} w_x(p, o, s) = w_y(p, o, s) &= 10^{-\min\left(\left[\frac{3}{2}\left(\frac{p-o}{2s}\right)\right]^4, 6{,}5\right)} \\ w_z(p, o, s) &= 10^{-\min\left(\left[\frac{3}{2}\left(\frac{p-o}{s}\right)\right]^4, 6{,}5\right)} \times \cos\left(s\frac{3\pi}{7}\right) \end{aligned}$$

Esto significa que g_j^{inside} puede simplificarse a

$$g_j^{inside}(x_o, y_o, z_o, s_x, s_y, s_z) = f_j^x(x_o, s_x) f_j^y(y_o, s_y) f_j^z(z_o, s_z)$$

siendo:

$$\begin{aligned} f_j^x(x_o, s_x) &= \sum_{x_s}\left[g_j^{point_x}(x_s) w_x(x_s, x_o, s_x)\right]^p \\ f_j^y(y_o, s_y) &= \sum_{y_s}\left[g_j^{point_y}(y_s) w_y(y_s, y_o, s_y)\right]^p \\ f_j^z(z_o, s_z) &= \sum_{z_s}\left[g_j^{point_z}(z_s) w_z(z_s, z_o, s_z)\right]^p \end{aligned}$$

Téngase en cuenta que, para disposiciones limitadas a un único plano de altavoces, $f_j^z(z_o, s_z) = 1$, y a una única fila de altavoces, $f_j^z(z_o, s_z) = f_j^y(y_o, s_y) = 1$.

Además, los valores muy pequeños de $f_j(c, s)$ $(10^{-6,5})$ se redondean a cero para evitar el desbordamiento de punto flotante en las implementaciones.

Se aplica la normalización a g_j^{inside}:

$$\tilde{g}_j^{inside} = \begin{cases} \dfrac{g_j^{inside}}{\sqrt{\sum_n\left[g_n^{inside}\right]^2}} & \sqrt{\sum_n\left[g_n^{inside}\right]^2} > tol \\ 0 & en\ caso\ contrario \end{cases}$$

siendo $tol = 10^{-5}$.

7.3.11.4 Combinación de ganancias limítrofes

Otra modificación es que, por motivos estéticos, resulta importante disponer de un modo en que no haya un altavoz opuesto activado. Esto se logra utilizando fuentes virtuales situadas únicamente en los límites. Para tratar determinadas disposiciones de altavoz como casos especiales:

- $dim = 1$ para disposiciones con una única fila de altavoces tras la aplicación de la exclusión de zona (por ejemplo, `0+2+0`);
- $dim = 2$ para disposiciones con un único plano de altavoces tras la aplicación de la exclusión de zona (por ejemplo, `0+5+0`);
- $dim = 4$ para disposiciones con más de dos planos de altura de altavoces distintos tras la aplicación de la exclusión de zona (por ejemplo, `3+7+0` y `9+10+3`); y
- $dim = 3$ en caso contrario.

Así, la ganancia limítrofe es:

$$\begin{aligned} g_j^{bound}(x_o, y_o, z_o, s_x, s_y, s_z) \quad &= b_j^{floor}(z_o, s_z) f_j^x(x_o, s_x) f_j^y(y_o, s_y) \\ &+ b_j^{ceil}(z_o, s_z) f_j^x(x_o, s_x) f_j^y(y_o, s_y) \\ &+ b_j^{left}(x_o, s_x) f_j^y(y_o, s_y) f_j^z(z_o, s_z) \\ &+ b_j^{right}(x_o, s_x) f_j^y(y_o, s_y) f_j^z(z_o, s_z) \\ &+ b_j^{front}(y_o, s_y) f_j^x(x_o, s_x) f_j^z(z_o, s_z) \\ &+ b_j^{back}(y_o, s_y) f_j^x(x_o, s_x) f_j^z(z_o, s_z) \end{aligned}$$

siendo:

$$\begin{aligned} b_j^{floor}(z_o, s_z) \quad &= \begin{cases} \left[g_j^{point}(z_s = -1{,}0) w(-1{,}0, z_o, s_z)\right]^p & dim = 4 \\ 0 & \text{en caso contrario} \end{cases} \\ b_j^{ceil}(z_o, s_z) \quad &= \begin{cases} \left[g_j^{point}(z_s = 1{,}0) w(1{,}0, z_o, s_z)\right]^p & dim \geq 3 \\ 0 & \text{en caso contrario} \end{cases} \\ b_j^{left}(x_o, s_x) \quad &= \left[g_j^{point}(x_s = -1{,}0) w(-1{,}0, x_o, s_x)\right]^p \\ b_j^{right}(x_o, s_x) \quad &= \left[g_j^{point}(x_s = 1{,}0) w(1{,}0, x_o, s_x)\right]^p \\ b_j^{front}(y_o, s_y) \quad &= \begin{cases} \left[g_j^{point}(y_s = 1{,}0) w(1{,}0, y_o, s_y)\right]^p & dim > 1 \\ 0 & \text{en caso contrario} \end{cases} \\ b_j^{back}(y_o, s_y) \quad &= \begin{cases} \left[g_j^{point}(y_s = -1{,}0) w(-1{,}0, y_o, s_y)\right]^p & dim > 1 \\ 0 & \text{en caso contrario} \end{cases} \end{aligned}$$

7.3.11.5 Combinación de las ganancias internas y limítrofes

En este momento es necesario combinar las ganancias limítrofes con las ganancias internas, por lo que se introduce un factor de desvanecimiento para todas las fuentes virtuales dentro de la sala, con una cantidad de desvanecimiento = «fracción del objeto fuera de la sala».

El resultado es:

$$g_j^{extent} = \left[\tilde{g}_j^{bound} + \left(\mu \times \tilde{g}_j^{inside}\right)\right]^{\frac{1}{p}}$$

siendo:

$$d_{bound} = \begin{cases} \min(x_o + 1{,}1 - x_o) & dim = 1 \\ \min(x_o + 1{,}1 - x_o, y_o + 1{,}1 - y_o) & dim = 2 \\ \min(x_o + 1{,}1 - x_o, y_o + 1{,}1 - y_o, z_o + 1, z_o - 1) & \text{en caso contrario} \end{cases}$$

$$\mu = \begin{cases} h(x_o, s_x)^3 & dim = 1 \\ h(x_o, s_x) h\left(y_o, s_y\right)^{\frac{3}{2}} & dim = 2 \\ h(x_o, s_x) h\left(y_o, s_y\right) h(z_o, s_z) & \text{en caso contrario} \end{cases}$$

y siendo $h(c, s)$ una función de desvanecimiento para una única dimensión.

$$h(c,s)=\begin{cases}\left[\frac{\max(2s,0{,}4)^3}{0{,}16\times 2s}\right]^{\frac{1}{3}} & d_{bound}\geq s\wedge d_{bound}\geq 0{,}4\\ \left[\frac{d_{bound}}{2}\left(\frac{d_{bound}}{0{,}4}\right)^2\right]^{\frac{1}{3}} & \text{en caso contrario}\end{cases}$$

A medida que parte del objeto extendido sale de la sala, todas las fuentes virtuales dentro del objeto empiezan a desvanecerse, a excepción de las que se encuentran en los límites. Cuando un objeto llega al límite, sólo las ganancias limítrofes contribuirán a las ganancias de extensión. d_{bound} es la distancia mínima con respecto a un límite.

Se aplica la normalización a g_j^{extent}

$$\widetilde{g}_j^{extent}=\begin{cases}\frac{g_j^{extent}}{\sqrt{\sum_n[g_n^{extent}]^2}} & \sqrt{\sum_n[g_n^{extent}]^2}>tol\\ 0 & \text{en caso contrario}\end{cases}$$

7.3.11.6 Combinación de las ganancias de extensión y las ganancias puntuales

Se combinan las ganancias de extensión con las ganancias puntuales y el desvanecimiento mutuo que se causan se aplica como una función de extensión:

$$g_j^{total}=\left(\alpha\times g_j^{point}(x_o,y_o,z_o)\right)+\left(\beta\times\tilde{g}_j^{extent}\right)$$

siendo:

$$\alpha=\begin{cases}\cos\left(\frac{s_{eff}}{s_{fade}}\times\frac{\pi}{2}\right) & s_{eff}<s_{fade}\\ 0 & \text{en caso contrario}\end{cases}$$

$$\beta=\begin{cases}\sin\left(\frac{s_{eff}}{s_{fade}}\times\frac{\pi}{2}\right) & s_{eff}<s_{fade}\\ 1 & \text{en caso contrario}\end{cases}$$

y $s_{fade}=0{,}2$.

Esto garantiza la panoramización y el crecimiento paulatino del objeto para una buena transición entre las extensiones más pequeñas y más grandes posibles.

Por último, se aplica la normalización a las ganancias:

$$G_j^S=\begin{cases}\frac{g_j^{total}}{\sqrt{\sum_n[g_n^{total}]^2}} & \sqrt{\sum_n[g_n^{total}]^2}>tol\\ 0 & \text{en caso contrario}\end{cases}$$

7.3.12 Exclusión de zona polar

La exclusión de zona se aplica mezclando el vector ganancia de altavoz generado al calcular la ganancia a fin de no enviar salidas a los altavoces de la zona excluida. Este procedimiento puede dividirse en dos partes: decidir qué altavoces se encuentran en la zona excluida (§ 7.3.12.1) y calcular la mezcla para evitar los altavoces excluidos (§ 7.3.12.2).

Tanto la selección de altavoces excluidos como el cálculo de la matriz de mezcla consideran únicamente la posición nominal de los altavoces, por lo que una ligera modificación de las posiciones de los altavoces no afectará al comportamiento de la exclusión de zona.

7.3.12.1 Selección de altavoces excluidos

La selección de altavoces se efectúa procesando una lista de objetos `ExclusionZone` y generando una bandera booleana para cada altavoz, que tomará el valor verdadero si el altavoz está en cualquiera de las zonas de exclusión y, por tanto, debe excluirse.

Para los objetos `CartesianZone` se utiliza la siguiente expresión para determinar si un altavoz está dentro de la zona, siendo $\{x, y, z\}$ la posición nominal del altavoz, convertida a partir de coordenadas polares con un radio de 1:

$$\begin{aligned} &\mathrm{minX} - \epsilon < \quad x < \mathrm{maxX} + \epsilon \\ \wedge\, &\mathrm{minY} - \epsilon < \quad y < \mathrm{maxY} + \epsilon \\ \wedge\, &\mathrm{minZ} - \epsilon < \quad z < \mathrm{maxZ} + \epsilon \end{aligned}$$

donde $\epsilon = 10^{-6}$ es un margen de seguridad que permite errores de redondeo durante la conversión entre coordenadas polares y cartesianas.

Para los objetos `PolarZone` se utiliza la siguiente expresión para determinar si un altavoz está dentro de la zona, siendo φ y θ el acimut y la elevación nominales del altavoz.

$$\begin{aligned} &\mathrm{minElevation} - \epsilon < \theta < \mathrm{maxElevation} + \epsilon \\ \wedge\, &\begin{pmatrix} & |\theta| > 90 - \epsilon \\ \vee\, & \mathrm{IAR}(\varphi, \mathrm{minAzimuth}, \mathrm{maxAzimuth}, \epsilon) \end{pmatrix} \end{aligned}$$

IAR es la función `inside_angle_range`; véase el § 6.2.

La elevación del altavoz debe pertenecer a la gama permitida, mientras que el acimut sólo tiene que mantenerse dentro de la gama permitida si la elevación absoluta es inferior a 90 grados.

La implementación se efectúa en
`core.objectbased.gain_calc.ZoneExclusionHandler.get_excluded`.

7.3.12.2 Mezcla para altavoces excluidos

Una vez determinados los altavoces dentro de la zona, se diseña una matriz de mezcla para desviar las ganancias de esos altavoces.

El objeto panoramizador de exclusión de zona asocia a cada uno de los altavoces de la disposición una lista de grupos de altavoces de salida. La matriz de mezcla funciona de tal manera que la ganancia de un altavoz excluido se desvía a todos los demás altavoces no excluidos del primer grupo en que haya altavoces no excluidos. Esta funcionalidad se describe más detalladamente en las siguientes dos cláusulas.

Por ejemplo, en el Cuadro 3 se muestran los grupos para altavoces en 4+5+0. La primera fila muestra que, si se excluye `M+030`, la salida de este altavoz se desviará a `M+000`, a menos que esté excluido, en cuyo caso se desviará a `M-030`, etc. hasta llegar a `U-110`.

Un ejemplo más complicado en que la agrupación tiene efecto es `M+000`. De excluirse, este canal se dividirá entre los altavoces no excluidos en {M + 030, M − 030}, a menos que esos dos altavoces estén excluidos, en cuyo caso se desviará a los altavoces no excluidos en {M + 110, M − 110}, etc.

CUADRO 3

Ejemplo de asociación de altavoces para 4+5+0

Entrada	Grupos de salida
M + 030	{M + 030}, {M + 000}, {M − 030}, {M + 110}, {M − 110}, {U + 030}, {U − 030}, {U + 110}, {U − 110}
M − 030	{M − 030}, {M + 000}, {M + 030}, {M − 110}, {M + 110}, {U − 030}, {U + 030}, {U − 110}, {U + 110}
M + 000	{M + 000}, {M + 030 , M − 030}, {M + 110 , M − 110}, {U + 030 , U − 030}, {U + 110 , U − 110}
M + 110	{M + 110}, {M − 110}, {M + 030}, {M + 000}, {M − 030}, {U + 110}, {U − 110}, {U + 030}, {U − 030}
M − 110	{M − 110}, {M + 110}, {M − 030}, {M + 000}, {M + 030}, {U − 110}, {U + 110}, {U − 030}, {U + 030}
U + 030	{U + 030}, {U − 030}, {U + 110}, {U − 110}, {M + 030}, {M + 000}, {M − 030}, {M + 110}, {M − 110}
U − 030	{U − 030}, {U + 030}, {U − 110}, {U + 110}, {M − 030}, {M + 000}, {M + 030}, {M − 110}, {M + 110}
U + 110	{U + 110}, {U − 110}, {U + 030}, {U − 030}, {M + 110}, {M − 110}, {M + 030}, {M + 000}, {M − 030}
U − 110	{U − 110}, {U + 110}, {U − 030}, {U + 030}, {M − 110}, {M + 110}, {M − 030}, {M + 000}, {M + 030}

Esta funcionalidad se implementa en `core.objectbased.zone.ZoneExclusionDownmix` y `core.objectbased.gain_calc.ZoneExclusionHandler`.

7.3.12.2.1 Determinación de los grupos de altavoces

Durante la inicialización se determinan los grupos de altavoces de salida para cada altavoz.

Por cada altavoz de entrada se asigna a cada altavoz de salida una tupla de flotantes denominada *key*. Los grupos de salida están así conformados por los altavoces de salida organizados por claves y reunidos en grupos con claves similares. Por tanto, el ordenamiento y la agrupación se definen principalmente de acuerdo con la función clave.

La clave de un altavoz de entrada y salida está formada por cuatro claves:

– Una prioridad de capa entera, que es cero si ambos altavoces están en la misma capa y aumenta a medida que se separan las capas de entrada y salida, prefiriéndose seleccionar un altavoz de la capa superior antes que uno de una capa inferior. Las prioridades de capa se muestran en el Cuadro 4.

– Una prioridad frontal/posterior entera, que es más baja si los altavoces de entrada y salida están ambos al frente, al lado o detrás del oyente. Dada la componente y de la posición nominal polar de los altavoces de entrada y salida, tras su conversión a coordenadas cartesianas, y_i e y_o, el cálculo es el siguiente:

$$|\mathrm{sgn}y_i - \mathrm{sgn}y_o|$$

– El vector distancia entre las posiciones nominales de los dos altavoces, prefiriéndose los movimientos más cortos.

– La diferencia absoluta entre las coordenadas y nominales de ambos altavoces a fin de separar los grupos que no son simétricos en los planos yz o xz.

CUADRO 4

Valor de prioridad de capa entre dos altavoces

Capa de entrada	Inferior	Media	Alta	Superior
Inferior	0	1	2	3
Media	3	0	1	2
Alta	3	2	0	1
Superior	3	2	1	0

7.3.12.2.2 Aplicación de la exclusión de zona

La matriz de mezcla para un conjunto de altavoces excluidos, E, se calcula de la siguiente manera:

– Para N altavoces, empezar con una matriz de mezcla $N \times N$, **D**, con cada elemento inicializado a 0.

– Para cada altavoz de entrada, i, considerar cada grupo de índices de altavoz posibles, C, en la fila i del cuadro de grupos.

- Si todos los altavoces del grupo pertenecen al grupo de altavoces ignorados, es decir, $C \subseteq E$, pasar al siguiente grupo.
- En caso contrario, para cada j de $C \setminus E$ (conjunto de altavoces en un grupo no excluido):

$$D_{i,j} = \frac{1}{|C \setminus E|}$$

y pasar al siguiente altavoz.

Si todos los altavoces están excluidos, se pone **D** a la matriz identidad.

D se aplica entonces al vector ganancia entrante, **G**, para producir $\mathbf{G}'$ de la siguiente manera:

$$\mathbf{G}'_j = \sqrt{\sum_i \mathbf{G}_i^2 \, \mathbf{D}_{i,j}}$$

7.4 Filtros de descorrelación

Al reproducir objetos cuando el parámetro *diffuse* es superior a 0, se utiliza el trayecto difuso del reproductor de objetos, que tiene un filtro de descorrelación por salida de altavoz.

Los filtros utilizados son filtros FIR de todo paso y fase aleatoria de $N = 512$ muestras. El filtro para una salida determinada se genera como sigue:

– Se genera un vector pseudoaleatorio, **r**, con valores dentro de la gama [0,1) de longitud $\frac{N}{2} - 1$ utilizando el generador de números pseudoaleatorios MT19937, al que se añade el índice del nombre de canal escogido de una lista de todos los nombres de canal de la disposición.

– Se define un vector fase, **p**, de longitud $\frac{N}{2} + 1$:

$$\mathbf{p}_n = \begin{cases} 2\pi\mathbf{r}_{n-1} & 1 \le n \le \frac{N}{2} - 1 \\ 0 & \text{en caso contrario} \end{cases}$$

– Se define el correspondiente vector frecuencia, **x**, como $\mathbf{x}_n = \exp(i\mathbf{p}_n)$.
– Se toma una transformada de Fourier de valor real inversa (función `irfft`) de los componentes de frecuencia no negativos de **x** para obtener el filtro de dominio temporal.

La implementación se efectúa en
`core.objectbased.decorrelate.design_decorrelators`.

El retardo introducido por estos filtros se corresponde con un retardo de $\frac{(N-1)}{2}$ muestras en el trayecto directo.

8 Reproducción de elementos con typeDefinition==DirectSpeakers

Para reproducir un *audioChannelFormats* con *typeDefinition==DirectSpeakers*, este se dirige a un altavoz correspondiente. De no ser posible, se utilizará PSP como solución de repuesto.

El algoritmo básico es el siguiente:

1) Para las entradas especificadas con definiciones comunes *audioPackFormats* que describen disposiciones especificadas en la Recomendación UIT-R BS.2051-2, se aplican las reglas de correspondencia del § 8.1.
2) Se determina si los metadatos se refieren a un canal LFE (véase el § 8.2). De ser así, sólo se considerarán las salidas LFE; en caso contrario, sólo se considerarán las salidas distintas de LFE.
3) Si cualquiera de las *speakerLabels* corresponde a un altavoz (véase el § 8.3), el canal se dirige al primer altavoz que corresponda. Si ninguna *speakerLabel* corresponde, se va al paso siguiente.
4) Si `screenEdgeLock` está especificado, la posición nominal se desplazará al borde horizontal y/o vertical de la pantalla. No se modifican los límites mínimo y máximo (véase el § 8.4).
5) Si la posición nominal de cualquier altavoz está dentro de los límites de posición especificados (véase el § 8.5), el canal se dirige al altavoz más cercano a la posición nominal especificada. Las posiciones de altavoz utilizadas se determinan en función del tipo de posición, como en el § 8.5. si no hay altavoces dentro de los límites, o si no hay un único altavoz más cercano a la posición nominal, se va al paso siguiente.
6) Si los metadatos hacen referencia a un LFE, se dirige el canal a `LFE1` (de existir) o se descarta. Si los metadatos se refieren a un canal no LFE, se utiliza la PSP correspondiente al tipo de coordenadas utilizado para definir su posición a fin de reproducir el canal en su posición nominal.

En las siguientes subcláusulas se definen cada uno de los pasos más detalladamente.

La implementación se efectúa en
`core.direct_speakers.panner.DirectSpeakersPanner`.

8.1 Reglas de correspondencia

– Si el último *audioPackFormat* enumerado en `type_metadata.audioPackFormats` no es un formato de paquetes de definiciones comunes (es decir, se especifica en los metadatos de entrada, no se lee del fichero de definiciones comunes), no se aplican las reglas de correspondencia.

– Buscar el ID del último *audioPackFormat* enumerado en `type_metadata.audioPackFormats` en el Cuadro 15 para determinar `input_layout`. Si no está en la lista, no se aplican las reglas de correspondencia.

– Intentar aplicar cada una de las reglas enumeradas en el Cuadro 16 sucesivamente. De aplicarse alguna de ellas, se utiliza `gains` para la primera regla de correspondencia enumerada para reproducir este canal. Si ninguna regla corresponde, se va al paso siguiente. Una regla corresponde si se cumplen todas las condiciones siguientes:

  • `rule.speakerLabel` es igual al primer (y único) *speakerLabel* tras aplicar la normalización indicada en el § 8.3.

  • `input_layout` (determinado anteriormente) está enumerado en `rule.input_layouts`, si tiene forma de lista.

  • El nombre de la disposición de altavoces de salida, `layout.name`, está enumerado en `rule.output_layouts`, si tiene forma de lista.

  • Todos los nombres de canal enumerados en `rule.gains` existen en `layout.channel_names`.

8.2 Determinación de LFE

Se considera que un canal es un canal LFE si el elemento frecuencia de audioChannelFormat tiene un valor descrito en el § 6.3, o si hay un *speakerLabel* que hace referencia a un canal LFE (LFE1 o LFE2 tras aplicar el procedimiento de correspondencia descrito a continuación).

NOTA – La función IAR no aplica ningún desplazamiento de +10 dB (véase la Recomendación UIT-R BS.775) al canal LFE a efectos de la calibración de la reproducción, con respecto al canal principal, ya que esto se realiza en el dispositivo de reproducción.

8.3 Correspondencia de etiquetas de altavoz

La correspondencia de *speakerLabels* sólo funciona para las etiquetas de la Recomendación UIT-R BS.2051-2 (por ejemplo, `M+030`) y los URN utilizados en el fichero de definiciones comunes de ADM especificado en la Recomendación UIT-R BS.2094-1 (por ejemplo, `urn:itu:bs:2051:0:speaker:M+030`). Las etiquetas de LFE1 y LFE2 se especifican en la Recomendación UIT-R BS.2051-2. Cuando se utilizan las siguientes *speakerLabels* en el archivo ADM, se aplican algunas sustituciones:

– `LFE → LFE1`

– `LFEL → LFE1`

– `LFER → LFE2`

8.4 Fijación de bordes de pantalla

La implementación de *screenEdgeLock* para *typeDefinition==DirectSpeakers* reutiliza el `ScreenEdgeLockHandler` empleado para *typeDefinition==Objects*, que se describe detalladamente en el § 7.3.4. Se utiliza para transformar únicamente la posición nominal, dejando como están los límites mínimo y máximo.

Esto implica que, si se especifican límites y éstos se interpretan como límites absolutos, independientemente de la posición de la pantalla, la fuente sólo fijará un canal dentro de los límites originales especificados. Si no se especifican límites, se activará el panoramizador de fuente puntual, haciendo que la fuente se fije al borde de la pantalla, independientemente de si hay ahí un altavoz o no. Se recomienda no utilizar simultáneamente *screenEdgeLock* y límites de coordenadas.

8.5 Correspondencia de límites

La especificación de límites mínimos o máximos amplía la gama permisible más allá de la posición nominal. Si no se especifican límites mínimos o máximos, éstos se ponen a las coordenadas nominales. Un altavoz corresponde si todas sus coordenadas entran dentro del *bounds* especificado, con la excepción de los altavoces con coordenadas polares en los polos (por ejemplo, `T+000`), que se corresponden con cualquier gama de acimut, pues tienen un acimut indeterminado.

Un altavoz con posición polar nominal `speaker` se ajusta a los límites especificados en coordenadas polares si

$$\begin{aligned} &\left(\begin{array}{ll} & \text{IAR}(\texttt{speaker.azimuth}, \texttt{azimuth.min}, \texttt{azimuth.max}, \epsilon) \\ \vee & |\texttt{speaker.elevation}| \geq 90° - \epsilon \end{array}\right) \\ \wedge\ & \texttt{elevation.min} - \epsilon \leq \texttt{speaker.elevation} \leq \texttt{elevation.max} + \epsilon \\ \wedge\ & \texttt{distance.min} - \epsilon \leq \texttt{speaker.distance} \leq \texttt{distance.max} + \epsilon \end{aligned}$$

Donde IAR es la función `inside_angle_range` (véase el § 6.2) y $\epsilon = 10^{-5}$ es un margen de seguridad para permitir los errores de redondeo.

Un altavoz con posición cartesiana `speaker`, cuyas coordenadas se han convertido en cartesianas de acuerdo con el § 7.3.9, se ajusta a los límites especificados con coordenadas cartesianas si

$$\begin{aligned} & \texttt{X.min} - \epsilon \leq \texttt{speaker.X} \leq \texttt{X.max} + \epsilon \\ \wedge\ & \texttt{Y.min} - \epsilon \leq \texttt{speaker.Y} \leq \texttt{Y.max} + \epsilon \\ \wedge\ & \texttt{Z.min} - \epsilon \leq \texttt{speaker.Z} \leq \texttt{Z.max} + \epsilon \end{aligned}$$

es verdadero.

9 Reproducción de elementos con typeDefinition==HOA

9.1 Formatos HOA soportados

9.1.1 Orden y grado HOA

Las señales HOA, definidas en la Recomendación UIT-R BS.2076-1, pueden reproducirse hasta el orden 50 (véanse los detalles a continuación). En ADM, los canales HOA están señalados individualmente según su *order* y su *degree* gracias a los subelementos tipo HOA correspondientes. Por tanto, es posible reproducir escenas HOA 3D completo (compuestas por todos los orden l y grados m hasta un determinado orden L), las escenas HOA 2D (compuestas por todos los componentes HOA de manera que $|m| = l$ hasta un determinado orden L) y las escenas HOA de orden mixto.

Sin embargo, cuando dos señales HOA comparten el mismo grado *y* el mismo orden, surge una excepción y no se reproducen las señales.

9.1.2 Normalización

La normalización de la señal HOA se indica mediante el subelemento tipo HOA *normalization*. Este reproductor soporta las tres normalizaciones posibles (N3D, SN3D y FuMa). En ADM la normalización HOA se especifica para cada una de las señales HOA individualmente, por lo que es teóricamente posible definir escenas HOA en las que distintas señales utilicen distintas normalizaciones. Sin embargo, este reproductor no soporta ese comportamiento: todos los

canales HOA de un *audioBlockFormat* deben compartir la misma normalización. Por último, téngase en cuenta que la normalización FuMa sólo se soporta hasta el orden tres.

9.2 Subelementos no soportados

En la actualidad, no se interpretan en la reproducción los siguientes tres subelementos de tipo HOA:

– *nfcRefDist*, que indica una distancia de referencia para los altavoces. El efecto de compensación en el campo cercano (NFC, *near-field compensation*), que compensa la desadaptación entre la distancia de referencia del altavoz y la distancia a la que los altavoces están situados en la disposición de reproducción, no se implementa en este reproductor. La implementación de este efecto en la reproducción HOA aumenta considerablemente la complejidad de cálculo del reproductor, aunque su influencia en la percepción del contenido de audio por el oyente es relativamente escasa.

– *screenRef*, que indica si el componente HOA está relacionado con la pantalla. La utilización prevista de este subelemento es ambigua en el contexto HOA, por lo que no se tiene en cuenta en la reproducción.

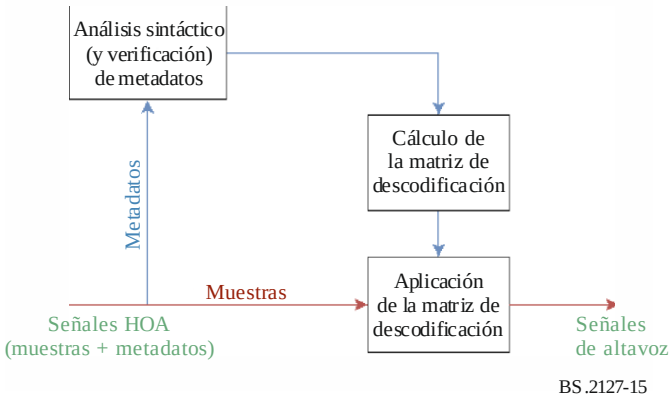
– *equation*, previsto para su utilización como sustituto de los subelementos *order* y *degree*. La actual norma ADM no contiene normas precisas sobre el formato utilizado para la especificación de fórmulas matemáticas. Por consiguiente, no es posible soportar fiablemente este subelemento.

Téngase en cuenta que, al igual que ocurre con el subelemento *normalization*, todos los canales HOA de un *audioBlockFormat* deben compartir idénticos valores *nfcRefDist* y *screenRef* para poder reproducirlos.

9.3 Reproducción de señales HOA por altavoces

FIGURA 15

Diagrama de flujo de reproducción de HOA



El proceso de reproducción de señales HOA por altavoces se resume en la Fig. 15. En primer lugar se analizan sintácticamente los metadatos ADM para identificar el formato del objeto HOA y verificar si es posible reproducir las señales sin ambigüedades. Concretamente, como ya se ha indicado, todos los canales HOA de un *audioBlockFormat* deben tener los mismos valores en los subelementos *normalization*, *nfcRefDist* y *screenRef*. A continuación se calcula una matriz de descodificación y se aplica esa matriz a las señales HOA, como se expresa en la siguiente ecuación:

$$\mathbf{S}_{\mathrm{spk}} = \mathbf{D}\,\mathbf{S}_{\mathrm{HOA}}$$

donde:

$\mathbf{S}_{\mathrm{spk}}$: es la matriz de señales de altavoz de dimensión $N_{\mathrm{spk}} \times N_{\mathrm{samp}}$

$\mathbf{S}_{\mathrm{HOA}}$: es la matriz de señales HOA de dimensión $N_{\mathrm{HOA}} \times N_{\mathrm{samp}}$

$\mathbf{D}$: es la matriz de valor real de dimensión $N_{\mathrm{spk}} \times N_{\mathrm{HOA}}$, denominada *matriz de descodificación HOA*

N_{HOA}, N_{spk} y N_{samp} denotan el número de señales HOA, las señales de altavoz y las muestras temporales, respectivamente.

En esta sección se muestra el cálculo de la matriz de descodificación para el ordenamiento de canales ACN, pero la atribución de canales utilizada es la especificada por los parámetros *order* y *degree* de *audioBlockFormat*.

La matriz de descodificación se aplica empleando la estructura de canal de procesamiento de bloques descrita en el § 6.4. Concretamente, para cada objeto `HOATypeMetadata` entrante se genera un único bloque de procesamiento `FixedMatrix`, que aplica la matriz de descodificación entre los tiempos determinados en el § 6.5.

9.3.1 Cálculo de la matriz de descodificación HOA

El reproductor implementa la técnica de descodificación HOA AllRAD [1]. Este método ofrece una descodificación HOA robusta en disposiciones de altavoces irregulares, como las indicadas en la Recomendación UIT-R BS.2051-2. El cálculo de la matriz de descodificación se efectúa en `core.scenebased.design.HOADecoderDesign`.

Conceptualmente, el método de descodificación AllRAD es equivalente a:

1) la descodificación de señales HOA a una cuadrícula de altavoces virtuales uniformemente distribuidos por la esfera, y
2) la panoramización de las señales de altavoces virtuales por altavoces reales.

La expresión matemática de este procedimiento es:

$$\mathbf{D}' = \nu\, \mathbf{G}\, \mathbf{D}_{\mathrm{virt}}$$

$$\mathbf{D} = \mathbf{D}' \mathrm{diag}(\mathbf{n}^{-1})$$

donde $\mathbf{D}'$ denota la matriz de descodificación HOA para la normalización *N3D,* $\mathbf{G}$ es la matriz de ganancia de panoramización, $\mathbf{D}_{\mathrm{virt}}$ es la matriz de descodificación de altavoz virtual y ν es un factor de normalización de energía. $\mathbf{D}$ es la matriz de descodificación completa tras la aplicación del vector de normalización HOA, $\mathbf{n}$, a $\mathbf{D}'$ para aplicar la normalización deseada.

9.3.1.1 Posición de altavoces virtuales

Para facilitar el cálculo de la matriz de descodificación es necesario distribuir las posiciones angulares de los altavoces virtuales lo más uniformemente posible por la esfera. Además, por norma general, debe haber el doble de posiciones de altavoces virtuales que de señales HOA.

En este reproductor las posiciones de altavoces virtuales constituyen un *diseño en T esférica* de 5 200 puntos, por lo que conviene para la descodificación de señales HOA hasta el orden 50.

9.3.1.2 Cálculo de la matriz de descodificación de altavoces virtuales

Para calcular la matriz de descodificación de altavoces virtuales, en primer lugar se calcula la matriz de coeficientes HOA para los altavoces virtuales, $\mathbf{Y}_{\mathrm{virt}}$. Esta matriz se obtiene con:

$$\mathbf{Y}_{\mathrm{virt}} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N_{\mathrm{virt}}}]$$

$$\mathbf{y}_n = [Y_0^0(\theta_n, \varphi_n), Y_1^{-1}(\theta_n, \varphi_n), \ldots]^{\mathrm{T}}$$

donde (θ_n, φ_n) denota los ángulos de elevación y acimut del n-ésimo altavoz virtual (utilizando el sistema de coordenadas HOA y la notación especificados en la Recomendación UIT-R BS.2076-1) e Y_l^m denota la función armónica esférica de orden l y grado m de valor real con normalización *N3D*. Se ha de tener en cuenta que el valor de cada término $Y_l^m(\theta, \varphi)$ depende de los subelementos *order* y *degree* de cada canal HOA.

A continuación se calcula la matriz de descodificación HOA de altavoces virtuales como una transposición de $\mathbf{Y}_{\mathrm{virt}}$:

$$\mathbf{D}_{\mathrm{virt}} = N_{\mathrm{samp}}^{-1} \mathbf{Y}_{\mathrm{virt}}^{\mathrm{T}}$$

Para la elección de las posiciones de altavoces virtuales y la normalización *N3D* esto equivale a tomar la pseudoinversa de $\mathbf{Y}_{\mathrm{virt}}$.

9.3.1.3 Cálculo de la matriz de ganancia de panoramización

Con el método de descodificación HOA AllRAD suele utilizarse la panoramización VBAP para calcular la matriz de ganancia de panoramización. En la implementación de este reproductor, el método utilizado para calcular las ganancias de panoramización es el previsto para la panoramización de objetos fuente puntual (`core.point_source`).

9.3.1.4 Normalización de energía

La matriz de descodificación HOA se normaliza de manera que, cuando la escena HOA consiste en una única fuente puntual, la potencia total de las señales de altavoz sea igual a la de la señal fuente, mediada para todas las posibles posiciones de la fuente en la esfera.

Matemáticamente, el factor de normalización, ν, se calcula de la siguiente manera:

$$\nu = \frac{\sqrt{N_{\mathrm{virt}}}}{\|\mathbf{G}\,\mathbf{D}_{\mathrm{virt}}\,\mathbf{Y}_{\mathrm{virt}}\|_{\mathrm{F}}}$$

siendo $\|\cdot\|_{\mathrm{F}}$ la norma de Frobenius.

9.3.1.5 Normalización HOA

La matriz de descodificación se divide por el vector $\mathbf{n}$ para convertir la señal en la normalización *N3D* para la que se ha diseñado $\mathbf{D}'$. $\mathbf{n}$ se define para un parámetro `norm` *normalization* como:

$$\mathbf{n}_n^m = \frac{N_{\mathrm{norm}_n}^{|m|}}{N_{\mathrm{N3D}_n}^{|m|}}$$

$$\mathbf{n} = [\mathbf{n}_0^0, \mathbf{n}_1^{-1}, \ldots]$$

10 Conversión de metadatos

En esta cláusula se define un método para la conversión entre parámetros polares y cartesianos en *audioBlockFormat*s con *typeDefinition==Objects*. Por definición la conversión de metadatos no puede ser exacta; los valores resultantes de la conversión no serán exactamente idénticos a los valores no convertidos. Por consiguiente, es necesario supervisar los resultados de la conversión. Téngase en cuenta que la conversión de extensión no es reversible, por lo que debe evitarse la conversión recurrente entre valores polares y cartesianos.

La interfaz de la funcionalidad de conversión es la siguiente:

```
AudioBlockFormat to_cartesian(AudioBlockFormat input);
AudioBlockFormat to_polar(AudioBlockFormat input);
```

Cuando se invoca `to_cartesian` con un `AudioBlockFormat input` en que se haya configurado `input.cartesian`, `input` se devuelve intacto. Por otra parte, cuando se invoca `to_polar` con un `AudioBlockFormat input` en que no se haya configurado `input.cartesian`, `input` se devuelve intacto.

En caso contrario, en ambos casos se invierte `input.cartesian` e `input` se modifica de la siguiente manera antes de devolverlo:

– `input.position` se convierte de acuerdo con el § 10.1.

– `input.width`, `input.height` e `input.depth` se convierten de acuerdo con el § 10.2.

– `input.objectDivergence` se convierte de acuerdo con el § 10.3.

La conversión se implementa en `core.objectbased.conversion`.

10.1 Conversión de *position*

Las posiciones se convierten de tal manera que la posición polar de un altavoz en la disposición `4+5+0` se corresponde con la coordenada cartesiana de ese altavoz utilizado en el panoramizador de fuente puntual cartesiano como se muestra en el Cuadro 8.

Hay que señalar que la misma conversión, basada en una configuración de canal 4+5+0, se utiliza independientemente de la disposición de canales del reproductor. Esto se hace para garantizar que los resultados de la conversión son siempre coherentes, incluso cuando la capa de reproducción del reproductor utilizada no se conoce durante la conversión. Se ha escogido la disposición 4+5+0 para garantizar una buena conversión del contenido creado con 0+5+0.

En esta cláusula se describen las definiciones comunes utilizadas para la conversión en ambos sentidos. Las funciones de conversión mismas se muestran en los § 10.1.1 y 10.1.2.

`map_linear_to_az` y `map_az_to_linear` definen una correspondencia reversible de las posiciones de la fuente entre coordenadas de acimutales (φ) y lineales (x) entre un par de altavoces cuyos acimutes son φ_l y $azimuth_r$, habida cuenta de las curvas de panoramización de los panoramizadores de fuente puntual utilizados para las coordenadas polares y cartesianas.

Por ejemplo, una posición polar φ_o entre 0° y −30° tiene una posición x determinada por:

$$x = \texttt{map\_az\_to\_linear}(0, -30, \varphi_o)$$

La correspondencia lineal a acimutal se define como:

$$\texttt{map\_linear\_to\_az}(\varphi_l, \varphi_r, x) = \varphi_{\texttt{mid}} + \varphi_{\texttt{rel}}$$

siendo:

$$\begin{aligned}
\varphi_{\mathtt{mid}} &= \tfrac{\varphi_l+\varphi_r}{2} \\
\varphi_{\mathtt{range}} &= \varphi_r - \varphi_{\mathtt{mid}} \\
g'_l &= \cos\tfrac{x\pi}{2} \\
g'_r &= \operatorname{sen}\tfrac{x\pi}{2} \\
g_r &= \tfrac{g'_r}{g'_l+g'_r} \\
\varphi_{\mathtt{rel}} &= \tfrac{180}{\pi}\arctan\left(2\left(g_r-\tfrac{1}{2}\right)\tan\left(\tfrac{\pi}{180}\varphi_{\mathtt{range}}\right)\right)
\end{aligned}$$

La función inversa se define como:

$$\mathtt{map\_az\_to\_linear}(\varphi_l,\varphi_r,\varphi) = \tfrac{2}{\pi}\operatorname{atan2}(g_r, 1-g_r)$$

siendo:

$$\begin{aligned}
\varphi_{\mathtt{mid}} &= \tfrac{\varphi_l+\varphi_r}{2} \\
\varphi_{\mathtt{range}} &= \varphi_r - \varphi_{\mathtt{mid}} \\
\varphi_{\mathtt{rel}} &= \varphi - \varphi_{\mathtt{mid}} \\
g_r &= \tfrac{1}{2} + \frac{\tan\left(\frac{\pi}{180}\varphi_{\mathtt{rel}}\right)}{2\tan\left(\frac{\pi}{180}\varphi_{\mathtt{range}}\right)}
\end{aligned}$$

Esta correspondencia se efectúa entre posiciones de altavoz de capa media, de acuerdo con las reglas siguientes, dando un acimut izquierdo y derecho y una posición x e y izquierda y derecha para un acimut de entrada dado:

$$find_{sector}(\varphi) \quad \begin{cases} \{30,0,\{-1,1\},\{0,1\}\} & IAR(\varphi,0,30) \\ \{0,-30,\{0,1\},\{1,1\}\} & IAR(\varphi,-30,0) \\ \{-30,-110,\{1,1\},\{1,-1\}\} & IAR(\varphi,-110,-30) \\ \{-110,110,\{1,-1\},\{-1,-1\}\} & IAR(\varphi,110,-110) \\ \{110,30,\{-1,-1\},\{-1,1\}\} & IAR(\varphi,30,110) \end{cases}$$

$$find_{\square}(\varphi) \quad \begin{cases} \{30,0,\{-1,1\},\{0,1\}\} & IAR(\varphi,0,45) \\ \{0,-30,\{0,1\},\{1,1\}\} & IAR(\varphi,-45,0) \\ \{-30,-110,\{1,1\},\{1,-1\}\} & IAR(\varphi,-135,-45) \\ \{-110,110,\{1,-1\},\{-1,-1\}\} & IAR(\varphi,135,-135) \\ \{110,30,\{-1,-1\},\{-1,1\}\} & IAR(\varphi,45,135) \end{cases}$$

donde IAR es la función `inside_angle_range` descrita en el § 6.2.

Los siguientes parámetros son comunes para los dos sentidos de conversión:

$$\begin{aligned}
\theta_{\mathtt{top}} &= 30 \\
\theta'_{\mathtt{top}} &= 45 \\
\epsilon &= 1\times10^{-10}
\end{aligned}$$

10.1.1 Polar a cartesiana

Para convertir una coordenada polar con acimut φ, elevación θ y distancia d en una coordenada cartesiana se utiliza la función

$$\texttt{point\_polar\_to\_cart}(\varphi, \theta, d) = x, y, z$$

donde si $|\theta| > \theta_{\texttt{top}}$, por lo que:

$$\begin{aligned} \theta' &= \theta'_{\texttt{top}} + (90 - \theta'_{\texttt{top}}) \frac{|\theta| - \theta_{\texttt{top}}}{90 - \theta_{\texttt{top}}} \\ z &= d\,\text{sgn}(\theta) \\ r_{xy} &= d\tan\left(\frac{\pi}{180}(90 - \theta')\right) \end{aligned}$$

En caso contrario:

$$\begin{aligned} \theta' &= \theta'_{\texttt{top}} \frac{\theta}{\theta_{\texttt{top}}} \\ z &= d\tan\left(\frac{\pi}{180}\theta'\right) \\ r_{xy} &= d \end{aligned}$$

Por último:

$$\begin{aligned} \{\varphi_l, \varphi_r, \{x_l, y_l\}, \{x_r, y_r\}\} &= \texttt{find\_sector}(\varphi) \\ \varphi' &= \texttt{relative\_angle}(\varphi_r, \varphi) \\ \varphi'_l &= \texttt{relative\_angle}(\varphi_r, \varphi_l) \\ p &= \texttt{map\_az\_to\_linear}(\varphi'_l, \varphi_r, \varphi') \\ x &= r_{xy}(x_l + p(x_r - x_l)) \\ y &= r_{xy}(y_l + p(y_r - y_l)) \end{aligned}$$

`relative_angle` se describe en el § 6.7.

10.1.2 Cartesiana a polar

Para convertir una posición cartesiana con coordenadas x, y y z en una coordenada polar, se utiliza la función

$$\texttt{point\_cart\_to\_polar}(x, y, z) = \varphi, \theta, d$$

donde si $|x| < \epsilon$ y $|y| < \epsilon$:

$$\{\varphi, \theta, d\} = \begin{cases} \{0,0,0\} & |z| < \epsilon \\ \{0,90\,\text{sgn}(z), |z|\} & \text{en caso contrario} \end{cases}$$

En caso contrario, se prosigue:

$$\begin{aligned}
\varphi' &= -\frac{180}{\pi}\operatorname{atan2}(x, y) \\
\{\varphi_l, \varphi_r, \{x_l, y_l\}, \{x_r, y_r\}\} &= \texttt{find_cart_sector}(\varphi') \\
[g_l \quad g_r] &= [x \quad y] \cdot \begin{bmatrix} x_l & y_l \\ x_r & y_r \end{bmatrix}^{-1} \\
r_{xy} &= g_l + g_r \\
\varphi'_l &= \texttt{relative_angle}(\varphi_r, \varphi_l) \\
\varphi_{\texttt{rel}} &= \texttt{map_linear_to_az}\left(\varphi'_l, \varphi_r, \frac{g_r}{r_{xy}}\right) \\
\varphi &= \texttt{relative_angle}(-180, \varphi_{\texttt{rel}}) \\
\theta' &= \frac{180}{\pi}\arctan\frac{z}{r_{xy}}
\end{aligned}$$

si$|\theta'| > \theta'_{\texttt{top}}$:

$$\begin{aligned}
|\theta| &= \theta_{\texttt{top}} + (90 - \theta_{\texttt{top}})\frac{|\theta'| - \theta'_{\texttt{top}}}{90 - \theta'_{\texttt{top}}} \\
\theta &= |\theta|\operatorname{sgn}\theta' \\
d &= |z|
\end{aligned}$$

En caso contrario:

$$\begin{aligned}
\theta &= \theta'\frac{\theta_{\texttt{top}}}{\theta'_{\texttt{top}}} \\
d &= r_{xy}
\end{aligned}$$

`local_coordinate_system` se define en el § 6.8.

10.2 Conversión de extensión

La conversión de parámetros de extensión se efectúa en dos partes:

– $whd2xyz$ y $xyz2whd$: funciones que convierten los parámetros de extensión de cartesiano a polar, suponiendo una posición de fuente directamente en frente del oyente con un radio 1.

– `point_polar_to_cart` y `point_cart_to_polar`: funciones que se ocupan de la conversión de posición y de extensión. Las posiciones se convierten utilizando los métodos descritos en el § 10.1. La conversión de extensión utiliza $whd2xyz$ y $xyz2whd$, rotando la extensión cartesiana para coincidir con la posición.

Téngase en cuenta que, por norma general, la conversión de extensión no es reversible.

10.2.1 Polar a cartesiana

`extent_polar_to_cart` toma una posición de fuente polar en forma de acimut, elevación y distancia, con anchura, altura y profundidad polares, y devuelve coordenadas x, y y z cartesianas y dimensiones x, y y z cartesianas:

$$\texttt{extent\_polar\_to\_cart}(\varphi, \theta, d, \texttt{width}, \texttt{height}, \texttt{depth}) = \{x, y, z, s_x, s_y, s_z\}$$

siendo:

$$\begin{aligned}
\{x, y, z\} &= \texttt{point_polar_to_cart}(\varphi, \theta, d) \\
\{s_{x,f}, s_{y,f}, s_{z,f}\} &= \texttt{whd2xyz}(\texttt{width}, \texttt{height}, \texttt{depth}) \\
[\mathbf{M}_x \quad \mathbf{M}_y \quad \mathbf{M}_z] &= \text{diag}([s_{x,f}, s_{y,f}, s_{z,f}]) \cdot \texttt{local_coordinate_system}(\varphi, \theta) \\
s_x &= \| \mathbf{M}_x \|_2 \\
s_y &= \| \mathbf{M}_y \|_2 \\
s_z &= \| \mathbf{M}_z \|_2
\end{aligned}$$

y

$$whd2xyz(width, height, depth) = \{s_{x,w}, max(s_{y,w}, s_{y,h}, s_{y,d}), s_{z,h}\}$$

siendo:

$$\begin{aligned}
s_{x,w} &= \begin{cases} \text{sen}\frac{\pi}{180}\frac{\texttt{width}}{2} & \texttt{width} < 180 \\ 1 & \text{en caso contrario} \end{cases} \\
s_{y,w} &= \frac{1-\cos\frac{\pi}{180}\frac{\texttt{width}}{2}}{2} \\
s_{z,h} &= \begin{cases} \text{sen}\frac{\pi}{180}\frac{\texttt{height}}{2} & \texttt{height} < 180 \\ 1 & \text{en caso contrario} \end{cases} \\
s_{y,h} &= \frac{1-\cos\frac{\pi}{180}\frac{\texttt{height}}{2}}{2} \\
s_{y,d} &= \texttt{depth}
\end{aligned}$$

10.2.2 Cartesiana a polar

`extent_cart_to_polar` toma una posición de fuente cartesiana en forma de coordenadas x, y y z y una extensión cartesiana de dimensiones x, y y z, y devuelve una posición y una extensión polares con acimut, elevación y distancia y anchura, altura y profundidad:

$$\texttt{extent\_cart\_to\_polar}(x, y, z, s_x, s_y, s_z) = \{\varphi, \theta, d, \texttt{width}, \texttt{height}, \texttt{depth}\}$$

siendo:

$$\begin{aligned}
\{\varphi, \theta, d\} &= \texttt{point_cart_to_polar}(x, y, z) \\
[\mathbf{M}_x \quad \mathbf{M}_y \quad \mathbf{M}_z] &= \text{diag}([s_x, s_y, s_z]) \cdot \texttt{local_coordinate_system}(\varphi, \theta)^T \\
s_{x,f} &= \| \mathbf{M}_x \|_2 \\
s_{y,f} &= \| \mathbf{M}_y \|_2 \\
s_{z,f} &= \| \mathbf{M}_z \|_2 \\
\{\texttt{width}, \texttt{height}, \texttt{depth}\} &= \texttt{xyz2whd}(s_{x,f}, s_{y,f}, s_{z,f})
\end{aligned}$$

y

$$xyz2whd(s_x, s_y, s_z) = \{w, h, d\}$$

siendo:

$$\begin{aligned} w_{sx} &= 2\frac{180}{\pi}\text{arcsens}s_x \\ w_{sy} &= 2\frac{180}{\pi}\arccos(1-2s_y) \\ w &= w_{sx} + s_x\max(w_{sy}-w_{sx},0) \\ h_{sz} &= 2\frac{180}{\pi}\text{arcsens}s_z \\ h_{sy} &= 2\frac{180}{\pi}\arccos(1-2s_y) \\ h &= h_{sz} + s_z\max(h_{sy}-h_{sz},0) \\ \{s_{x,\text{eq}}, s_{y,\text{eq}}, s_{z,\text{eq}}\} &= \text{whd2xyz}(w,h,0) \\ d &= \max(0, s_y - s_{y,\text{eq}}) \end{aligned}$$

10.3 Conversión objectDivergence

`azimuthRange` y `positionRange` se convierten de acuerdo con la siguiente relación:

$$positionRange = tan\frac{270\times azimuthRange}{\pi}$$

11 Estructuras de datos y cuadros

11.1 Estructuras de metadatos internas

11.1.1 Estructuras compartidas

```
struct Position { };

struct PolarPosition : Position {
  float azimuth, elevation, distance = 1;
};

struct CartesianPosition : Position {
  float x, y, z;
};

struct Screen { };

struct PolarScreen : Screen {
  float aspectRatio;
  PolarPosition centrePosition;
  float widthAzimuth;
};

struct CartesianScreen : Screen {
  float aspectRatio;
  CartesianPosition centrePosition;
  float widthX;
};

struct Frequency {
  optional<float> lowPass;
  optional<float> highPass;
};
```

```
struct ExtraData {
  Fraction object_start;
  Fraction object_duration;
  Screen reference_screen;
  Frequency channel_frequency;
};
```

11.1.2 Metadatos de entrada

```
struct ChannelLock {
  optional<float> maxDistance;
};

struct ObjectDivergence {
  float value;
  optional<float> azimuthRange;
  optional<float> positionRange;
};

struct JumpPosition {
  bool flag;
  optional<float> interpolationLength;
};

struct ExclusionZone { };

struct CartesianZone : ExclusionZone {
  float minX;
  float minY;
  float minZ;
  float maxX;
  float maxY;
  float maxZ;
};

struct PolarZone : ExclusionZone {
  float minElevation;
  float maxElevation;
  float minAzimuth;
  float maxAzimuth;
};

struct ScreenEdgeLock {
    enum Horizontal { LEFT; RIGHT; };
    enum Vertical { BOTTOM; TOP; };

    optional<Horizontal> horizontal;
    optional<Vertical> vertical;
};

struct ObjectPosition { };

class PolarObjectPosition : ObjectPosition {
    float azimuth, elevation, distance;
    ScreenEdgeLock screenEdgeLock;
};

class CartesianObjectPosition | ObjectPosition {
    float X, Y, Z;
    ScreenEdgeLock;
};
```

```
struct AudioBlockFormatObjects {
  ObjectPosition position;
  bool cartesian;
  float width, height, depth;
  float diffuse;
  optional<ChannelLock> channelLock;
  optional<ObjectDivergence> objectDivergence;
  optional<JumpPosition> jumpPosition;
  bool screenRef;
  int importance;
  vector<ExclusionZone> zoneExclusion;
};

struct ObjectTypeMetadata {
  AudioBlockFormatObjects block_format;
  ExtraData extra_data;
};
```

11.1.3 Datos del entorno de reproducción

```
struct Channel {
  string name;
  /// The real position of the loudspeaker
  PolarPosition polar_position;
  /// The nominal position of the loudspeaker as in bs.2051-2.
  PolarPosition polar_nominal_position;
  bool is_lfe;
};

struct Layout {
  /// the ITU-format layout name, e.g. "9+10+3"
  string name;
  vector<Channel> channels;
  Screen screen;
};
```

11.2 Posiciones de altavoces alocéntricas

Estos datos, que se incluyen aquí como referencia, pueden consultarse en formato de lectura automática en `iar/core/data/allo_positions.yaml`.

CUADRO 5

Posiciones de altavoz alocéntricas para 0+2+0

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0

CUADRO 6

Posiciones de altavoz alocéntricas para 0+5+0

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+110	−1	−1	0
M-110	1	−1	0
LFE1	−1	1	−1

CUADRO 7

Posiciones de altavoz alocéntricas para 2+5+0

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+110	−1	−1	0
M-110	1	−1	0
U+030	−1	1	1
U-030	1	1	1
LFE1	−1	1	−1

CUADRO 8

Posiciones de altavoz alocéntricas para 4+5+0

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+110	−1	−1	0
M-110	1	−1	0
U+030	−1	1	1
U-030	1	1	1
U+110	−1	−1	1
U-110	1	−1	1
LFE1	−1	1	−1

CUADRO 9

Posiciones de altavoz alocéntricas para 4+5+1

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+110	−1	−1	0
M-110	1	−1	0
U+030	−1	1	1
U-030	1	1	1
U+110	−1	−1	1
U-110	1	−1	1
B+000	0	1	−1
LFE1	−1	1	−1

CUADRO 10

Posiciones de altavoz alocéntricas para 3+7+0

Canal	X	Y	Z
M+000	0	1	0
M+030	−1	1	0
M-030	1	1	0
U+045	−1	1	1
U-045	1	1	1
M+090	−1	0	0
M-090	1	0	0
M+135	−1	−1	0
M-135	1	−1	0
UH+180	0	−1	1
LFE1	−1	1	−1
LFE2	1	1	−1

CUADRO 11

Posiciones de altavoz alocéntricas para 4+9+0

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+090	−1	0	0
M-090	1	0	0
M+135	−1	−1	0
M-135	1	−1	0
U+045	−1	1	1
U-045	1	1	1
U+135	−1	−1	1
U-135	1	−1	1
LFE1	−1	1	−1

CUADRO 12

Posiciones de altavoz alocéntricas para 9+10+3

Canal	X	Y	Z
M+060	−1	0,414214	0
M-060	1	0,414214	0
M+000	0	1	0
M+135	−1	−1	0
M-135	1	−1	0
M+030	−1	1	0
M-030	1	1	0
M+180	0	−1	0
M+090	−1	0	0
M-090	1	0	0
U+045	−1	1	1
U-045	1	1	1
U+000	0	1	1
T+000	0	0	1
U+135	−1	−1	1
U-135	1	−1	1
U+090	−1	0	1
U-090	1	0	1
U+180	0	−1	1
B+000	0	1	−1

CUADRO 12 (*fin*)

Canal	X	Y	Z
B+045	−1	1	−1
B-045	1	1	−1
LFE1	−1	1	−1
LFE2	1	1	−1

CUADRO 13

Posiciones de altavoz alocéntricas para 0+7+0

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+090	−1	0	0
M-090	1	0	0
M+135	−1	−1	0
M-135	1	−1	0
LFE1	−1	1	−1

CUADRO 14

Posiciones de altavoz alocéntricas para 4+7+0

Canal	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+090	−1	0	0
M-090	1	0	0
M+135	−1	−1	0
M-135	1	−1	0
U+045	−1	1	1
U-045	1	1	1
U+135	−1	−1	1
U-135	1	−1	1
LFE1	−1	1	−1

11.3 Correspondencia de datos para DirectSpeakers

Estos datos, que se incluyen aquí como referencia, pueden encontrarse en formato de lectura automática en `core.direct_speakers.panner.itu_packs` y `core.direct_speakers.panner.rules`.

CUADRO 15

Correspondencia entre definiciones comunes *audioPackFormatID* y disposiciones (véase el § 8.1)

audioPackFormatID	`input_layout`
AP_00010001	0+1+0
AP_00010002	0+2+0
AP_00010003	0+5+0
AP_00010004	2+5+0
AP_00010005	4+5+0
AP_00010007	3+7+0
AP_00010008	4+9+0
AP_00010009	9+10+3
AP_0001000c	0+5+0
AP_0001000f	0+7+0
AP_00010010	4+5+1
AP_00010017	4+7+0

CUADRO 16

Reglas de correspondencia para DirectSpeakers (véase el § 8.1)

***speakerLabel* de entrada**	**Ganancias de reproducción de salida**	`input_layouts`	`output_layouts`
M+000	M+000 = 1		
M+000	M+030 = M-030 = $\sqrt{\frac{1}{2}}$		
M+060	M+060 = 1		
M-060	M-060 = 1		
M+060	M+110 = $\sqrt{\frac{1}{3}}$, M+030 = $\sqrt{\frac{2}{3}}$		
M-060	M-110 = $\sqrt{\frac{1}{3}}$, M-030 = $\sqrt{\frac{2}{3}}$		
M+060	M+030 = M+090 = $\sqrt{\frac{1}{2}}$		
M-060	M-030 = M-090 = $\sqrt{\frac{1}{2}}$		
M+060	M+030 = 1		
M-060	M-030 = 1		

CUADRO 16 (*continuación*)

***speakerLabel* de entrada**	**Ganancias de reproducción de salida**	`input_layouts`	`output_layouts`
M+090	M+090 = 1		
M-090	M-090 = 1		
M+090	M+030 = $\sqrt{\frac{1}{3}}$, M+110 = $\sqrt{\frac{2}{3}}$	9+10+3	
M-090	M-030 = $\sqrt{\frac{1}{3}}$, M-110 = $\sqrt{\frac{2}{3}}$	9+10+3	
M+090	M+030 = M+110 = $\sqrt{\frac{1}{2}}$		
M-090	M-030 = M-110 = $\sqrt{\frac{1}{2}}$		
M+090	M+030 = $\sqrt{\frac{1}{2}}$		
M-090	M-030 = $\sqrt{\frac{1}{2}}$		
M+110	M+110 = 1		
M-110	M-110 = 1		
M+110	M+135 = 1		
M-110	M-135 = 1		
M+110	M+030 = $\sqrt{\frac{1}{2}}$		
M-110	M-030 = $\sqrt{\frac{1}{2}}$		
M+135	M+135 = 1		
M-135	M-135 = 1		
M+135	M+110 = 1		
M-135	M-110 = 1		
M+135	M+030 = $\sqrt{\frac{1}{2}}$		
M-135	M-030 = $\sqrt{\frac{1}{2}}$		
M+180	M+180 = 1		
M+180	M+135 = M-135 = $\sqrt{\frac{1}{2}}$		
M+180	M+110 = M-110 = $\sqrt{\frac{1}{2}}$		
M+180	M+030 = M-030 = $\sqrt{\frac{1}{4}}$		
U+000	U+000 = 1		
U+000	U+030 = U-030 = $\sqrt{\frac{1}{2}}$		

CUADRO 16 (*continuación*)

***speakerLabel* de entrada**	**Ganancias de reproducción de salida**	`input_layouts`	`output_layouts`
U+000	U+045 = U-045 = $\sqrt{\frac{1}{2}}$		
U+000	M+000 = 1		
U+000	M+030 = M-030 = $\sqrt{\frac{1}{2}}$		
U+030	U+030 = 1		
U-030	U-030 = 1		
U+030	U+045 = 1		
U-030	U-045 = 1		
U+030	M+030 = 1		
U-030	M-030 = 1		
U+045	U+045 = 1		
U-045	U-045 = 1		
U+045	U+030 = 1		
U-045	U-030 = 1		
U+045	M+030 = 1		
U-045	M-030 = 1		
U+090	U+090 = 1		
U-090	U-090 = 1		
U+090	UH+180 = $\sqrt{\frac{1}{3}}$, U+045 = $\sqrt{\frac{2}{3}}$	9+10+3	
U-090	UH+180 = $\sqrt{\frac{1}{3}}$, U-045 = $\sqrt{\frac{2}{3}}$	9+10+3	
U+090	U+030 = U+110 = $\sqrt{\frac{1}{2}}$		
U-090	U-030 = U-110 = $\sqrt{\frac{1}{2}}$		
U+090	U+045 = U+135 = $\sqrt{\frac{1}{2}}$		
U-090	U-045 = U-135 = $\sqrt{\frac{1}{2}}$		
U+090	M+090 = 1		
U-090	M-090 = 1		
U+090	U+030 = M+110 = $\sqrt{\frac{1}{2}}$		
U-090	U-030 = M-110 = $\sqrt{\frac{1}{2}}$		
U+090	M+030 = M+110 = $\sqrt{\frac{1}{2}}$		

CUADRO 16 (*continuación*)

***speakerLabel* de entrada**	**Ganancias de reproducción de salida**	`input_layouts`	`output_layouts`
U-090	M-030 = M-110 = $\sqrt{\frac{1}{2}}$		
U+090	M+030 = $\sqrt{\frac{1}{2}}$		
U-090	M-030 = $\sqrt{\frac{1}{2}}$		
U+110	U+110 = 1		
U-110	U-110 = 1		
U+110	U+135 = 1		
U-110	U-135 = 1		
U+110	U+045 = UH+180 = $\sqrt{\frac{1}{2}}$		
U-110	U-045 = UH+180 = $\sqrt{\frac{1}{2}}$		
U+110	M+110 = 1		
U-110	M-110 = 1		
U+110	M+135 = 1		
U-110	M-135 = 1		
U+110	M+030 = $\sqrt{\frac{1}{2}}$		
U-110	M-030 = $\sqrt{\frac{1}{2}}$		
U+135	U+135 = 1		
U-135	U-135 = 1		
U+135	U+110 = 1		
U-135	U-110 = 1		
U+135	U+045 = $\sqrt{\frac{1}{3}}$, UH+180 = $\sqrt{\frac{2}{3}}$	9+10+3	
U-135	U-045 = $\sqrt{\frac{1}{3}}$, UH+180 = $\sqrt{\frac{2}{3}}$	9+10+3	
U+135	U+045 = UH+180 = $\sqrt{\frac{1}{2}}$		
U-135	U-045 = UH+180 = $\sqrt{\frac{1}{2}}$		
U+135	M+135 = 1		
U-135	M-135 = 1		
U+135	M+110 = 1		
U-135	M-110 = 1		
U+135	M+030 = $\sqrt{\frac{1}{2}}$		

CUADRO 16 (*continuación*)

***speakerLabel* de entrada**	**Ganancias de reproducción de salida**	`input_layouts`	`output_layouts`
U-135	M-030 = $\sqrt{\frac{1}{2}}$		
U+180	U+180 = 1		
U+180	UH+180 = 1		
U+180	U+135 = U-135 = $\sqrt{\frac{1}{2}}$		
U+180	U+110 = U-110 = $\sqrt{\frac{1}{2}}$		
U+180	M+135 = M-135 = $\sqrt{\frac{1}{2}}$		
U+180	M+110 = M-110 = $\sqrt{\frac{1}{2}}$		
U+180	M+030 = M-030 = $\sqrt{\frac{1}{4}}$		
UH+180	UH+180 = 1		
UH+180	U+180 = 1		
UH+180	U+135 = U-135 = $\sqrt{\frac{1}{2}}$		
UH+180	U+110 = U-110 = $\sqrt{\frac{1}{2}}$		
UH+180	M+135 = M-135 = $\sqrt{\frac{1}{2}}$		
UH+180	M+110 = M-110 = $\sqrt{\frac{1}{2}}$		
UH+180	M+030 = M-030 = $\sqrt{\frac{1}{4}}$		
T+000	T+000 = 1		
T+000	U+045 = U-045 = U+135 = U-135 = $\sqrt{\frac{1}{4}}$		
T+000	U+030 = U-030 = U+110 = U-110 = $\sqrt{\frac{1}{4}}$		
T+000	U+045 = U-045 = UH+180 = $\sqrt{\frac{1}{3}}$		
T+000	U+045 = U-045 = M+135 = M-135 = $\sqrt{\frac{1}{4}}$		
T+000	U+030 = U-030 = M+110 = M-110 = $\sqrt{\frac{1}{4}}$		
T+000	M+030 = M-030 = M+135 = M-135 = $\sqrt{\frac{1}{4}}$		

CUADRO 16 (*fin*)

***speakerLabel* de entrada**	**Ganancias de reproducción de salida**	`input_layouts`	`output_layouts`
T+000	M+030 = M-030 = M+110 = M-110 = $\sqrt{\frac{1}{4}}$		
T+000	M+030 = M-030 = $\sqrt{\frac{1}{4}}$		
B+000	B+000 = 1		
B+000	M+000 = 1		
B+000	M+030 = M-030 = $\sqrt{\frac{1}{2}}$		
B+045	B+045 = 1		
B-045	B-045 = 1		
B+045	M+030 = 1		
B-045	M-030 = 1		
LFE1	LFE1 = 1	9+10+3, 3+7+0	9+10+3, 3+7+0
LFE2	LFE2 = 1	9+10+3, 3+7+0	9+10+3, 3+7+0
LFE1	LFE1 = $\sqrt{\frac{1}{2}}$	9+10+3, 3+7+0	
LFE2	LFE1 = $\sqrt{\frac{1}{2}}$	9+10+3, 3+7+0	
LFE1	LFE1 = 1		

Bibliografía

[1] F. Zotter y M. Frank (2012), *All-round ambisonic panning and decoding, Journal of the audio engineering society*, vol. 60, no. 10, pp. 807-820.

[2] V. Pulkki, (1997), *Virtual sound source positioning using vector base amplitude panning, Journal of the audio engineering society*, vol. 45, no. 6, pp. 456-466.

Adjunto 1
del Anexo 1
(informativo)

Tabla de referencia a las partes correspondientes de la especificación de metadatos ADM

A1.1 Metadatos ADM en el reproductor ADM del UIT-R

En el Cuadro siguiente se resumen los elementos clave del reproductor y las cláusulas del Anexo 1 en que se tratan. Las especificaciones han de obtenerse de las referencias enumeradas.

Metadatos ADM ***sub-element (attribute)[coordinate system]***	**Recomendación UIT-R BS.2076-1**	**Anexo 1 a esta Recomendación**
typeDefinition == "DirectSpeakers"	§ 5.4.3.1 Cuadro 11	§ 8
speakerLabel		§ 8.2
position (azimuth, elevation, distance, screenEdgeLock)		§ 8
typeDefinition == "Matrix"	§ 5.4.3.2	§ 5.2.6.1.1 § 5.2.6.4
outputChannelIDRef	Cuadro 12	§ 5.2.6.1.1
matrix → coefficient (gain, gainVar, phase, phaseVar, delay, delayVar)	Cuadro 13	§ 5.6.4
input / outputPackFormatIDRef	§ 5.5.5.1	§ 5.2.6.1.1
encode / decodePackFormatIDRef		§ 5.2.6.1.1
typeDefinition == "Objects"	§ 5.4.3.3	§ 7
position (azimuth, elevation, distance, screenEdgeLock) [*polar*]	Cuadro 14	§ 6.1 § 7 § 7.3.4
position (X, Y, Z, screenEdgeLock) [*cartesian*]	Cuadro 15	§ 6.1 § 7 § 7.3.10
width, height, depth [*polar*]	Cuadro 14	§ 7.3.8
width, height, depth [*cartesian*]	Cuadro 15	§ 7.3.11
cartesian	Cuadro 16	§ 7.3.1 § 7.3.2
gain		§ 7.3.1
diffuse		§ 7.3.1 § 7.4
channelLock (maxDistance)		§ 7.3.6
objectDivergence (azimuthRange, positionRange) [*polar*]		§ 7.3.7 § 7.3.1
objectDivergence (azimuthRange, positionRange) [*cartesian*]		§ 7.3.7 § 7.3.1
jumpPosition (interpolationLength)		§ 7.2

Metadatos ADM *sub-element (attribute)[coordinate system]*	Recomendación UIT-R BS.2076-1	Anexo 1 a esta Recomendación
zoneExclusion → zone (minX, maxX, minY, maxY, minZ, maxZ, minElevation, maxElevation, minAzimuth, maxAzimuth)		§ 7.3.5 § 7.3.12
screenRef		§ 7.3.3
importance		§ 5.3.1 § 5.2.7.1.1
typeDefinition == "HOA"	§ 5.4.3.4	§ 9 § 5.2.7.3
equation	Cuadro 17	§ 9.2
order		§ 9.1.1 § 9.3.1.2
degree		§ 9.1.1 § 9.3.1.2
normalization	§ 5.4.3.4	§ 9.1.2 § 9.3.1.5
nfcRefDist	Cuadro 17	§ 9.2
screenRef		§ 9.2
typeDefinition == "Binaural"	§ 5.4.3.5	–

Adjunto 2
del Anexo 1
(informativo)

Configuración de altavoces virtuales alternativa

A2.1 Especificación de una configuración de altavoces virtuales alternativa

La configuración de altavoces virtuales VBAP alternativa a la especificada en el § 6.1.3.1 describe las posiciones de los altavoces virtuales no situados en los polos y sus coeficientes de repliegue. Los metadatos ADM reciben el mismo tratamiento especificado en el cuerpo de esta Recomendación y no se necesitan metadatos adicionales. Las posiciones de altavoces virtuales alternativas y sus coeficientes de repliegue se basan en optimizaciones acústicas. A continuación se describe la configuración de altavoces virtuales alternativa.

A2.1.1 Proceso de configuración

Para la configuración se siguen los pasos descritos en el § 6.1.3.1 a excepción del paso 2, que debe ser el siguiente:

2) En primer lugar se determinan los altavoces virtuales de acuerdo con los Cuadros del § A2.1.2. En cada subcláusula del § A2.1.2 se define una configuración de altavoces virtuales y sus coeficientes de repliegue para las disposiciones definidas en la Recomendación UIT-R BS.2051-2.

Los demás pasos de la configuración, a saber, el paso (1) y los pasos (3) a (6) son los descritos en el § 6.1.3.1.

A2.1.2 Cuadros de altavoces virtuales y repliegue

En los siguientes Cuadros, los altavoces virtuales (especificados mediante acimut y elevación) se muestran en la primera fila, mientras que los altavoces físicos ocupan la primera columna. Las posiciones nominales y reales de los altavoces virtuales son idénticas. En el Cuadro se muestran los coeficientes de repliegue entre altavoces virtuales y altavoces físicos.

Sistema A: 0+2+0

Para el sistema A:0+2+0 se emplea un método basado en una mezcla del sistema B:0+5+0 al sistema A:0+2+0, como se describe en el § 6.1.2.4. para obtener los canales 0+5+0, se emplean, como se indica a continuación, los altavoces virtuales para el sistema B:0+5+0.

Sistema B: 0+5+0

	−45, 45	45, 45	−135, 45	135, 45	−45, −45	45, −45	−135, −45	135, −45
M+030		1,0				1,0		
M-030	1,0				1,0			
M+000								
LFE1								
M+110			0,3162	0,9486			0,3162	0,9486
M-110			0,9486	0,3162			0,9486	0,3162

Sistema C: 2+5+0

	−135, 30	135, 30	−45, −45	45, −45	−135, −45	135, −45
M+030				1,0		
M-030			1,0			
M+000						
LFE1						
M+110	0,3162	0,9486			0,3162	0,9486
M-110	0,9486	0,3162			0,9486	0,3162
U+030						
U-030						

Sistema D: 4+5+0

	−45, −45	45, −45	−110, −45	110, −45
M+030		1,0		
M-030	1,0			
M+000				
LFE1				
M+110			0,3162	0,9486
M-110			0,9486	0,3162
U+030				
U-030				
U+110				
U-110				

Sistema E: 4+5+1

Esta disposición tiene altavoces superiores e inferiores. No se necesitan altavoces virtuales, pues la cubierta está completa.

Sistema F: 3+7+0

	−135, 30	135, 30	−45, -45	45, −45	−135, −45	135, −45
M+000						
M+030				1,0		
M-030			1,0			
U+045						
U-045						
M+090						
M-090						
M+135		0,7071				1,0
M-135	0,7071				1,0	
UH+180	0,7071	0,7071				
LFE1						
LFE2						

Sistema G: 4+9+0

	−45, −45	45, −45	−135, −45	135, −45
M+030		1,0		
M-030	1,0			
M+000				
LFE1				
M+090				
M-090				
M+135				1,0
M-135			1,0	
U+045				
U-045				
U+135				
U-135				
M+SC				
M-SC				

Sistema H: 9+10+3

Contiene altavoces en los hemisferios superior e inferior. Al estar completa la cubierta, no se necesitan altavoces virtuales.

Sistema I: 0+7+0

	−45, 45	45, 45	−135, 45	135, 45	−45, −45	45, −45	−135, −45	135, −45
M+030		1,0						
M-030	1,0				1,0	1,0		
M+000								
LFE1								
M+090								
M-090								
M+135				1,0				1,0
M-135			1,0				1,0	

Sistema J: 4+7+0

	−45, −45	45, −45	−135, −45	135, −45
M+030		1,0		
M-030	1,0			
M+000				
LFE1				
M+090				
M-090				
M+135				1,0
M-135			1,0	
U+045				
U-045				
U+135				
U-135				