国　　际　　电　　信　　联　　盟

ITU-R

国际电联无线电通信部门

ITU-R BS.2127-0 建议书

（06/2019）

高级音响系统的音频定义模型渲染器

BS 系列

广播业务（声音）

国际电信联盟

前言

无线电通信部门的作用是确保包括卫星服务在内的所有无线电通信服务合理、公平、有效和经济地使用无线电频谱，并在不受频率范围限制的情况下进行研究，在此基础上通过建议。

无线电通信部门的管理和政策职能由研究组支持的世界和区域无线电通信大会和无线电通信全会履行。

知识产权政策 (IPR)

ITU-R的知识产权政策在ITU-T/ITU-R/ISO/IEC的通用专利政策中有详细的描述。专利权人提交专利声明和许可声明所使用的表格可从http://www.itu.int/ITU-R/go/patents/en获得，其中还可找到ITU-T/ITU-R/ISO/IEC和ITU-R专利信息数据库共同专利政策实施指南。

ITU-R系列建议书

（也可在线查询**http://www.itu.int/publ/R-REC/en**）

系列	标题
BO	卫星传送
BR	用于制作、存档和播出的录制；电视电影
BS	广播业务（声音）
BT	广播业务（电视）
F	固定业务
M	移动、无线电测定、业余和相关卫星服务
P	无线电波传播
RA	射电天文
RS	遥感系统
S	卫星固定业务
SA	空间应用与气象学
SF	固定卫星与固定业务系统的频率共用与协调
SM	频谱管理
SNG	卫星新闻采集
TF	时间信号和频率标准发射
V	词汇和相关问题

注：本ITU-R建议书的英文版根据ITU-R第1号决议详述的程序予以批准。

电子出版
2020年，日内瓦

ITU-R BS.2127-0*建议书**

高级音响系统的音频定义模型渲染器

（2019年）

范围

本建议书规定了用于ITU-R BS.2051-2建议书中规定的高级音响系统和ITU-R BS.2076-1建议书中规定的音频定义模型（ADM）中的音频相关元数据的参考渲染器，包括用于节目交换的参考渲染器。音频渲染器在所提供的内容元数据和本地环境元数据的基础上，将一组附带相关元数据的音频信号转化为不同配置的音频信号和元数据。

注 – 正在制定解释渲染器使用的指南。

关键词

ADM、音频定义模型、元数据、渲染器、高级音响系统、信道基音频、基于音频的对象、场景基音频、多声道音频

国际电联无线电通信全会，

考虑到

a) ITU-R BS.1909-0建议书 –对带有或不带伴图的高级多声道立体声系统的性能要求，阐述了对带有或不带伴图的高级音响系统的要求；

b) ITU-R BS.2051-2建议书 – 用于节目制作的高级音响系统，定义了具有超出ITU-R BS.775-3建议书规定的再现配置的系统，或具有任何可支持基于声道、基于对象或基于场景的输入信号或此类输入信号与元数据的组合的高级音响系统。

c) ITU-R BS.2076-1建议书 –音频定义模型，定义了允许可靠地描述音频文件的格式和内容的元数据模型的结构；

d) ITU-R BS.2094-1建议书–音频定义模型的通用定义，包含一组音频定义模型的通用定义；

e) ITU-R BS.2125-0建议书–音频定义模型的串行表示，规定了基于音频定义模型的元数据格式，并将其分割成帧的时间序列；

f) 高级音响系统的再现需要呈现与声音信号相关的元数据渲染，以便将内容呈现给ITU-R BS.2051-2扬声器配置之一的内容；

g) 高级音响系统的用户可自由选择渲染方法；

h) 希望有可用于高级音响系统节目的单参考渲染方法的开放规范；

---

* 提请ISO、IEC、SMPTE和ETSI注意本建议书。

** 无线电通信第6研究组于2021年根据ITU-R第1号决议对本建议书进行了编辑性修正。

i) 单参考渲染器应允许内容制作者和广播公司在内容制作期间监控和执行质量控制，验证元数据的使用，并确保与制作链的其他元素的互操作性，

建议

1 附件1中所述的渲染方法应作为如何解释ITU-R BS.2076-1建议书中规定的ADM元数据和伴随的音频信号的参考；

2 下文注1被视为本建议书的一部分。

注1 – 可在自愿基础上遵循本建议书。但是，本建议书可能包括某些强制性的规定（以确保互操作性或适用性等），只有当所有这些强制性的规定都得到执行时才能使本建议书得到遵守。“须”或其他一些强制性语言（如“必须”）及其相应否定形式用以表示要求，使用这些词语绝不意味着要求部分或全部遵守本建议书。

附件 1

高级音响系统 ADM 渲染器的规范

目录

1 引言

本建议书描述了一个音频渲染器，它提供了对ITU-R BS.2076-1建议书中规定的音频定义模型（ADM）元数据的完整解释。建议使用ADM元数据来描述高级音响系统（AdvSS）节目制作中使用的音频格式，也称为下一代音频（NGA）系统。该渲染器能够将音频信号渲染到ITU-R BS.2051-2建议书中规定的所有扬声器配置。

本规范附带了一个开源参考实现，该实现是用Python编写的，用于基于文件的ADM处理，可从以下网址获得：

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

本规范文件是参考代码的说明。

1.1 缩略语/词汇表

ADM 音频定义模型
BMF 广播元数据交换格式
BW64 广播波形64格式
BWF 广播波形格式
HOA 高阶环境声学
NGA 下一代音频
PSP 点源声像定位器
VBAP 矢量基振幅平移
XML 可扩展标记语言

2 惯例

2.1 符号

在本建议书中，将采用下列惯例：

– 斜体文本指ITU-R BS.2076-1建议书：*audioObject*的ADM元素、子元素、参数或属性

– 等距文本指参考实现：`core.point_source.PointSourcePanner`的源代码（变量、函数、类）。应该注意的是，出于可读性的原因，前缀`iar.`省略。

– 大写粗体用于矩阵：**X**

– 小写粗体用于向量：**x**

– x_n形式的下标表示向量**x**的第n个元素

– 带有彩色突出显示的等距文本部分用于描述数据结构：

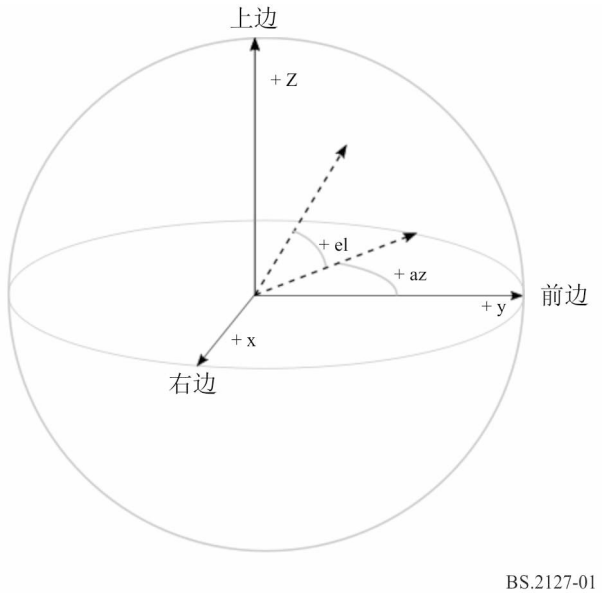
```
struct PolarPosition : Position {
float azimuth, elevation, distance = 1;
};
```

2.2 坐标系

本文件中同时使用笛卡尔坐标和极坐标。

图1

坐标系



极坐标按照ITU-R BS.2076-1建议书规定如下：

– 方位角，用φ表示，是水平面上的角度，前0度，正角度逆时针。

– 仰角，用θ表示，是水平面上方的角度，前0度，正角度向上。

笛卡尔坐标根据ITU-R BS.2076-1建议书规定如下：

– 正Y轴指向前方。

– 正X轴指向右边。

– 正Z轴指向顶部。

第9节中规定的HOA解码器使用ITU-R BS.2076-1建议书中规定的HOA坐标系和符号，其中：

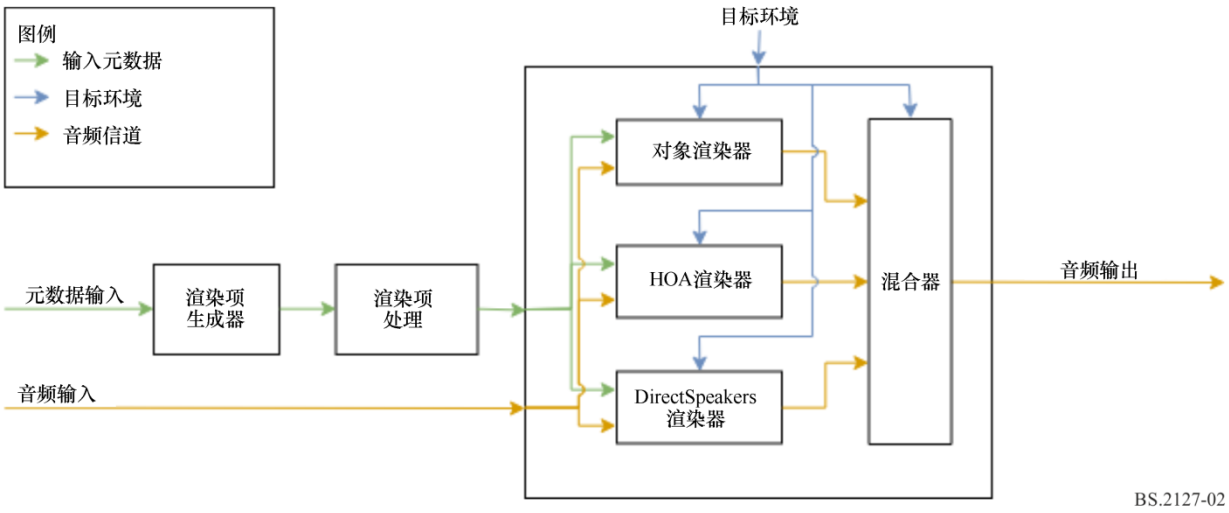
– 仰角θ是由正Z轴的弧度所表示的角。

– 方位角，用ϕ表示，是以弧度表示的水平面内的角度，前为0，正角度逆时针。

3 结构

图2

总体架构概述



总体架构由几个核心组件和处理步骤组成，本文件的以下章节将对此进行描述。

– ADM数据到一组可渲染项的转换在第5.2节中描述。

– 应用重要性和转换模拟的可选处理应用于第5.3节所述的渲染项。

渲染本身根据项的类型（*typeDefinition*）拆分为子组件：

- 第7节描述了基于对象内容的渲染。
- 直接扬声器信号的渲染如第8节所述。
- HOA渲染见第9节。
- 所有零件的共用部件在第6节中说明。

图中不显示矩阵类型处理，因为此类型是在创建渲染项期间处理的，并且是其他类型的渲染器的一部分。

3.1 目标环境行为

在初始化时，用户可以从ITU-R BS.2051-2建议书中规定的扬声器布局中选择扬声器布局。

每个扬声器的标称位置（polar_nominal_position）如ITU-R BS.2051-2建议书所规定。M+SC和M-SC的标称方位角为15°和−15°。

每个扬声器的实际位置（polar_position）可由用户指定。如果没有给出，则使用标称位置。根据ITU-R BS.2051-2建议书中给出的范围检查给定的实际位置；如果它们不在范围内，则发出错误。此外，M+SC和M-SC扬声器的绝对方位角必须介于5°和25°之间，或介于35°或60°之间。

4 ADM-XML接口

ADM是一个通用的元数据模型，可以自然地表示为XML文档。以下小节描述如何将ADM映射到内部数据结构。这些是在本建议书的过程中使用的，并且与参考实现所使用的数据结构一致。

应该注意的是，尽管XML是表示ADM元数据的典型和通用形式，但是渲染器并不限于此表示。

ADM和内部数据结构之间的映射遵循一组简单的规则，如下所述。与所有规则一样，也有一些例外；这些在下面的小节中进行了描述。

– 所有主要ADM元素应表示为从ADMElement派生的子类，该子类具有签名：

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

– 每个ADM元素类应扩展为所有ADM属性和子元素，这些属性和子元素映射到类属性。

– 如果子元素包含多个值，则它本身就是一个类。例如，*jumpPosition*子元素是一个具有以下签名的类：

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

– 在XML解析过程中，对其他ADM元素的引用以普通ID的形式存储，使用子元素名作为属性名（例如AudioObject.audioPackFormatIDRef）。为了简化后面的访问，这些引用将在下面的步骤中解析，其中解析的元素将直接添加到每个数据结构（AudioObject.audioPackFormats）。

遵循这些规则，AudioContent元素的完整签名如下所示：

```
class AudioContent : ADMElement {
  string audioContentName;
  string audioContentLanguage;
  LoudnessMetaData loudnessMetadata;
int dialogue;
```

```
    vector<AudioObject*> audioObjects;
    vector<string> audioObjectIDRef;
  };
```

主要ADM元素及其专用类在fileio.adm.elements.main_elements元素中实现。引用解析在每个类（在ADM和每个主ADM元素中）中实现为lazy_lookup_references方法。

ADM的解析和写入是在fileio.adm.xml中实现的。

4.1 AudioBlockFormat

*audioBlockFormat*不同于其他ADM元素，因为其子元素和属性因类型定义而异。为了反映这一点，AudioBlockFormat被分成多个类，每个类对应一个支持的*typeDefinition*：AudioBlockFormatObjects、AudioBlockFormatDirectSpeakers和AudioBlockFormatHoa。

它们以fileio.adm.elements.block_formats实现。

4.2 位置子元素

位置由ADM中的多个位置子元素表示。为了简化内部处理，这些子元素的值被合并到AudioBlockFormat表示中的单个属性中。

对于*typeDefinition==Objects*，这是ObjectPolarPosition或ObjectCartesianPosition，具体取决于使用的坐标系。对于*typeDefinition==DirectSpeakers*，这是DirectSpeakerPolarPosition或DirectSpeakerCartesianPosition。

4.3 TypeDefinition

*typeDefinition*和*typeLabel*属性描述一个单独的属性。因此，在内部只能使用一个实体来代表它们。

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

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

5 渲染项

RenderingItem是要渲染的ADM项的一种表示，它包含了所有必要的信息。因此，一个项目应代表一个单独的*audioChannelFormat*或一组*audioChannelFormats*。由于每个*typeDefinition*都有不同的要求，因此有必要为每个*typeDefinition*使用不同的元数据结构以适应其特定的需要。

下一节将更详细地描述使用的元数据结构。

5.1 元数据结构

RenderingItems基于以下基类：

- TypeMetadata，用于保存渲染项所需的所有（可能是时变的）参数；
- MetadataSource保存一系列TypeMetadata对象；以及
- RenderingItem将MetadataSource与音频样本源和渲染器不需要的额外信息相关联。

由于每个*typeDefinition*都有不同的要求，因此必须为每个*typeDefinition*子类化TypeMetadata和RenderingItem，以适应其特定需求。MetadataSource与*typeDefinition*无关。公共数据合并为ExtraData：

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

重要数据应存储在ImportanceData结构中：

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

对输入音频样本的引用应封装在TrackSpec结构中，以允许指定无声轨迹和矩阵处理。DirectTrackSpec指定应直接从指定的输入轨迹读取样本。SilentTrackSpec规定所有样本均为零。

```
struct TrackSpec {};

struct DirectTrackSpec : TrackSpec {
int track_index;
};

struct SilentTrackSpec : TrackSpec {
};
```

提供了两种TrackSpec类型来支持*typeDefinition==DirectSpeakers*。MatrixCoefficientTrackSpec指定将coefficient（来自矩阵*audioBlockFormat*coefficient元素）中指定的参数应用于input_track的样本，而MixTrackSpec指定应将来自多个TrackSpecs的样本混合在一起。

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

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

这在core.utils.metadata_input输入中实现。下面的小节将更详细地描述每个*typeDefinition*的具体实现。

5.1.1 DirectSpeakers

对于*typeDefinition==DirectSpeakers*，TypeMetadata应包含*audioBlockFormat*、包含*audioChannelFormat*的*audioPackFormats*列表以及在ExtraData中收集的公共数据。

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

由于*typeDefinition==DirectSpeakers*的每个*audioChannelFormat*都可以独立处理，RenderingItem只包含一个TrackSpec。

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

5.1.2 Matrix（矩阵）

在渲染其他类型的项时，应使用TrackSpec机制支持*typeDefinition==Matrix*，因此不需要明确的MatrixTypeMetadata或MatrixRenderingItem类。

5.1.3 Objects（对象）

ObjectTypeMetadata应保存*audioBlockFormat*和在ExtraData中收集的公共数据。

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

由于*typeDefinition==Objects*的每个*audioChannelFormat*都可以独立处理，RenderingItem只能包含一个单独的TrackSpec。

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

5.1.4 HOA

对于*typeDefinition==HOA*，情况与*typeDefinition==DirectSpeakers*和*typeDefinition==Objects*不同，因为一组*audioChannelFormats*需要一起处理。这就是为什么HOATypeMetadat不包含audioBlockFormat和ExtraData，而是从*audioBlockFormat*中提取必要的信息并直接存储在HOATypeMetadata中。

```
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;
};
```

出于同样的原因，HOARenderingItem的情况不同。在这里，HOARenderingItem不仅包含一个TrackSpec，还包含一个TrackSpecs向量。

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

5.1.5 Binaural

由于不支持*typeDefinition*==*Binaural*，因此没有BinauralTypeMetadata或BinauralRenderingItem类。

5.2 渲染项的确定

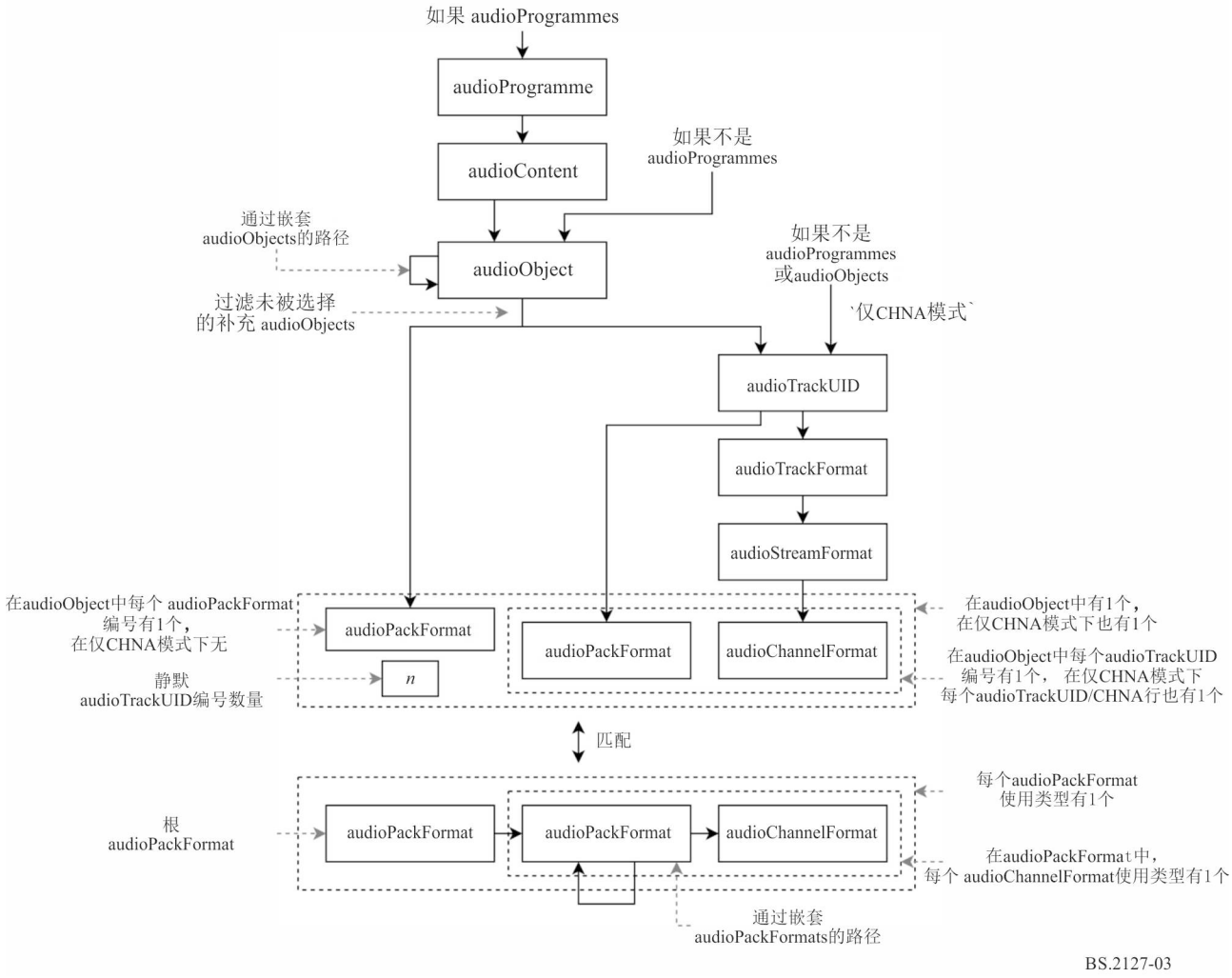
为确定RenderingItems，应分析ADM结构。图3说明了所采用的路径。

项选择过程的状态在称为“项选择状态”的单个对象中的各个组件之间进行，当完全填充时，该状态表示构成单个RenderingItem的所有组件。每个组件接受单个项选择状态，并返回其副本（从零到多），其中填充了更多项。这些步骤在select_rendering_items中组合在一起，这是每个组件依次修改时状态上的嵌套循环。

这在core.select_items中实现。

图3

通过ADM结构路径确定RenderingItems



5.2.1 开始点

根据文件中包含的元素，渲染项选择可以从ADM结构中的多个点开始。

*audioProgramme*元素，则选择单个*audioProgramme*；否则，如果有*audioObject*元素，则应选择所有*audioObjects*；否则，将选择所有*audioTrackUID*s（CHNA行）（称为“仅CHNA模式”）。

5.2.2 audioProgramme选择

只选择了一个*audioProgramme*。用户可以选择要使用的程序。如果未选择*audioProgramme*，则应选择ID数值最低的音频节目。

5.2.3 audioContent选择

所选*audioProgramme*引用的所有*audioContents*都将被选中。

5.2.4 audioObject选择

`audioObjects`应设置为通过*audioObject*层次结构的所有可能路径，依次从选定的*audioContent*（以下*audioObject*链接）开始。

5.2.5 补充*audioObject*处理

*audioComplementaryObject*引用应解释为定义*audioObjects*组，其中仅复制一个*audioObject*。

组由*audioComplementaryObject*引用来描述，从组中的默认*audioObject*到组中的所有非默认*audioObjects*。用户可以提供一组要选择的音频对象，这将覆盖默认值。由此，确定一组要忽略的*audioObjects*，如果*audioObject*路径中的任何*audioObjects*在此集合中，则丢弃状态。

5.2.5.1 选择要忽略的互补*audioObjects*

首先，用户选择的*audioObjects*集应使用每个组的默认值进行扩充：对于每个根*audioObject*（具有*audioComplementaryObject*引用的*audioObject*），如果根*audioObjects*定义的组中没有任何*audioObjects*，则应添加根*audioObjects*（默认值）。

要忽略的*audioObjects*集是所有互补audioObjects（即具有*audioComplementaryObject*引用的*audioObjects*和由*audioComplementaryObject*引用指向的audioObjects）的集合减去用户选择的*audioObjects*的扩充集合。

如果选择了不属于任何补充组的*audioObjects*，或者在单个*audioObject*组中选择了多个*audioObject*（可能是由于用户错误，也可能是由于组重叠），则会引发错误。

5.2.6 *audioPackFormat*匹配

下一步应根据*audioPackFormat*和*audioChannelFormat*结构匹配*audioObject*中的信息（*audioPackFormat*、*AudioTrackUID*和静默音轨数量的列表，或仅匹配CHNA模式下的所有*audioTrackUID*的列表）。

这被指定为一个匹配/搜索问题，而不是必须解决的通过引用结构的特定路径，因为两边有多个元素必须匹配而不冲突才能形成有效的解决方案。

只有找到一个解决方案时，匹配才被视为有效。如果找不到解决方案，则元数据是相互矛盾的，并应提出错误。如果发现多个解决方案，则元数据不明确，并将引发错误。对于这两种类型的错误，都会运行诊断程序，以便向用户显示错误的可能原因。

5.2.6.1 要匹配的包

要与之匹配的*audioPackFormats*的规范作为`AllocationPack`结构的列表给出：

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

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

每一个都应指定根*audioPackFormat*（`root_pack`，引用要分配的所有频道的顶级*audioPackFormat*）和要在该包中匹配的频道列表。每个频道都是*audioChannelFormat*引用和该频道可能关联的*audioPackFormats*列表的组合。

对于每个类型定义的audioPackFormatpack, 其 *typeDefinition* !=Matrix，创建AllocationPack对象，其中：

– root_pack是pack。

– channels为每个可从pack访问的*audioChannelFormat*（在*audioPackFormat*链接之后递归地）提供一个条目，其中pack_formats包含从pack到*audioChannelFormat*（包括pack）路径上的所有*audioPackFormat*s。

虽然这是*audioPackFormat*和*audioChannelFormat*结构的一个小简化，但是这种表示的优点是它能够以*Matrix*内容表示*audioPackFormat*和*audioChannelFormat*引用结构，如下所述。

5.2.6.1.1 矩阵处理

矩阵*audioPackFormats*可以根据预期效果以多种方式引用。这些引用结构反映在以下为每个*typeDefinition*==*Matrix*的audioPackFormat pack生成的AllocationPacks中：

– 如果pack是一个直接或解码矩阵，那么如果一个*audioObject*同时引用pack和一组*audioTrackUID*s，而这些*audioTrackUID*s反过来引用pack和pack的输入信道或编码*audioPackFormat*，则应应用该矩阵：

  • root_pack就是pack。

  • channels在pack的输入*audioPackFormat*中为每个*audioChannelFormat*信道包含一个值（根据类型，可以是*encodePackFormat*或*inputPackFormat*），其中channel_format格式是channel，pack_formats格式是[pack]。

– 如果pack是一个直接或解码矩阵，如果一个*audioObject*同时引用pack和一组*audioTrackUID*s，进而引用pack（或子pack）和pack的频道，则该矩阵应被视为以前应用于文件中的样本：

  • root_pack就是pack。

  • channels在pack中为每个*audioChannelFormat*channel包含一个值，其中channel_format是channel，pack_formats包含从pack到channel路径上的所有*audioPackFormat*s。

– 如果pack是一个解码矩阵，则如果*audioObject*引用pack和一组*audioTrackUID*s（反过来引用*encodePackFormat*和*encodePackFormat*的*inputPackFormat*的信道），则可以应用其*encodePackFormat*，后跟pack：

  • root_pack是pack。

  • channels在pack的*encodePackFormat*的*inputPackFormat*中，为每个*audioChannelFormat*信道包含一个值，其中channel_format为channel，pack_formats包含从*inputPackFormat*到channel路径上的所有*audioPackFormats*。

矩阵*audioPackFormat*的“类型”由以下规则决定：

– 如果它同时引用*inputPackFormat*和*outputPackFormat*，它是一个直接矩阵。

– 如果它引用了*inputPackFormat*，没有引用*outputPackFormat*，那么它是一个编码矩阵。

– 如果它引用了*outputPackFormat*，没有引用*inputPackFormat*，那么它是一个解码矩阵。

– 如果它既没有引用*inputPackFormat*，也没有引用*outputPackFormat*，则会引发错误。

5.2.6.2 要匹配的音轨和*audioPackFormat*引用

由AllocationPacks匹配的音轨应由以下三个规则确定：

– tracks，一个AllocationTracks的列表，每个声道意味着一个*audioTrackUID*（或CHNA行）：

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

channel_format 是通过遵循 *audioTrackFormat* 、 *audioStreamFormat* 和 *audioChannelFormat* 引用从 *audioTrackUID* 获得的， 而 pack_format 是由 *audioTrackUID*直接引用的。

– pack_refs，在*audioObject*中找到的*audioPackFormat*引用的可选列表。

num_silent_tracks，要分配的“静默”音轨数，表示为从*audioObject*到ATU_00000000的引用。

为*audioObject*确定这些结构时：

– tracks为从*audioObjec*引用的每个（非静默）*audioTrackUID*包含一个条目。

– pack_refs包含在*audioObject*中的*audioPackFormat*引用列表。

– num_silent_tracks是引用的静默*audioTrackUID*s的数量（对应于*audioObject*中对ATU_0000000的引用）。

在仅CHNA模式下：

– tracks为文件中的每个*audioTrackUID*（或CHNA行）包含一个条目。

– pack_refs为None。

– num_silent_tracks是0。

5.2.6.3 匹配

匹配解决方案被指定为AllocatedPack对象的列表：

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

每一个都将pack中的每个*audioChannelFormat*与一个track相关联，如果未指定AllocationTrack，则将其与静默音轨相关联。

有效的解决方案具有以下属性：

1. 对于每个AllocatedPack，AllocationPack中的每个信道在allocation中只发生一次。
2. 每个在tracks中的音轨在输出中恰好发生一次。
3. 输出中引用的静默音轨数等于num_silent_tracks。
4. 对于每个关联的AllocationChannel channel和AllocationTrack track，track.channel_format 为 channel.channel_format ，track.pack_format为channel.pack_formats。

5. 如果pack_refs不是None，则pack_refs与每个分配的pack的pack.pack.root_pack值之间存在一对一的对应关系。

除了AllocationPacks的顺序或其中的allocations顺序之外，其他相同的解决方案都被认为是等效的。

可以使用枚举所有有效且唯一（非等效）解的任何方法。在参考实现中，通过将上述属性视为约束满足问题并使用回溯搜索枚举所有解决方案来找到解决方案。

5.2.6.3.1 例子

包的格式匹配在以下一系列的例子中重点阐述。

首先在例子中应用的结构会被定义。c1、c2等和p1、p2等呈现了对*audioChannelFormat*s和*audioPackFormat*s的引用（但可能像allocate_packs这样的对象都仅仅会用Allocation...结构中的信息，用特征来比较这些引用之间的不同）。

一个单声道包和引用它的音轨：

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

一个双信道包，有两对引用音轨：

```
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)
```

解析*audioObject*中的单个单声道会产生包含单个分配包的单个解决方案：

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

解析在仅CHNA模式下的单个单声道音轨会产生同样的结构：

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

解析一个静默音轨会产生同样的结构，除非对音轨的引用被None代替：

```
assert allocate_packs(
    packs=[ap1, ap2],
    tracks=[],
```

```
        pack_refs=[p1],
        num_silent_tracks=1,
    ) == [[AllocatedPack(pack=ap1, allocation=[(ac1, None)])]]
```

如果在包引用中信道比音轨更多，那么因为规则2与规则5相斥会导致没有解决方案。

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

如果在包引用中静默音轨比信道更多，那么由于规则2与规则5相斥会导致没有解决方案。

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

如果在包引用与在音轨中的信道/音轨信息不相符，那么由于规则1、4和5相斥，会导致没有解决方案。

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

如果*audioObject*中有多个多信道包实例，则对包的音轨分配不明确，因此有多个解决方案：

```
    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 后处理的解决方案

应该注意的是，匹配的结果是根据输入结构（`AllocationPack`、`AllocationChannel`、`AllocationTrack`）指定的，而不是对ADM结构的底层引用。这是为了允许*audioPackFormat*和*audioChannelFormat*引用（在*audioObject*和*audioTrackUID*中）与提供给渲染器的信息之间的任意映射，因为使用*typeDefinition*==*Matrix*时没有简单的对应关系。

对于非矩阵AllocatedPack pack，映射很简单。output_pack是pack.pack.root_pack，pack.allocation中的分配与实际信道分配之间有一对一的映射：AllocationChannel channel被映射到channel.channel_format，AllocationTrack track被映射到与track关联的*audioTrackUID*（或CHNA行）的跟踪索引的DirectTrackSpec，并且丢失的AllocationTrack被映射到SilentTrackSpec。

对于矩阵AllocatedPack pack，需要更复杂的映射：

pack.root_pack总是一个解码包或直接包（见第5.2.6.1.1节），因此output_pack是pack.root_pack.outputPackFormat。

跟踪分配的输出信道包含root_pack中每个*audioChannelFormat*matrix_channel的一个条目。这些信道与*outputChannelFormat*引用建立的output_pack中的*audioChannelFormats*有一对一的对应关系。

*audioChannelFormat*是matrix_channel.block_formats[0].outputChannelFormat。

TrackSpec是通过递归地遵循*inputChannelFormat*从matrix_channel到pack.allocation中引用的*audioChannelFormat*s来构建的，嵌套MatrixCoefficientTrackSpecs和MixTrackSpecs以应用*coefficient*元素中指定的处理并将多个输入信道混合在一起：

– 如果pack.allocation中引用了matrix_channel，则返回与关联的AllocationTrack相对应的DirectTrackSpec或SilentTrackSpec（见上文）。

– 否则，返回包含matrix_channel.block_formats[0].matrix中每个*coefficient*元素c的MatrixCoefficientTrackSpec的MixTrackSpec。将c中指定的处理应用于递归确定的c.inputChannelFormat的跟踪规范。

在参考实现中，这是在AllocationPack的两个子类中实现的，这两个子类具有查询*audioPackFormat*和信道分配的方法，供渲染器使用。AllocationTracks和它们对应的*audioTrackUID*s之间的关联同样使用AllocationTrack的子类来维护。

5.2.7 输出渲染项

一旦确定了根*audioPackFormat*，并且为其每个信道分配了TrackSpec，找到的所有信息都将转换为一个或多个RenderingItems。

执行此操作的过程取决于根*audioPackFormat*的类型。

5.2.7.1 共享组件

渲染项中的某些数据在类型之间共享，因此也以相同的方式派生。

5.2.7.1.1 重要性

ImportanceData对象应派生自项选择状态，其值如下：

– audio_object在路径的所有*audioObject*s中被指定为最不重要。

– audio_pack_format具有从根*audioPackFormat*到*audioChannelFormat*路径上的任何*audioPackFormat*中指定的最低重要性。

在这两种情况下，None（未指定重要性）被定义为最高重要性。

5.2.7.1.2 额外数据

ExtraData对象应派生自项选择状态，其值如下：

– object_start是路径上最后一个*audioObject*的*start*时间（在仅CHNA模式下None）。

– object_duration是路径上最后一个*audioObject*的*duration*（在仅CHNA模式下None）。

– reference_screen是所选*audioProgramme*的*audioProgrammeReferenceScreen*（没选即为None）。

– channel_frequency是所选*audioChannelFormat*的*frequency*要素。（或者None，如果没有选择一个，比如在创建一个HOA渲染项时）。

5.2.7.2 typeDefinition==Objects或DirectSpeakers的输出渲染项

确定*Objects*和*DirectSpeakers*的渲染项的过程是相似的–只是所涉及的类型和参数的选择不同。

在信道分配中，每对*audioChannelFormat*和track_spec生成一个渲染项。

创建一个MetadataSource，在选定的*audioChannelFormat*中为每个*audioBlockFormat*生成一个RenderingItem（适当类型），其中extra_data字段如上所述确定，并且audioPackFormats字段包含根*audioPackFormat*和*audioChannelFormat*之间路径上的所有*audioPackFormat*。它被包装在RenderingItem对象中（同样是适当类型的），其track_spec和importance如上所述确定。

5.2.7.3 *typeDefinition==HOA*的输出渲染项

每个包分配生成一个HOARenderingItem，其中包含渲染构成HOA流的一组信道所需的所有信息。此信息跨多个*audioChannelFormats*和*audioPackFormats*（嵌套时）传播，它们必须一致。

HOA *audioChannelFormats*只能包含一个*audioBlockFormat*元素；否则会引发错误。

创建一个HOATypeMetadata对象，其参数根据表1派生。

表1

HOATypeMetadata参数的性质

HOATypeMetadata参数	*audioBlockFormat*参数	*audioPackFormat*参数	计数
rtime	*rtime*		单一
duration	*duration*		单一
orders	*order*		每个信道
degrees	*order*		每个信道
normalization	*normalization*	*normalization*	单一
nfcRefDist	*nfcRefDist*	*nfcRefDist*	单一
screenRef	*screenRef*	*screenRef*	单一

应首先确定根*audioPackFormat*中每个*audioChannelFormat*的所有参数。对于同时具有*audioBlockFormat*和*audioPackFormat*参数的参数，可以在*audioChannelFormat*中的唯一*audioBlockFormat*上设置该参数，或者在从根*audioPackFormat*到*audioChannelFormat*的路径

上设置任何*audioPackFormat*。如果为给定的*audioChannelFormat*找到一个参数的多个副本，则它们的值应相同，否则将引发错误。如果找不到给定参数和*audioChannelFormat*的值，则应用ITU-R BS.2076-1建议书中指定的默认值。

找到特定*audioChannelFormat*的*nfcRefDist*后，0值应转换为`None`，这意味着不应应用NFC。这是在这个阶段（而不是在XML解析期间）执行的，例如，*nfcRefDist==0.0*被认为与*nfcRefDist==1.0*冲突。

对于只有一个值的参数（除`orders`和`degrees`外），所有*audioChannelFormats*的参数应相等，否则将产生错误。

整个*audioPackFormat*的`extra_data`如上所述确定。

应生成一个`HOARenderingItem`，其中一个条目在信道分配（如上所述）中的每一个条目的`track_specs`和`importances`中，一个`MetadataSource`仅包含上述`HOATypeMetadata`对象。

5.3 渲染项处理

有些渲染项功能是通过修改选定渲染项清单而实现的。第5.3.1节描述了内容如何根据特定的重要级别而被删除，第5.3.3节描述了下行元数据转换的影响是如何模拟的。

5.3.1 重要性模拟

ITU-R BS.2076-1建议书定义的*importance*参数允许渲染器出于尚未确定的、特定于应用程序的原因丢弃低于某个重要级别的项。

ADM指定了应使用的三个不同的重要参数：

– 作为audioObject属性的*importance*

– 作为audioPackFormat属性的*importance*

– 作为typeDefinition==Object的audioBlockFormat属性的*importance*

这些*importance*属性之间最重要的区别是，*audioBlockFormat*的重要性是时间依赖的，即它可能随时间而变化，而*audioObject*和*audioPackFormat*的重要性是静态的。

每一个*importance*属性都可以用单独的阈值。所需要的阈值的确定被认为是高度应用程序和特定用例，因此超出了产品渲染器规范的范围。相反，渲染器提供了模拟将给定的重要性阈值应用于ADM的效果的方法。这使内容生产者能够调查使用*importance*值对渲染的影响。因此，重要性仿真不是实际渲染过程的一部分，而是作为`RenderingItems`的后处理步骤应用的。

5.3.1.1 `RenderingItems`的重要性值

每个渲染项可以有自己的一组有效*importance*值，因为*audioObjects*和*audioPackFormats*可能是嵌套的。因此，对于每个`RenderingItem`，将考虑确定该`RenderingItem`所涉及的所有引用*audioObjects*和*audioPackFormats*。

适用以下规则：

– 如果audioObject的importance值低于阈值，则也应丢弃所有引用的audioObjects。为了实现这一点，导致RenderingItem的所有audioObjects的最低importance值应用作该RenderingItem的audioObjectimportance。
– 如果audioPackFormat的importance值低于阈值，则也应丢弃所有引用的audioPackFormat。为了实现这一点，应将导致RenderingItem的所有audioPackFormats的最低importance值用作此RenderingItem的audioPackFormatimportance。
– 在确定RenderingItem的importance时，不应考虑没有importance值的audioObject。
– 在确定RenderingItem的importance时，不应考虑没有importance值的audioPackFormat。

这在fileio.utils.RenderingItemHandler中实现。

5.3.1.2 静态重要性处理

给定一个具有ImportanceData的RenderingItem，如果静态重要性值（*audioObject*、*audioPackFormat*）低于相应的用户定义阈值，则应从要渲染的项列表中删除该项：

$$\text{importance.audio\_object} < \text{audio\_object\_threshold}$$
$$\vee\ \text{importance.audio\_pack\_format} < \text{audio\_pack\_format\_threshold}$$

这在core.importance.filter_audioObject_by_importance和core.importance.filter_audioPackFormat_by_importance中执行。

5.3.1.3 时变重要性处理

audioBlockFormat（*typeDefinition==Object*）级别上的重要性处理不能通过筛选RenderingItems来完成，因为此项可能仅在一段时间内低于阈值。为了模拟在该特定情况下丢弃RenderingItem，在*audioBlockFormat*的持续时间内，RenderingItem应有效地静音。在这种情况下，“静音*audioBlockFormat*”相当于假定*audioBlockFormat*bf.的bf.gain等于零。

这在core.importance.MetadataSourceImportanceFilter中实现。

5.3.2 转换仿真

元数据转换的模拟可以选择性地应用于渲染项。转换仿真可能被禁用，设置为将元数据转换为极坐标形式，或设置为将元数据转换为笛卡尔形式。

如果启用转换模拟，则从第10节中选择适当的函数，并将其应用于所选渲染项中*typeDefinition==Objects*的*audioBlockFormats*。

6 共享渲染器组件

本节包含在不同typeDefinitions的子渲染器之间共享的组件的描述。

6.1 极坐标点源声像定位器

点源声像定位器组件是渲染器的核心；给定扬声器布局和3D方向的信息，它为每个扬声器产生一个增益，当应用于单声道波形/数字信号并在扬声器上再现时，该增益应使侦听器感知从所需方向发出的声音。

点源声像定位器用于整个渲染器–它用于渲染由对象元数据指定的点源，以及扩展渲染系统的一部分，作为*DirectSpeakers*渲染器的回退，以及作为HOA解码器设计过程的一部分。

此渲染器中的点源声像定位器基于VBAP公式[2]，具有一些增强功能，使其更适合在广播环境中使用：

– 除了VBAP中的三元组扬声器外，点源声像定位器还支持扬声器的原子四边形。这解决了与在其他系统中使用虚拟扬声器相同的问题，但会产生更平滑的整体声像定位功能。

– 扬声器布局的三角测量在标称扬声器位置上执行，并扭曲以匹配实际扬声器位置，从而确保在给定布局的自适应范围内，声像定位行为始终一致。

– 虚拟扬声器和向下混合用于修改在某些情况下的渲染，以便校正观察到的感知效果并在稀疏布局中得到期望的行为。

– 为避免设计复杂化，以满足极为有限的扬声器布局，0+2+0作为特殊情况处理。

6.1.1 结构

点源声像定位器拥有一个带有`RegionHandler`接口的对象列表；每个区域对象应负责在给定的空间范围内产生扬声器增益。

为了产生给定方向的增益，点源声像定位器应依次查询每个区域，如果可以处理该方向，则返回一个增益矢量；如果不能处理，则返回一个空结果；使用从找到的第一个可以处理该方向的区域得到的增益矢量。

在任何有效的点源声像定位器中，以下两个条件都适用：

– 至少有一个区域能够处理任何给定的方向。

– 所有能够处理给定方向的区域都会产生类似的增益（在一定的公差范围内）。

– 在任何区域内，所产生的增益相对于期望的方向都是平滑的。

这些特性共同确保了点源声像定位器所产生的增益在所有方向上都有很好的定义，并且在一定的公差范围内相对于方向总是平滑的。

下一节将介绍可用的`RegionHandler`类型以及用于为给定布局生成区域列表的配置过程。

此行为在`core.point_source.PointSourcePanner`中实现。

另外，`PointSourcePannerDownmix`类是用相同的接口实现的。当查询位置时，它调用另一个`PointSourcePanner`来获取增益向量，并对其应用向下混合矩阵和功率常态化。这在第6.1.3.1节中用于重新映射虚拟扬声器。

6.1.2 区域类型

大多数区域为输出信道的子集产生增益；从该信道子集到信道的全矢量的映射在core.point_source.RegionHandler.handle_remap中实现。

6.1.2.1 三重态

这表示由三个扬声器组成的球形三角形区域，实现基本的VBAP。

该区域应使用三个扬声器的3D位置进行初始化：

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

给定方向D的三个输出增益$\mathbf{g}$如下：

- $\mathbf{g} \cdot \mathbf{P} = s\mathbf{d}$对于一些$s > 0$，在小公差范围内。
- $g_i \geq 0 \; \forall \; i \in \{1,2,3\}$
- $\| \mathbf{g} \|_2 = 1$

此RegionHandler类型在core.point_source.Triplet中实现。

6.1.2.2 VirtualNgon

这表示由n个真实扬声器组成的区域，该区域通过添加一个虚拟扬声器分成三角形。每个三角形由两个相邻的真实扬声器和虚拟扬声器组成，虚拟扬声器通过提供的下混系数下混到真实扬声器。

例如，如果使用四个实际扬声器位置$\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4\}$和一个虚拟扬声器位置$\mathbf{p}_v$，则将创建以下三角形：

- $\{\mathbf{p}_v, \mathbf{p}_1, \mathbf{p}_2\}$
- $\{\mathbf{p}_v, \mathbf{p}_2, \mathbf{p}_3\}$
- $\{\mathbf{p}_v, \mathbf{p}_3, \mathbf{p}_4\}$
- $\{\mathbf{p}_v, \mathbf{p}_4, \mathbf{p}_1\}$

当使用位置查询此RegionHandler类型时，应依次尝试每个三角形，直到其中一个三角形返回有效增益，方法与顶级点源声像定位器相同。这就产生了一个n增益的矢量，用于实际扬声器，$\mathbf{g} = \{g_1, \ldots, g_n\}$,，以及虚拟扬声器$g_v$的增益，该增益通过提供的下混系数$\mathbf{w}_{\text{dmx}}$下混到实际扬声器：

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

最后，这是增益的常态化，导致最终增益：

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

此RegionHandler类型在core.point_source.VirtualNgon中实现。

6.1.2.3 QuadRegion（四边形区域）

这表示由四个扬声器组成的球形四边形区域。

扬声器的增益是通过首先把方向划分为x和y两个分量来计算的，x可以被认为四边形内的水平方向，左边缘为0，右边缘为1，y是垂直位置，下边缘为0，上边缘为1。

使用方程(1)和（2）将x和y映射到每个扬声器的增益。通过解方程（1）和（3）可以确定产生给定速度矢量的x和y值（也得到了扬声器增益）。

该问题的解决方案与VBAP具有相似的复杂性，并且在四边形的边缘处得到与VBAP相同的增益，使得可以在第6.1.1节的规则下与单点源声像定位器中的其他`RegionHandler`类型一起使用。

所得到的增益相对于区域内的位置是无限可微的，产生的结果与在常见情况下虚拟扬声器之间的成对声像定位相当。

此`RegionHandler`类型在`core.point_source.QuadRegion`中实现。

6.1.2.3.1 表达方法

给定四个扬声器的笛卡尔位置，从听者的角度来看，$\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]$按逆时针顺序排列，计算源方向$\mathbf{d}$的增益向量$\mathbf{g}$为：

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

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

其中x和y的选择使得速度矢量$\mathbf{g} \cdot \mathbf{P}$具有所需的方向$\mathbf{d}$。速度矢量$r$的大小不相关，因为增益是功率标准化的：

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

对于一些$r > 0$。

6.1.2.3.2 解决方案

给定一个x值，所有速度矢量$\mathbf{d}$与该x值在一个平面上，该平面由坐标系原点和沿四边形顶部和底部一定距离的两点构成：

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

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

因此：

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

这个方程可以解出给定源方向$\mathbf{d}$的x。

收集x项：

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

展开交叉积并收集项：

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

最后，乘以$\mathbf{D}$：

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

因此x的解是多项式的跟，可以用标准方法求解。

通过用上述方程中的$\mathbf{P}'$代替$\mathbf{P}$也可以确定y：

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

然后可以使用方程1和2计算增益$\mathbf{g}$。由于在方程（4）中忽略了$\mathbf{d}$的范围，因此可以找到产生与所需速度矢量正相反的速度矢量的解。可以通过以下测试进行检查：

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

6.1.2.4 StereoPanDownmix（立体声声像定位下混）

立体声（0+2+0）点源的输出信号由基于从0+5+0到0+2+0的下混频的方法提供。该方法是单独实现的。

程序如下：

– 使用配置为0+5+0的点源声像定位器声像定位输入方向，以产生按M+030、M-030、M+000、M+110、M-110顺序的5个增益矢量$\mathbf{g}'$。

– 应用从0+5+0到0+2+0的格式转换矩阵，以M+030、M-030的顺序产生立体声增益$\mathbf{G}''$：

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

– 使$\mathbf{g}''$功率正常化至由$\mathbf{g}'$中前后扬声器之间的平衡所确定的值，使得M+030和M-030之间的源不衰减，而M-110和M+110之间的源衰减3dB。

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

此RegionHandler类型在core.point_source.StereoPanDownmix中实现。

注 – $\mathbf{g}$从（0+5+0）到（0+2+0）与ITU-R BS.775建议书中规定的下混系数完全匹配，如下所示：

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

6.1.3 配置过程

配置过程生成一个点源声像定位器，其中包含给定布局的上述RegionHandler类型。配置过程接受Layout对象（定义见第11.1.3节），并生成PointSourcePanner。

配置过程最初通过Layout::name属性选择行为。如果Layout::name属性为0+2+0，则配置由第6.1.3.2节中描述的立体声专用配置功能处理。所有其他情况由第6.1.3.1节中描述的通用功能处理。

配置过程在core.point_source.configure中处理。

6.1.3.1 通用布局过程

要为通用扬声器布局配置PointSourcePanner，请使用以下过程：

1. 使用M+SC或M-SC标签更新扬声器标称位置的方位角，以确保使用宽间隔屏幕扬声器进行正确的三角测量。如果实际方位角（polar_position.azimuth）为φ，则标称方位角φ_n（polar_nominal_position.azimuth）为：

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

2. 确定重新映射的虚拟扬声器集，如下所述。这些扬声器被添加到布局中的一组扬声器中，将被视为与实际扬声器相同。
3. 创建两个标准化笛卡尔扬声器位置列表，将在下一步中使用；一个包含标称扬声器位置（用于三角化扬声器布局），另一个包含实际扬声器位置（用于创建区域）。标称扬声器位置是ITU-R BS.2051-2建议书中规定的位置，而实际扬声器位置是当前再现系统实际使用的位置。
4. 在每个扬声器位置列表中，添加一个或两个虚拟扬声器，该扬声器将成为VirtualNgon的中心：
   - 总是添加$0,0,-1$（在监听器下方），因为ITU-R BS.2051-2建议书中未定义扬声器布局在该位置有扬声器。
   - 如果布局中没有标签为T+000或UH+180的扬声器，则添加$0,0,1$（监听器上方）。当UH+180存在时，这个扬声器不被使用的原因是，当在推荐ITU-R BS2051-2建议书中定义的3+7+0布局中使用时，该位置可以与虚拟扬声器的位置一致，从而在声像定位功能中产生阶跃变化。
5. 取扬声器标称位置的凸面外壳。如果使用浮点算法实现该算法，则错误可能会导致凸面外壳的某些面被拆分–面合并在公差集中，从而使结果与使用精确算法实现该算法的结果相同。
6. 创建具有以下区域的PointSourcePannerDownmix：
   - 对于不包含步骤3中添加的虚拟扬声器的凸面外壳的每个侧面：
     - 如果侧面有三条边，则创建一个Triplet，其中扬声器的实际位置对应于侧面的顶点。
     - 如果侧面有四条边，则创建一个QuadRegion，其中扬声器的实际位置对应于侧面的顶点。
   - 对于步骤3中添加的每个虚拟扬声器，在边缘创建一个具有相邻扬声器（与虚拟扬声器共享凸面外壳面的所有扬声器）的实际位置、位于中心的虚拟扬声器的位置以及设置为$\frac{1}{\sqrt{n}}$的所有下混系数的VirtualNgon，其中n是相邻扬声器的数量扬声器。

请注意，ITU-R BS.2051-2建议书中未定义的布局会导致具有四条以上边的面。

下混系数将虚拟扬声器映射到物理扬声器，如下所述。

这是在core.point_source._configure_full中实现的。

6.1.3.1.1 直接下混虚拟扬声器的测定

对于每个中间层扬声器，如果在该区域的上层或下层没有真正的扬声器，则在上层和下层以与真正的扬声器相同的方位角添加一个虚拟扬声器。这些虚拟扬声器应具有下混系数，该系数将其输出直接映射到相应的中级扬声器。

与真实扬声器一样，虚拟扬声器具有真实位置和标称位置，真实位置由真实扬声器的真实位置导出，标称位置由真实扬声器的标称位置导出。虚拟扬声器的包含与否取决于实际扬声器的标称位置，因此对于给定布局，始终使用相同的虚拟扬声器集。

要确定给定布局的虚拟扬声器集，请使用以下步骤：

– 对于每个$i \in [1, N]$，其中$N = \mathrm{len}(\mathrm{layouts.channels})$，定义信道数：

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

– 定义三组信道索引，确定布局的上、中、下层信道：

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

– 虚拟扬声器的标称和实际方位角与相应的真实扬声器相同。实际仰角是该层中实际扬声器的平均仰角（如果有），否则为下层和上层$-30°$或$30°$。下层和上层的标称仰角始终为$-30°$或$30°$。

定义两个标称仰角：

$$\theta'_{u,n} = 30°$$

$$\theta'_{l,n} = -30°$$

定义两个实际仰角：

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

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

– 如果相应的中层扬声器的绝对标称方位角大于或等于该层上的真实扬声器的最大绝对标称方位角，加上40°，则只在一个层上创建扬声器。这些方位角限制定义为：

$$L_u = \begin{cases} 0 & |S_u| = 0 \\ \max_{j \in S_u} |\varphi_{j,n}| + 40° & \text{otherwise} \end{cases}$$

$$L_l = \begin{cases} 0 & |S_l| = 0 \\ \max_{j \in S_l} |\varphi_{j,n}| + 40° & \text{otherwise} \end{cases}$$

– 对于每个j在S_m中：

- 如果$\varphi_{j,n} \geq L_u$，则创建一个虚拟上层扬声器，由Channel结构Channel标识，具有：

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

- 如果$\varphi_{j,n} \geq L_l$，则创建一个虚拟的低层扬声器，由一个Channel结构Channel标识，具有：

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

两者都具有下混系数，将增益从该扬声器路由到相应的中间层扬声器j。

这在core.point_source.extra_pos_vertical_nominal中实现。

6.1.3.2 0+2+0过程

对于0+2+0，将返回具有单个StereoPanDownmix区域的PointSourcePanner。

这在core.point_source._configure_stereo中实现。

6.2 确定角度是否在公差范围内

将角度与给定角度范围进行比较时，使用inside_angle_range函数，允许指定包括坐标系后部的范围。这在第7.3.12.1节和8.4节中的区域排除和*DirectSpeakers*组件中使用。

签名是：

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

如果角x在圆弧内，则返回true，该圆弧从start开始逆时针移动到end，并按tol展开。所有角度都以度为单位。

通常情况下其中：

$$-180 \le \mathrm{start} \le \mathrm{end} \le 180$$

此功能相当于：

$$\mathrm{start} - \mathrm{tol} \le \mathrm{x}' \le \mathrm{end} + \mathrm{tol}$$

其中$\mathrm{x}' = \mathrm{x} + 360 \times i$，对于某些$i$，这样$-180 < \mathrm{x}' \le 180$。

在其他情况下，这种行为更为微妙。例如，如果$\mathrm{start} = 90$，$\mathrm{end} = -90$，则指定坐标系的后半部分：

$$\mathrm{x}' \le -90 \vee \mathrm{x}' \ge 90$$

一些示例范围和等效表达式如表2所示。

表2

表达式相当于inside_angle_range (x, start, end, tol)

start	end	tol	同等表达
−90	90	0	$-90 \le \mathrm{x}' \le 90$
−90	90	5	$-95 \le \mathrm{x}' \le 95$
90	−90	0	$\mathrm{x}' \le -90 \vee \mathrm{x}' \ge 90$
90	−90	5	$\mathrm{x}' \le -85 \vee \mathrm{x}' \ge 85$
0	0	0	$\mathrm{x}' = 0$
180	180	0	$\mathrm{x}' = 180$
−180	−180	0	$\mathrm{x}' = 180$
180	180	5	$\mathrm{x}' \le -175 \vee \mathrm{x}' \ge 175$
−180	180	0	true

此功能在`core.geom.inside_angle_range`实现。

6.3 从频率元数据确定信道是否为LFE信道

频率元数据可以作为*audioChannelFormats*的*frequency*子元素存在，可用于确定信道是否有效地是LFE信道。

以下数据结构用于表示频率元数据：

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

有签名的函数

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

求值

$$\mathrm{frequency.lowPass} \wedge \neg \mathrm{frequency.highPass} \wedge (\mathrm{frequency.lowPass} \le 200\,\mathrm{Hz})$$

如果信道被假定为LFE信道，则返回`True`，否则返回`False`。

这是在`core.renderer_common.is_lfe`中实现的。

6.4 区块处理信道

当渲染定时ADM元数据时，一些功能需要与所有*typeDefinition*值相同–对于给定的输入信道子集，在时间范围之间进行一些处理，在输出上生成扬声器信道。

图4

用于处理相关信道的结构。外部提供的组件表示为蓝色

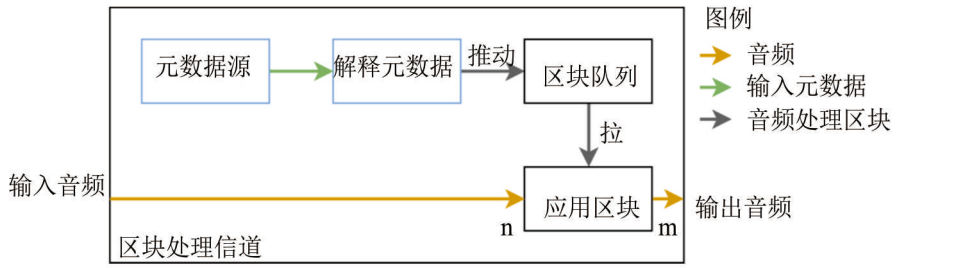


图4显示了用于实现此目的的结构。这个组件的接口如下:

```
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);
};
```

`MetadataSource`由系统提供，作为向渲染器馈送元数据的机制。它有以下接口：

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

通过重复调用`get_next_block`，区块处理信道接收如第5节所述的一系列`TypeMetadata`块，这些`TypeMetadata`块对应于渲染期间所需的有时间限制的元数据块。

这些元数据块由`interpret_metadata`函数解释，该函数由渲染器为每个*typeDefintion*提供。这些函数接受一个`TypeMetadata`并返回一个`ProcessingBlock`对象列表，该列表封装了实现给定`TypeMetadata`所需的有时间限制的音频处理。第7.2条详细说明了*typeDefinition==Objects*的解释。对于*typeDefinition==HOA*和*typeDefinition==DirectSpeakers*，将返回单个`ProcessingBlock`。

`ProcessingBlock`对象具有以下外部接口：

```
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);
}
```

传递给`process`的样本被假定为输入/输出文件中样本的子集，这样`input_samples`[i]和`output_samples`[i]代表全局输入和输出样本`in_out_samples_start` + i。`first_sample`和`last_sample`属性定义了受`process`影响的全局样本数s的范围：

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

start_sample和end_sample是分项开始和结束样本数，用于确定first_sample和last_sample属性，并可由ProcessingBlock子类实现。

BlockProcessingChannel对象存储一列ProcessingBlock，该队列通过从metadata_source请求块，并通过interpret_metadata.传递它们来重新填充。BlockProcessingChannel.process将此队列中的程序方块图应用于传递给它的样本，使用first_sample和last_sample来确定何时移动到下一个块。

此结构允许渲染器的组件分离；音频样本可以独立于元数据块大小以块大小进行处理，同时保留样本精确的元数据处理，并且不会使渲染器复杂化，并考虑到具体的时间问题。

允许渲染器获取元数据块的决定将计时元数据的解释保留在渲染器中–如果元数据被推入渲染器中，则执行推送的组件必须知道何时需要下一个块，这取决于其中的计时信息。

此功能在core.renderer_common中实现。

6.4.1 实现的ProcessingBlock类型

三种常见的处理区块类型是：

FixedGains采用单个输入信道并应用n个增益，将输出相加为n个输出信道。

FixedMatrix接受N个输入信道，并应用NxM增益矩阵形成M个输出信道。

InterpGains采用单个输入信道，并应用n个线性插值增益，将输出相加为n个输出信道。提供了两个增益向量gains_start和gains_end，它们是在start_sample和end_sample时应用的增益。在样本s处应用于信道i的增益$g(i,s)$由下式给出：

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

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

6.5 定时元数据的通用解释

块开始和结束时间的确定在不同*typeDefinitions*的渲染器之间共享。对于TypeMetadata对象block，使用以下过程：

– 包含块的对象的开始和结束时间是从block.extra_data.object_start和block.extra_data.object_duration.确定的。如果object_start为None，则假定该对象在时间0开始。如果object_duration为None，则假定它扩展到无穷大。

– 块的开始和结束时间是由rtime和duration属性决定的：

- 如果rtime和duration不为None，则块开始时间为对象开始时间加rtime，块结束时间为块开始时间加duration。
- 如果rtime和duration为None，则假定块从对象的开始时间扩展到对象的结束时间。
- 其他rtime和duration系列被认为是**error**。–对于*audioChannelFormat*内的多个*audioBlockFormat*对象，应同时提供*rtime*和*duration*，而对于覆盖整个*audioObject*的单个块，不应提供*rtime*或*duration*。否则，行为是不确定的。

应检查时间是否一致。不允许在对象结束时间之后结束的块或按顺序重叠的块，并将其视为错误。错误条件意味着实现者必须考虑输入数据有问题。正确的做法是修复产生它的系统。在引用实现中，通过停止渲染过程并向用户报告错误来处理错误。其他实现可能根据其目标应用程序环境使用不同的错误处理策略。

这是在core.renderer_common.InterpretTimingMetadata中实现的。

6.6 TrackSpecs说明

渲染器的音频输入是通过多信道总线直接从输入文件读取的。RenderingItems形式的输入元数据包括TrackSpec对象，它是用于从总线提取信道的指令，包括将多个信道混合在一起的矩阵预处理。

每种TrackSpec类型的处理都是在core.track_processor中实现的。

给定TrackSpec，可以创建一个TrackProcessor对象，该对象有一个单独方法process(sample_rate, input_samples)，它将指定的处理应用到input_samples并返回单信道结果（在给定的采样率下）。

6.6.1 SilentTrackSpec

对于n个输入样本，SilentTrackSpec的process返回n个零值样本。

6.6.2 DirectTrackSpec

DirectTrackSpec track_spec的process返回在track_spec.track_index（用零基础数据）中的输入样本。

6.6.3 MixTrackSpec

MixTrackSpec track_spec的process返回在TrackProcessor上为track_spec.input_tracks中的每个子音轨调用进程的结果之和。

6.6.4 MatrixCoefficientTrackSpec

MatrixCoefficientTrackSpec track_spec将track_spec.coefficient中指定的矩阵process（表示单个矩阵*coefficient*元素的参数）应用于轨道规格。track_spec.input_track指定的单个信道。

如果track_spec.coefficient.gain不是None，则样本乘以gain。

如果track_spec.coefficient.delay不是None，则样本延迟n个样本，delaymsec，四舍五入到最近的样本（关系朝0断开）：

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

不支持某些参数。如果gainVar、delayVar、phaseVar或phase不为None，或delay为负，则会产生错误。

6.7 相对角

relative_angle(x,y)用于为y找到一个相等角度，大于或等于x。

relative_angle(x, y)返回$y' = y + 360n$，其中n是最小的整数，使得$y' \geq x$。

6.8 坐标变换

cart函数定义为根据第2.2节从极坐标位置转换到笛卡尔位置：

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

其中：

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

还定义了从笛卡尔位置提取方位角和仰角的逆变换：

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

函数local_coordinate_system生成一个旋转矩阵，该矩阵将{0,1,0}映射到给定的方位角和仰角：

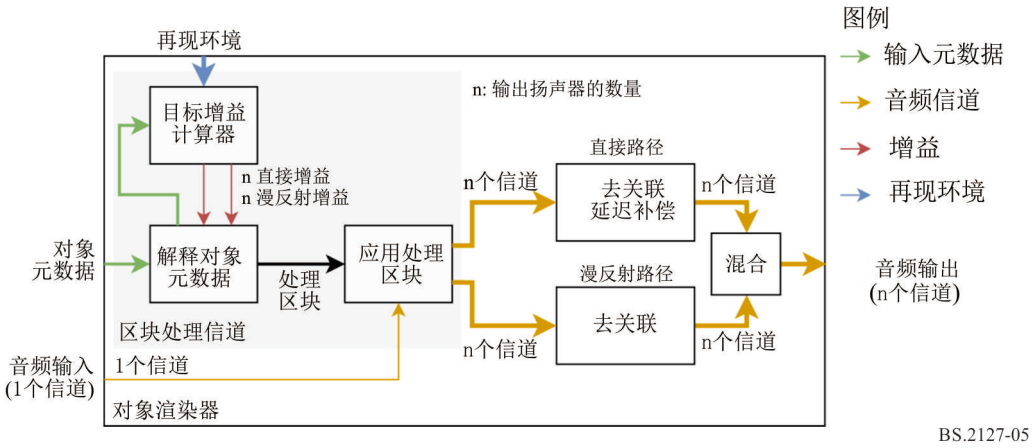
$$\texttt{local\_coordinate\_system}(\varphi, \theta) = \begin{bmatrix} \texttt{cart}(\varphi - 90, 0, 1) \\ \texttt{cart}(\varphi, \theta, 1) \\ \texttt{cart}(\varphi, \theta + 90, 1) \end{bmatrix}$$

7 使用typeDefinition==Objects的渲染项

7.1 结构

表5

对象渲染器的结构



*typeDefinition==Objects*的渲染器结构如图5所示。此图显示了应用于单个渲染项的处理；渲染多个项的行为就像此结构对每个项重复一样，输出混合在一起。

元数据以`ObjectRenderingItem`对象的形式进入渲染器，其中包含一个跟踪指数和一个`ObjectTypeMetadata`对象的源，这些对象表示标识轨道规格的有时间限制的渲染参数。

对于每个`ObjectTypeMetadata`对象，应用第7.2节中描述的方法；该方法解释计时元数据，并使用第7.3节中描述的增益计算器计算增益向量。这将产生`ProcessingBlock`对象，这些对象将有时间限制的信号处理操作应用于输入音频以产生直接和漫反射总线，每个总线包含每个扬声器一个信道。第6.4节描述了这种方法以及封装它的`BlockProcessingChannel`类。

漫反射总线通过每个信道的去相关滤波器组，直接总线延迟匹配，然后混合在一起形成输出。解联滤波器和延迟在7.4节中有描述。

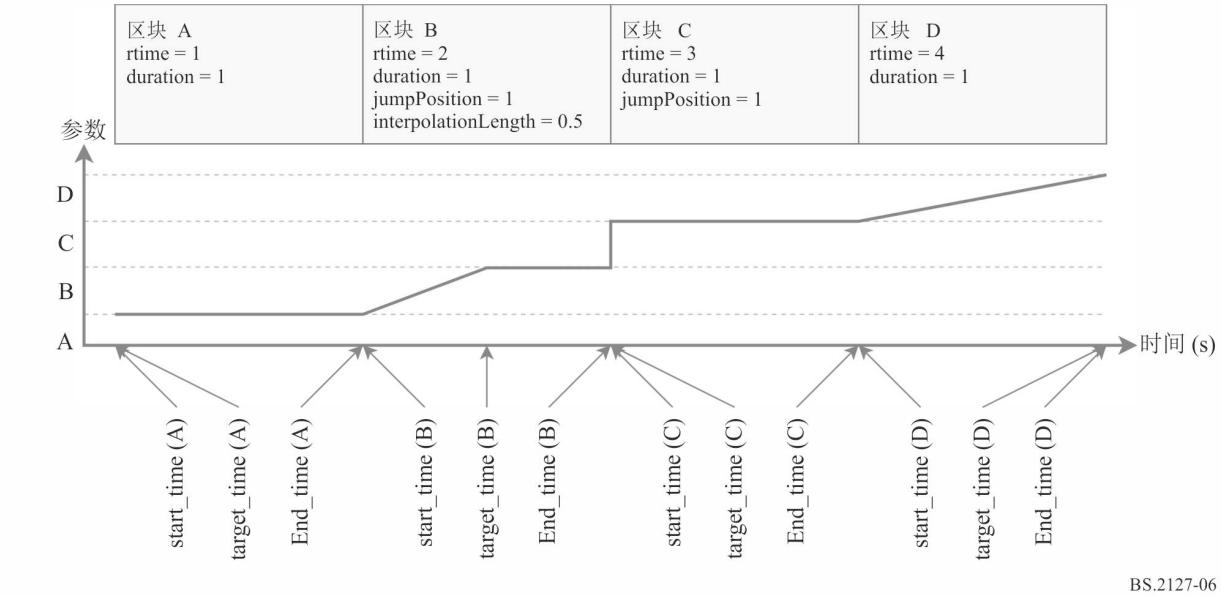
此结构在`core.objectbased.renderer.ObjectRenderer`中实现。

7.2 InterpretObjectMetadata

对象计时元数据在`InterpretObjectMetadata`类中解释，该类适合块处理信道结构。

图6

audioBlockFormats和解释的插值曲线示例



对于每个输入ObjectTypeMetadata，使用以下过程:

– 区块的开始和结束时间start_time和end_time根据第6.5条确定。

– 根据图6中相应的块所示的以下情况来确定该块中的插值结束的时间、target_time：

A

如果这是第一个块，或者前一个块的end_time小于当前块的start_time，则：

$$target\_time = start\_time$$

B

如果bf.jumpPosition.flag已设置，且bf.jumpPosition.interpolationLength不是None，则：

$$target\_time = start\_time + bf.jumpPosition.interpolationLength$$

C

如果bf.jumpPosition.flag已设置，且bf.jumpPosition.interpolationLength是None，则:

$$target\_time = start\_time$$

D

如果bf.jumpPosition.flag未设置，则在整个区块进行插值：

$$target\_time = end\_time$$

– 使用当前块的GainCalculator实例计算增益矢量interp_to。interp_to是为前一个块计算的增益矢量。

– 如果start_time < target_time，则创建一个InterpGainsProcessingBlock，该块从interp_from增益到interp_from增益再到start_time和target_time之间进行插值。

– 如果target_time < end_time，则创建一个FixedGains ProcessingBlock，该块在start_time和target_time之间应用interp_to。

这是在core.objectbased.renderer.InterpretObjectMetadata中实现的。

7.3 增益计算器

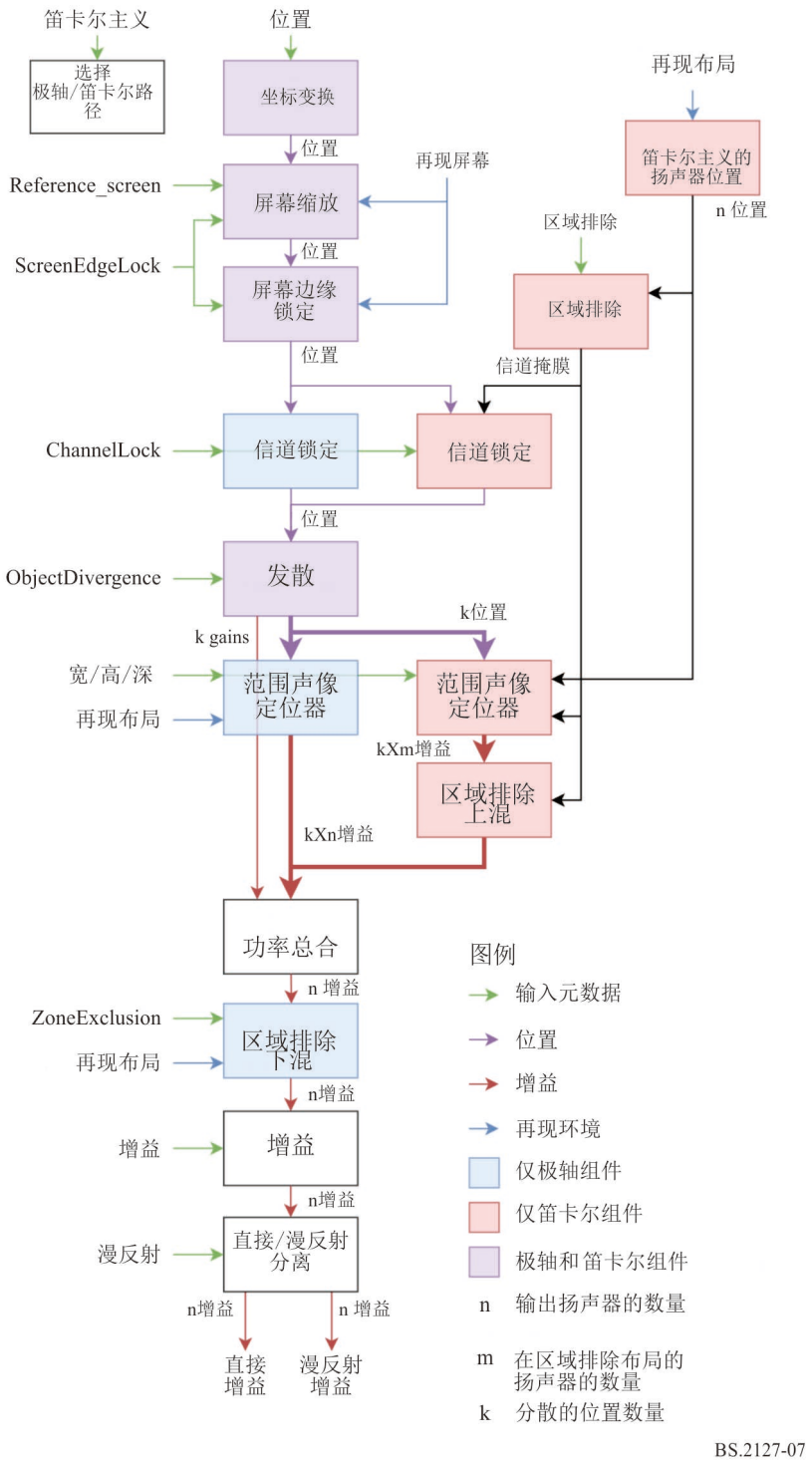
给定一个ObjectTypeMetadata对象，该对象计算直接和漫反射路径上每个扬声器的增益。这个组件的接口是：

```
structDirectDiffuseGains {
    vector<float> direct;
    vector<float> diffuse;
};

class GainCalc {
    GainCalc(Layout layout);

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

7.3.1 结构

图7

***typeDefintion==Objects*的增益计算器的结构**



此组件主要由本节中列出的子组件组成。图7示出了这些组件之间的信号流图。包含`block_format`属性`bf`的`ObjectTypeMetadata`的`otm`的行为如下：

– 将第7.3.2节中描述的坐标变换应用于bf.position以产生CartesianPosition对象position。

– 屏幕缩放采用第7.3.3节所述的方法，参数position, bf.screenRef, otm.extra_data.reference_screen和bf.cartesian，更新position。此组件用再现屏幕（layout.screen）和再现布局（layout）初始化。

– 使用第7.3.4节中描述的方法应用屏幕边缘锁定，参数为position, bf.position.screenEdgeLock and bf.cartesian，根据结果更新位置。此组件用再现屏幕(layout.screen)和再现布局(layout)初始化。

– 如果bf.cartesian，则：

  • 根据第7.3.9节的规定，确定layout.without_lfe中每个扬声器的异中心位置，从而产生allo_channel_positions数组。

  • 第7.3.5节中所述的区域排除算法适用于所有allo_channel_positions和bf.zone_exclusion，产生要排除的扬声器布尔掩码，excluded。

  • 在第7.3.6节所述的非中心配置中，参数position与bf.channelLock和excluded一起应用，更新position。

  否则：

  • 第7.3.6节中描述的以自我为中心的配置中的信道锁定与参数position和bf.channelLock一起应用，更新position。

– 使用第7.3.7节中描述的方法应用散度，参数为position, bf.objectDivergence和bf.cartesian。这将产生最多三个扩展源，增益和位置存储在diverged_gains和diverged_positions中。

– 如果bf.cartesian，则：

  • 第7.3.11节中所述的范围声像定位器应用于diverged_positions的每个p，参数channel_positions, p, bf.width, bf.height, bf.depth产生非排除扬声器数组的增益矢量。channel_positions是从allo_channel_positions[i]中选择的非排除信道位置列表，其中，excluded[i]为False。

  • 根据excluded，这些增益矢量上混，导致excluded[i]的每个扬声器i的增益为False，excluded[i]的每个扬声器i的增益为True，并且存储在gains_for_each_pos。

  否则：

  • 第7.3.8节中所述的范围扩展器应用于每个p的diverged_positions，参数p, bf.width, bf.height, bf.depth导致每个扬声器的增益矢量存储在gains_for_each_pos。

– gains_for_each_pos的增益都与一个由diverged_gains决定的功率混合在一起：

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

– 如果未设置bf.cartesian，则对gains和bf.zoneExclusion应用第7.3.12节所述的区域排除，从而产生新的gains向量。此组件使用layout.without_lfe初始化。

– gains通过添加值为0的LFE信道增益来扩展，以产生gains_full，在layout中每个扬声器有一个值。

– gains_full被分割成一个直接和扩散矢量，以控制直接和漫反射路径，具体取决于bf.diffuse参数。它们作为具有属性的DirectDiffuseGains返回：

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

7.3.1.1 讨论（资料性）

增益计算器的结构受以下两个原则的影响：

– 如果参数是稀疏的（即只使用少量可能的元数据字段），则需要保留对这些参数的明显解释。

– 当参数组合一起使用时，将选择为用户提供不同有用行为的最大可能性的选项。

例如：

– 信道锁定是作为位置修改来实现的–如果信道锁定是自己使用的（具有适当的*maxDistance*），那么由于点源声像定位器的行为，源将锁定到信道，然而，信道锁定也可以与范围参数一起使用，例如，产生围绕特定扬声器的扩展源。

– 漫反射度与范围无关–通过适当设置范围参数可以获得完全扩展的漫反射源，但这也允许使用小于完全范围的去相关过滤。

7.3.2 坐标变换

在core.objectbased.gain_calc.coord_trans中实现了一个简单的坐标变换，用于将输入位置转换为统一的笛卡尔坐标。它有以下签名:

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

position首先转换为笛卡尔向量**p**。

如果position是一个ObjectCartesianPosition，那么**p**的元素在返回之前被剪裁到范围$[-1,1]$：

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

否则**p**将未经修改返回。

对于实数，clip 定义为：

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

并简单地应用于向量中的每个元素：

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

7.3.3 屏幕缩放

屏幕缩放组件扭曲源位置，以补偿生产和再现环境之间屏幕几何结构的差异。这个组件的接口是:

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

使用的两个屏幕定义是：

参考屏幕

audioProgramme元素中列出的audioProgrammeReferenceScreen，如果没有提供，则使用默认的极坐标屏幕大小。这是元数据生产期间使用的屏幕几何图形。

再现屏幕

再现环境中的屏幕几何图形，在该环境中，渲染器的输出将被监听。

参考屏幕内的位置被扭曲，以便它们出现在再现屏幕中的相应位置。

7.3.3.1 内部屏幕表示

有关两个屏幕的信息可以以极坐标或笛卡尔坐标（PolarScreen 或 CartesianScreen 对象）提供。与对象源位置不同，两者之间没有明显的等价性，但是为了简化实现，需要一个可以表示两种屏幕类型的单一屏幕表示。这是 PolarEdges 结构的用途，它存储左右屏幕边缘的方位角，以及顶部和底部屏幕边缘的仰角：

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

一个 PolarEdges 对象是从给定的 PolarScreen 或 CartesianScreen 对象创建的，首先将屏幕转换为笛卡尔中心位置和两个矢量（沿x和z方向），这两个矢量定义屏幕的表面，然后查找每个边的方位角和仰角。

对于 PolarScreen screen，其中：

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

使用以下步骤：

– 中心位置是中心位置的简单笛卡尔变换：

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

– 计算笛卡尔宽度和高度:

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

– local_coordinate_system用于查找屏幕的x和z矢量：

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

对于CartesianScreen screen，其中：

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

使用以下步骤：

– 直接使用中心位置:

$$centre = screen.centrePosition$$

– 计算宽度和高度:

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

– 屏幕x和z向量定义如下：

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

对于这两种屏幕类型，PolarEdges 对象可以用：

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

7.3.3.2 位置补偿

在某些输出布局中，当cartesian==true，侦听器前面的垂直声像定位可能会扭曲。这是使用 core.screen_common.compensate_position 功能补偿的：

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

其中：

– φ_r是由θ的分段线性插值形成的，从：

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

至:

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

– φ'是由φ的分段线性插值形成的，从：

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

至:

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

7.3.3.3 方向扭曲

位置扭曲在 core.screen_scale.PolarScreenScaler.scale_az_el 中定义，它独立地扭曲方位角和仰角值。给定参考屏幕的 PolarEdges ref 和再现屏幕的 PolarEdges rep，其工作原理如下：

– 分段线性插值应用于方位角，从值映射

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

到

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

– 分段线性插值应用于仰角，从值映射

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

到

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

这是在 core.screen_scale.PolarScreenScaler.scale_position 中扭曲的，该位置将 scale_az_el 应用于笛卡尔向量的方位角和仰角分量，保持距离不变。

7.3.3.4 元数据解释

如果设置了 screenRef 并且提供了再现屏幕，则该位置将通过参考和再现屏幕的设置 PolarScreenScaler.scale_direction。否则，位置将原封不动地返回。

如果未设置 screenRef 或未提供再现屏幕，则未经修改返回位置。否则，行为取决于 cartesian 标志：

– 如果设置了cartesian坐标，则通过使用第10.1节中描述的转换应用极坐标缩放和补偿，从而产生一个新位置$\{x',y',z'\}$:

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

– 否则，scale_az_el将应用于位置的方位角和仰角分量。

7.3.4 屏幕边缘锁定

屏幕边缘锁定组件扭曲源位置，以便将源放置在屏幕的指示边缘上。它有以下接口：

```
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
  );
};
```

初始化时，该组件将 reproduction_screen 转换为 polar_edges 对象 PolarEdges，如第 7.3.3.1 节所述。

把 handle_az_el 独立修改方位角和仰角，从而生成新的方位角和仰角：

– 如果screen_edge_lock.horizontal为LEFT，则方位角设置为polar_edges.left_azimuth；如果为RIGHT，则设置为polar_edges.right_azimuth；否则方位角不变。

– 如果screen_edge_lock.vertical为TOP，然后将仰角设置为polar_edges.top_elevation；如果是BOTTOM，则仰角设置为polar_edges.bottom_elevation；否则仰角不变。

如果未提供 reproduction_screen，则不会发生位置修改。

处理在极坐标域中进行，因此必须首先转换笛卡尔位置。如果使用 handle_vector 方法而不是 handle_az_el 方法，则应用前后转换。

– 如果设置了cartesian坐标，则通过使用第10.1节中描述的转换应用极坐标缩放和补偿，从而产生一个新位置$\{x', y', z'\}$：

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

– 否则，handle_az_el将应用于位置的方位角和仰角组件。

该组件在 core.screen_edge_lock.ScreenEdgeLockHandler 中实现。

7.3.5 笛卡尔区域排除

笛卡尔区域排除算法从作为 channel_positions 的完整再现布局开始，并处理 ExclusionZone 对象以识别应移除哪些扬声器–这遵循了第 7.3.12.1 节中规定的算法。对于位于 ExclusionZone 对象指定的任何区域内的每个扬声器，将删除该扬声器，如果这导致将一行扬声器（共享相同的y和z坐标）减少为一个扬声器，然后移除行中的所有扬声器，以保持第 7.3.10 节中点源声像定位器所要求的基本特性。

如果应用区域排除的过程将导致删除所有扬声器，则不删除任何扬声器。

最后，建立了一个上混矩阵，该矩阵以单位增益将缩小布局中的信道映射到完整布局中的原始信道。

这在 core.allocentric.apply_zone_exclusion 中实现。

7.3.6 信道锁定

信道锁作为位置转换来实现。如果设置了 *channelLock*，并且扬声器在 *maxDistance* 中指定的范围内，则该位置将转换为扬声器最接近原始位置的位置。在没有分歧、范围、区域排除和扩散元数据的情况下，源将由选定的扬声器直接复制。

objectbased._gain_calc.ChannelLockHandlerBase 具有以下签名：

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

excluded 是一个信道排除掩码，用于指示应忽略哪些扬声器，并且只在以非中心路径中使用，因为只有在区域排除后才在那里执行信道锁定。

对于同心路径，ChannelLockHandlerBase 是在 core.objectbased._gain_calc.EgoChannelLockHandler 中配置的。

对于分配中心路径，ChannelLockHandlerBase 是在 core.objectbased._gain_calc.AlloChannelLockHandler 中配置的。

在同心模式下，所考虑的扬声器位置是 layout 中的归一化真实扬声器位置，而在非中心模式下，它们的位置则根据第 7.3.9 节中所述的 core.allocentric.positions_for_layout(layout)。

要应用信道锁元数据，请使用以下过程：

- 如果excluded不是None，则不要考虑扬声器，在以下步骤中，excluded[n] == True（n是第n个扬声器）。
- 如果channelLock为None，则返回原始position。
- 如果channelLock.maxDistance不是None，则计算每个扬声器位置和position之间的ℓ_2距离，并标识所有扬声器（在一定公差范围内），其中距离小于channelLock.maxDistance（尽可能短）。
- 如果没有找到扬声器，则返回position。

– 在一组可能的扬声器中，识别最接近position的扬声器。在自我为中心的配置中，使用position和每个扬声器之间的ℓ_2距离；在非自我为中心的配置中，使用position和每个扬声器之间的加权距离。加权距离计算如下：

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

其中：

$$w_x = \frac{1}{16}$$
$$w_y = 4$$
$$w_z = 32$$

– 如果没有唯一的最近扬声器（在一定的公差范围内），则从具有最高优先级的最近扬声器组中选择扬声器。扬声器的优先级排序是通过元组的字典比较确定的：

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

其中φ和θ是扬声器的实际方位角和仰角。较低的元组具有较高的优先级–具有较低绝对仰角的扬声器具有最高的优先级，其连接被仰角、绝对方位角和方位角打断。

– 返回所选扬声器的位置。

7.3.7 发散

发散是通过在原始源位置$\mathbf{p}_c$的左侧和右侧添加两个额外的源位置$\mathbf{p}_l$和$\mathbf{p}_r$来实现的。每个源位置都与增益值g_l、g_c和g_r相关联。

发散元数据在 core.objectbased.gain_calc.diverge 中解释，并带有以下签名：

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

此函数接受 3D 位置（在本例中为信道锁函数的输出），并应用 objectDivergence 差异元数据。生成三个源位置和相关的增益，每个增益都传递到范围声像定位器进行渲染。

这些增益和位置的计算如下所述。

7.3.7.1 增益计算

对于给定的 objectDivergence.value$x$，三个增益计算如下：

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

这满足以下要求:

– $\forall x,\ g_l + g_r + g_c = 1$

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

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

– $x = 1 \Longrightarrow g_l = g_r = 0.5 \wedge g_c = 0$

7.3.7.2 位置计算

产生的位置取决于块格式中的 cartesian 标志。如果设置了 azimuthRange 和 cartesian，或者设置了 positionRange 而未设置 cartesian，则会发出警告。

7.3.7.2.1 当cartesian == true时的行为

对于 position 值$\mathbf{p}$和 objectDivergence.positionRange 值x，中心位置只需沿x轴向左和向右移动，并剪切到$[-1,1]$：

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

clip定义见第 7.3.2 节。

7.3.7.2.2 当cartesian == false时的行为

位置根据给定的 objectDivergence.azimuthRange$a$计算，因此从听者的角度来看，左、右声源与中心的左、右各成a度，三个声源都在一条直线上。

这是通过定义三个以$+y$轴为中心的位置，在距离$d = \| \mathbf{p}_c \|_2$的情况下实现的，其中$\mathbf{p}_c$是原始源位置：

$$\begin{aligned} {}'_l &= \mathrm{cart}(a, 0, d) \\ P'_r &= \mathrm{cart}(-a, 0, d) \\ P'_c &= \mathrm{cart}(0, 0, d) \end{aligned}$$

然后，通过旋转矩阵$\mathbf{M}$围绕原始源方向旋转，旋转矩阵 M 的定义使得$\mathbf{p}_c{}'$映射到原始源位置$\mathbf{p}_c$：

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

7.3.8 极限值

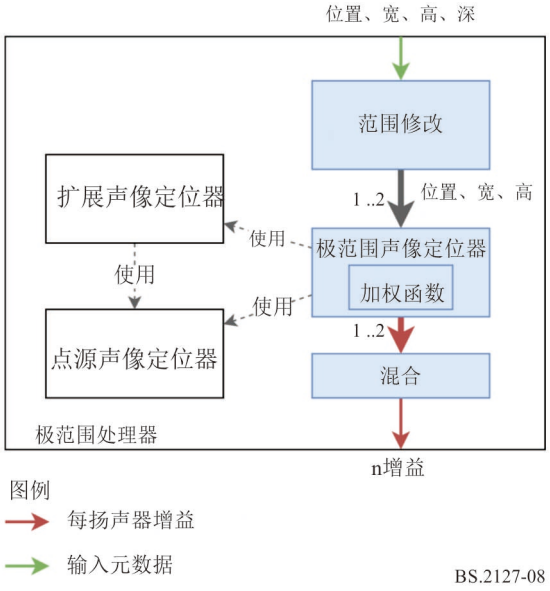
ADM 极范围参数在 core.objectbased.gain_calc.PolarExtentHandler 中处理；它使用下面描述的模块为给定的位置和范围参数生成增益向量。

这个类的接口是：

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

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

图8

范围处理器的结构



`PolarExtentHandler` 类的结构如图 8 所示。

在内部，该对象保存对第 7.3.8.2 节所述 `PolarExtentPanner` 的引用，该值用于计算增益向量。

对 `width`, `height` 和 `position` 参数进行复制和修改，以处理 `depth` 参数和 `position` 距离分量；这些参数通过极性范围声像定位器，为每个参数生成扬声器增益矢量，最后将这些增益矢量混合在一起。该程序在第 7.3.8.2 节中有说明。

极性范围渲染模式使用扩展声像定位器来生成扬声器增益，如下所述。

7.3.8.1 扩展声像定位器

渲染器中扩展源的形状是根据一个加权函数定义的，该加权函数给定一个三维方向，可以计算该方向的权重。这个权重可以被认为是一个给定的物体应该在给定的方向上被复制的量。例如，对于监听器前面的源，其宽度大于其高度，可以使用如图 10 所示的加权函数。

通过产生反映该加权函数的每扬声器增益，将这些增益应用于对象的单声道波形，并将去相关滤波应用于产生的声道，可以获得具有预定范围参数的扩展或漫反射声源的印象。

为了计算给定加权函数的增益向量，使用了 `SpreadingPanner` 类。

扩展声像定位器中使用的 1652 个虚拟源位置的确定方法如下：

对于-90°和 90°（均含）之间的每个仰角θ以 5°为步长，计算在该仰角处围绕圆均匀间隔的点数n，以在单位球体表面上实现近似均匀的密度：

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

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

然后，对于0到$n-1$（含）范围内的每个i，计算方位角 φ：

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

如此获得了笛卡尔坐标点(φ, θ, 1)。

这种类型的对象包含一组虚拟源位置和每个位置的扬声器增益矢量。

在启动过程中，点源声像定位器用于计算每个位置的增益矢量。

为了计算给定加权函数的增益矢量，将加权函数应用于虚拟源位置。得到的每虚拟源增益矢量乘以预先计算的扬声器增益矢量，以获得单个每扬声器增益矢量。然后对其进行功率归一化，得到最终的增益矢量。

这在 `core.objectbased.extent.SpreadingPanner` 中实现。

7.3.8.2 渲染极范围

在极模式下，用于计算 `position`, `width`, `height` 和 `depth` 参数的扬声器增益的步骤如下：

– `depth`参数被解释为两个方向相同但距离不同的扩展源。这两个距离是：

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

– 对于每个距离，用极性范围声像定位器从`position`计算增益向量$\mathbf{g'}_1$和$\mathbf{g'}_2$，以及由极性范围修改函数修改的`width`和`height`，如下所述。

– 将增益矢量混合在一起以产生输出增益矢量$\mathbf{g}$，其中$\mathbf{g}_i$是扬声器i的增益：

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

7.3.8.2.1 极范围修正函数

在给定距离参数的情况下，使用范围修改函数修改宽度和高度参数。

它具有以下属性：

– 当$\texttt{distance} = 0$，范围总是360°。

– 当$\texttt{distance} = 1$，使用原始范围。

– 当$\texttt{distance} > 1$，范围随着距离的增加而减小。

– 当$0 < \texttt{distance} < 1$，对于较小的范围，范围在$\texttt{distance} = 0$时变化更大。

`extent` 和 `distance` 的范围修改函数定义如下：

– 程度的程度线性地映射到沿x轴的范围，具有最小尺寸：

$$\mathrm{min\_size} = 0.2$$

$$\mathrm{size} = \mathrm{min\_size} + \frac{(1-\mathrm{min\_size})\times \mathrm{extent}}{360^\circ}$$

– 一个直角三角形，如果形成的话，相邻的边是距离，相反的边是距离。然后使用形成的角度来确定新范围；这是为距离1和distance计算的：

$$e_1 = 4 \times \frac{180}{\pi} \times \mathrm{atan2}(\mathrm{size}, 1)$$

$$e_d = 4 \times \frac{180}{\pi} \times \mathrm{atan2}(\mathrm{size}, \mathrm{distance})$$

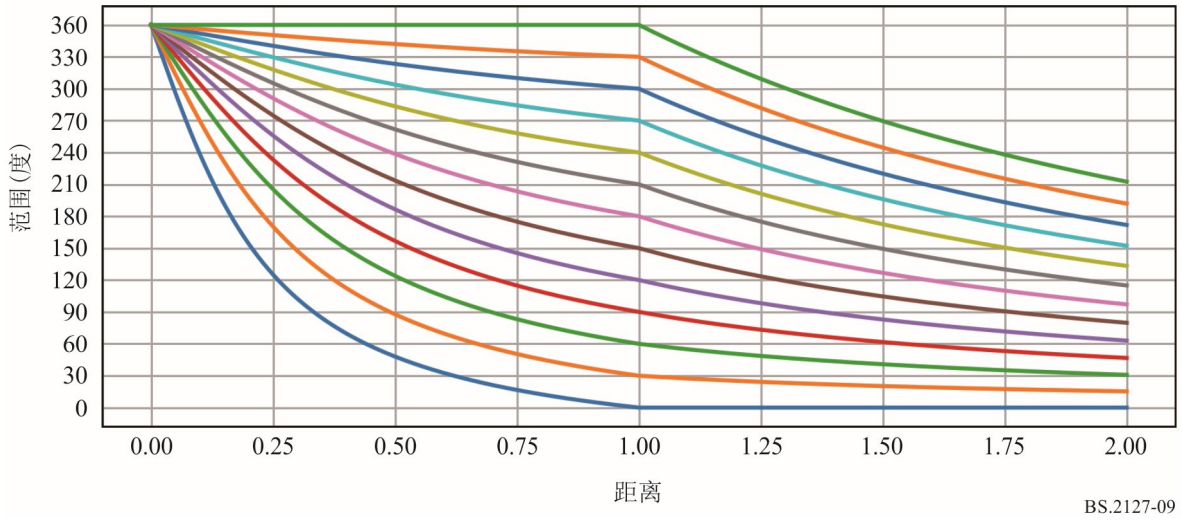
– 采用分段线性插值法将映射e_d恢复到原来的范围，且$e_d = e_1$:

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

这在 core.objectbased.gain_calc.PolarExtentHandler.extent_mod 中实现。范围修改函数的形状如图 9 所示。

图9

极性扩展源的扩展修改功能



注 – 每行显示给定输入范围的输出范围如何随距离变化。在$\mathrm{distance} = \mathbf{1}$的情况下不会修改范围，因此，例如，最下面的一行显示了对于 0 的输入范围，修改后的范围如何随距离变化。

7.3.8.2.2 极范围声像定位器

为了处理 ADM 中允许的全部位置和范围，在应用极性加权函数之前必须修改大小。使用以下步骤：

– 修改后的宽度和高度计算为$max\{width, 5^\circ\}$和$max\{height, 5^\circ\}$；它们与第7.3.8.1节中描述的扩展声像定位器和下面描述的极性加权函数一起使用，以产生扩展增益矢量g_s

– 该位置被传递到点源声像定位器以产生点源增益矢量$\mathbf{g}_\mathrm{p}$。

将这两个矢量混合在一起，得到矢量g，对于零宽度和零高度，只使用点源增益，如果宽度或高度大于5度，只使用扩展增益：

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

其中：

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

这需要支持小范围–这里扩展函数的非零部分必须足够大，以覆盖多个采样点，以便产生平滑增益，并且这要求最小的扩展量，其可能大于所需的量。

这是在 core.objectbased.extent.PolarExtentPanner.calc_pv_spread.中实现的。

7.3.8.2.3 极加权函数

极轴范围渲染的加权函数由 3D 笛卡尔矢量 position、角度 width 和 height（以度为单位）参数化。由于不使用位置的距离分量，因此可以将其视为方向。

加权函数如下：

– 计算旋转矩阵，将位置{0,1,0}（直接在侦听器前面）映射到源位置。这个旋转矩阵的形式是围绕{1,0,0}旋转，然后围绕{0,0,1}旋转。这是在core.objectbased.extent.calc_basis上实现的。

– 如果高度大于宽度，则翻转坐标系以简化计算，因为宽度为w、高度为h的源的加权函数应与宽度为h、高度为w的源的加权函数相同，并围绕源位置旋转90°。这是通过交换宽度和高度变量，以及交换旋转矩阵的x和z行来实现的。例如，参见图10和图11，它们具有相同的形状，但旋转90度（忽略由使用的投影引起的扭曲）。

– 在方位角-仰角空间中，最大加权圆$\mathtt{width} \times \mathtt{height}$矩形（体育场）内的近似加权函数现在为1，稍加修改：

  - 圆帽在笛卡尔空间中是圆形的，因为重量是根据两个矢量在其中心的角度计算的。当$\mathtt{width} = \mathtt{height}$,时，加权函数为圆形。
  - 在$\mathtt{width} > 180°$时，宽度增加，以便当宽度达到360°时，圆形部分完全重叠，形成一个“带”，其中加权函数对于相同仰角的所有位置具有相同的值。见图12和13。
  - 在加权函数的边缘添加淡入度；当与范围的角度距离达到10度时，权重从1降至0。

此功能是在 core.objectbased.extent.PolarExtentPanner.get_weight_func 中实现的。

图10

width = **90°**和height = **30°**的极性加权函数

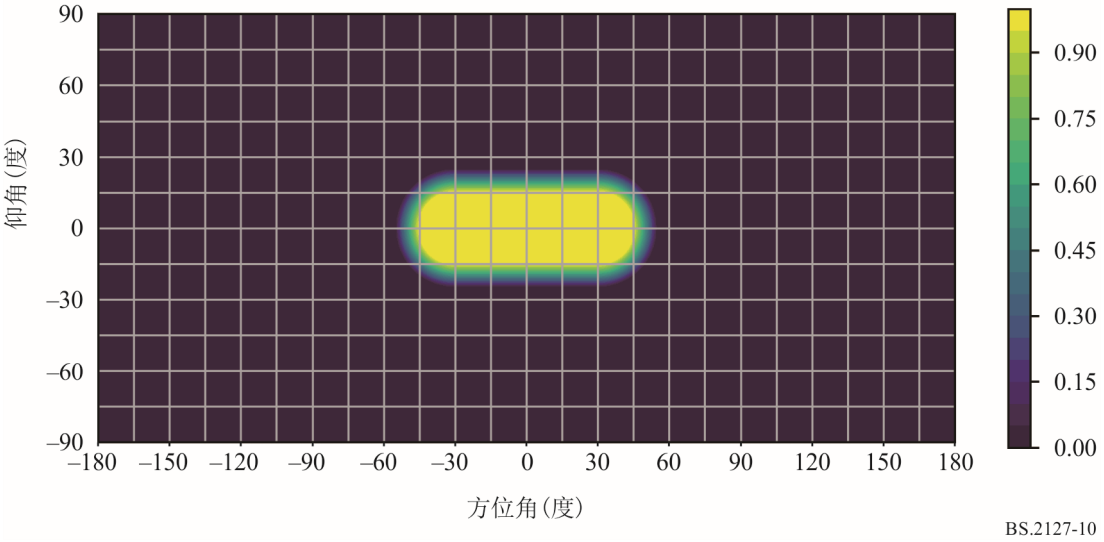


图11

width = **30°**和height = **90°**的极性加权函数

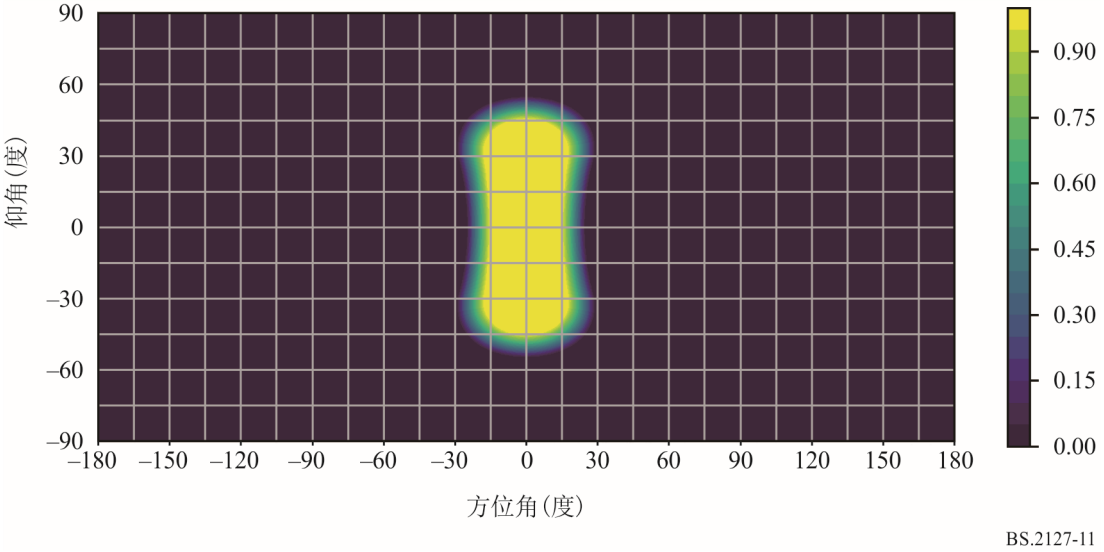


图12

width = **300°**和height = **30°**的极性加权函数

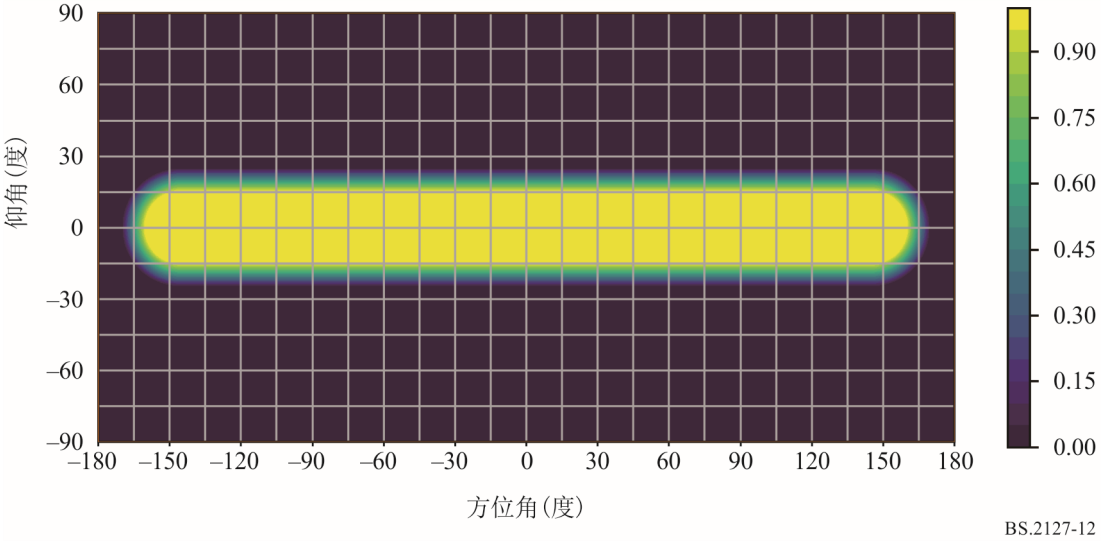
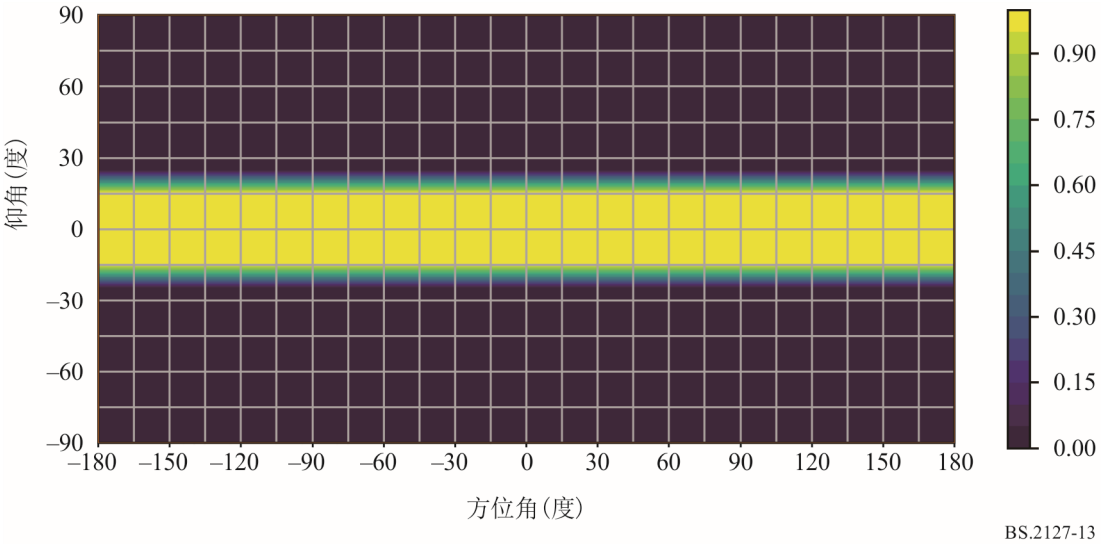


图13

width = **360°**和height = **30°**的极性加权函数



7.3.9 笛卡尔扬声器位置

若要使用第 7.3.10 节中规定的笛卡尔点源声像定位器，必须为布局中的每个扬声器找到笛卡尔位置。

这个组件的接口如下：

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

首先，与 `layout.name` 匹配的位置表见第 11.2 节。

对于 `layout.channels` 中的每个信道，输出 `CartesianPosition` 的 x、y 和 z 参数确定如下：

– 如果`channel.name`为`M+SC`或`M-SC`，则：

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

请注意，这假设point_polar_to_cart具有无限精度。实际上，必须修改这些位置，以便：

- $z = 0$
- 两个屏幕扬声器的y坐标必须相同。
- 两个屏幕扬声器的x坐标必须完全对称，大约为0。

– 否则，这些值将在标有channel.name的表中给出。

这是在 core.allocentric.实现的。

7.3.10 笛卡尔点源声像定位器

笛卡尔点声像定位算法是“双平衡”声像定位器概念的三维扩展，广泛应用于 5.1 和 7.1 声道环绕声制作。

声像定位器的输入包括一个对象的位置$[p_{ox}, p_{oy}, p_{oz}]$和N个输出扬声器的位置，都在笛卡尔坐标系中。设$[p_{sx}(j), p_{sy}(j), p_{sz}(j)]$表示扬声器$j$的位置。

关于扬声器布局，点源声像定位器要求满足以下条件，以便能够在房间的任何位置准确放置幻像声像：

– 扬声器必须在z维中组合成一个或多个离散平面。
– 每个平面上的扬声器必须在y维度上组合成一个或多个离散行。
– 在$-1 < y < 1$的任何一排（即不与房间前墙或后墙相交的任何一排），必须在$x = 1$和$x = -1$处安装扬声器。
– 每个扬声器位置必须位于房间立方体的表面，即地板、天花板或墙壁上。
– 符合这些条件的位置可按照第7.3.9节规定的程序找到。

对于给定的源位置，扬声器增益大约为：

– 查找源上下的扬声器层，并基于层和源的z位置计算这两个层的z增益。
– 在找到的每一层中，找到源位置前后的一行扬声器，并基于行和源的y位置计算每一行的y增益。
– 在找到的每一行中，在源位置的左侧和右侧找到一对扬声器，并基于扬声器和源的x位置计算每个扬声器的x增益。

最多可选择 8 个扬声器；每个扬声器的增益为$x \times y \times z$；其他扬声器的增益为零。

给出了该算法的精确规范，计算了每个扬声器j.的增益$g^{point}(j_x, j_y, j_z)$。注意到每个轴是可分离的，也有助于观察$g^{point}(x, y, z) = g^{point_x}(x) \times g^{point_y}(y) \times g^{point_z}(z)$，三个独立的增益可以作为算法的中间值。

```
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
  } elseif (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
  } elseif (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
    } elseif (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
  }
```

注意，最多 8 个扬声器将具有非零增益，并且扬声器增益的平方和始终为 1，因此声像定位操作是节能的。

这是在 core.point_source.AllocentricPanner 中实现的。

7.3.11 笛卡尔范围声像定位器

范围声像定位器的目的是计算输出扬声器布局中每个扬声器的增益系数，给定对象位置和对象范围。范围的意图是使对象看起来更大，这样当对象最大限度地填充房间时，当它被设置为 0 时，对象渲染为点对象。

为了实现这一点，范围声像定位器会考虑房间中许多虚拟源的网格。每个虚拟源发射扬声器的方式与使用点源声像定位器渲染的任何对象完全相同。当给定一个对象位置和对象范围时，范围声像定位器确定这些虚拟源中的哪些（以及多少）将作出贡献。

以下步骤对于计算具有范围的对象的增益是必要的。下面的一个小节将更详细地解释每个步骤。

1. 预缩放范围参数。
2. 计算所有虚拟源的点收益。
3. 将来自房间内虚拟资源的所有收益合并，以产生内部范围收益。
4. 合并房间边界上虚拟源的所有增益，以产生边界范围增益。
5. 合并内部和边界范围增益以产生最终范围增益。
6. 将对象的最终范围增益与点增益合并。

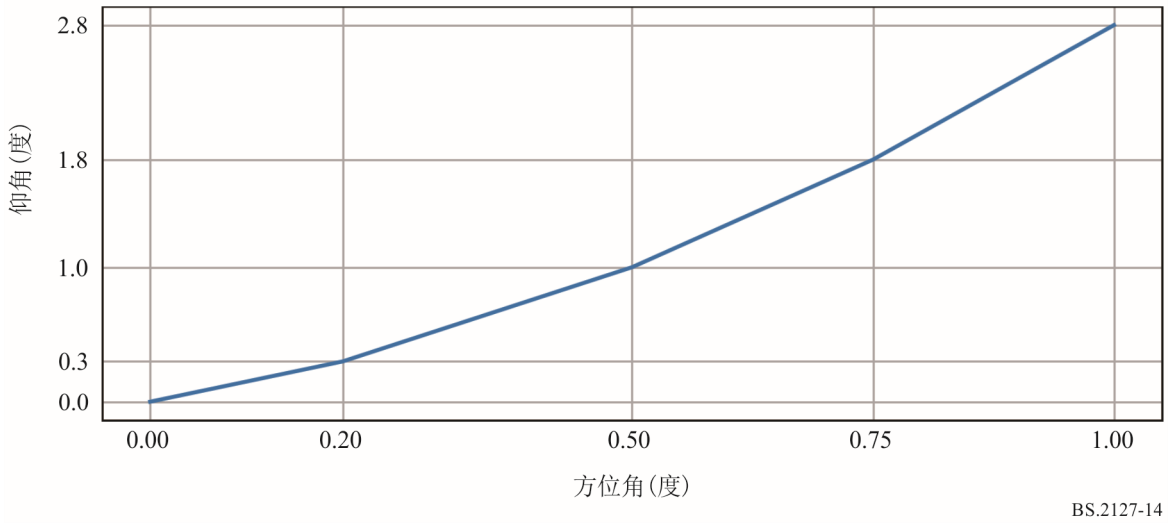
笛卡尔范围声像定位器是在 core.objectbased.allo_extent.get_gains 中实现的。

7.3.11.1 范围参数的预缩放

在计算任何增益之前，范围参数值被放大，以便源加权函数的行为更直观。用户暴露于 $s \in [0,1]$ 值，这些值被映射到算法使用的实际范围内 $[0,2.8]$。映射由值对 $(0,0), (0.2,0.3), (0.5,1.0), (0.75,1.8), (1,2.8)$ 定义的分段线性函数完成，如图 14 所示。最大值 2.8 确保当范围设置为最大(1.0)时，它真正占据整个房间。在下面的内容中，变量 $\hat{s}_x, \hat{s}_y, \hat{s}_z$ 表示应用映射后的输入范围值。

图14

ADM范围参数与算法内部范围值的分段线性映射



为了在范围的极端值下保持所需的行为，在$\hat{s}_x, \hat{s}_y, \hat{s}_z$上的最小值应用如下：

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

算法中始终使用这些限制值s_x, s_y, s_z。

7.3.11.2 计算虚拟源增益

虚拟源网格定义为$N_x \times N_y \times N_z$点的静态矩形均匀网格。网格跨越每个维度中的位置范围$[-1,1]$。密度的设置方式应包括典型布局中扬声器之间的一些信号源。经验测试表明，$N_x = N_y = N_z = 40$创建了一个适当的虚拟源网格[1]。符号(x_s, y_s, z_s)将用于表示虚拟源的可能坐标。根据第 7.3.10 节中描述的笛卡尔点源声像定位器算法，每个虚拟源为布局的每个扬声器$j = 1, \ldots, N_j$创建一组增益$g_j^{point}(x_s, y_s, z_s)$。注意，如果由于区域排除对象（见 7.3.5 节）而将任何扬声器排除在布局之外，则在计算增益时使用缩小的扬声器布局。

7.3.11.3 结合室内虚拟源增益

对象位置和范围$(x_o, y_o, z_o, s_x, s_y, s_z)$用于计算一组权重，这些权重确定每个虚拟源将对最终增益[2]贡献多少。每个虚拟源的权重表示为$w(x_s, y_s, z_s, x_o, y_o, z_o, s_x, s_y, s_z)$，并用于缩放每个虚拟源的点增益。加权后，将所有虚拟源收益相加，得出内部范围收益：

---

[1] 对于没有底层扬声器的扬声器布局，Z维中的虚拟源范围限制为[0,1]，N_z的建议值为20。

[2] 对于没有底层扬声器的扬声器布局，范围算法使用$z_o = max(p_{oz}, 0)$作为对象在Z维中的位置。否则$z_o = p_{oz}$。对于所有扬声器布局，范围算法使用与点源声响定位器相同的X和Y位置（即$y_o = p_{oy}, x_o = p_{ox}$）。

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

然而，范围算法结合虚拟源增益的方式取决于对象的范围。一般来说，这可以描述为：

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

范围相关的指数p控制扬声器增益的平滑度。它保证了目标在小s处的均匀生长和在大s处的正确能量分布。要计算p，首先按降序对$\{s_x, s_y, s_z\}$排序，并标记得到的有序空间坐标轴：$\{s_1, s_2, s_3\}$。然后，可以将三者结合起来，给出一个有效的范围：

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

对于具有单个扬声器平面的布局，例如 0+5+0 或如果区域排除导致布局缩减为单个平面，则首先按降序排序$\{s_x, s_y\}$，并将生成的有序 duo 标记为：$\{s_1, s_2\}$，给出：

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

对于 0+2+0 布局（立体声），或者如果区域排除将扬声器集减少到一行，则$s_{eff} = s_x$。

有效范围用于计算分段定义的指数：

$$p = \begin{cases} 6 & s_{eff} \le 0.5 \\ 6 - 4 \times \frac{s_{eff} - 0.5}{s_{max} - 0.5} & otherwise \end{cases}$$

其中$s_{max} = 2.8$，当s在其最大值时，$p = 2$。

加权函数也可以单独处理每个轴，并且如果使用可分离的加权函数，则整个范围计算简化：

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

所选函数看起来像是圆和正方形（或三维中的球体和立方体）之间的某种东西：

$$\begin{aligned} w_x(p, o, s) = w_y(p, o, s) &= 10^{-\min\left(\left[\frac{3}{2}\left(\frac{p-o}{2s}\right)\right]^4, 6.5\right)} \\ w_z(p, o, s) &= 10^{-\min\left(\left[\frac{3}{2}\left(\frac{p-o}{s}\right)\right]^4, 6.5\right)} \times \cos\left(s\frac{3\pi}{7}\right) \end{aligned}$$

这意味着g_j^{inside}可以简化为

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

其中：

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

注意，对于仅限于单个扬声器平面的布局，$f_j^z(z_o, s_z) = 1$,；对于单个扬声器行，$f_j^z(z_o, s_z) = f_j^y(y_o, s_y) = 1$。

此外，非常小的值$f_j(c, s)$）$(10^{-6.5})$被舍入为零，以避免实现中的浮点下溢。

归一化步骤应用于g_j^{inside}:

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

其中$tol = 10^{-5}$。

7.3.11.4 合并边界增益

另一个修改是，出于美观的原因，重要的是要有一个模式，没有相反的扬声器发射。这是通过使用仅位于边界上的虚拟源来实现的。特殊情况下处理某些扬声器布局；

- $dim = 1$适用于区域排除后只有一行扬声器的布局（例如.0+2+0），
- $dim = 2$适用于区域排除后只有一个扬声器平面的布局（例如.0+5+0），
- $dim = 4$适用于应用区域排除后具有两个以上不同高度扬声器平面的布局（例如3+7+0和9+10+3），以及
- $dim = 3$适用于其他情况。

边界增益是：

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

其中：

$$
\begin{aligned}
b_j^{floor}(z_o, s_z) &= \begin{cases} \left[g_j^{point}(z_s = -1.0)w(-1.0, z_o, s_z)\right]^p & dim = 4 \\ 0 & \text{otherwise} \end{cases} \\
b_j^{ceil}(z_o, s_z) &= \begin{cases} \left[g_j^{point}(z_s = 1.0)w(1.0, z_o, s_z)\right]^p & dim \geq 3 \\ 0 & \text{otherwise} \end{cases} \\
b_j^{left}(x_o, s_x) &= \left[g_j^{point}(x_s = -1.0)w(-1.0, x_o, s_x)\right]^p \\
b_j^{right}(x_o, s_x) &= \left[g_j^{point}(x_s = 1.0)w(1.0, x_o, s_x)\right]^p \\
b_j^{front}(y_o, s_y) &= \begin{cases} \left[g_j^{point}(y_s = 1.0)w(1.0, y_o, s_y)\right]^p & dim > 1 \\ 0 & \text{otherwise} \end{cases} \\
b_j^{back}(y_o, s_y) &= \begin{cases} \left[g_j^{point}(y_s = -1.0)w(-1.0, y_o, s_y)\right]^p & dim > 1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

7.3.11.5 结合内部和边界增益

现在需要将边界增益与内部增益相结合，因此为房间内的所有虚拟源引入淡出因子，淡出量=“房间外对象的分数”。

结果是：

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

其中：

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

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

且$h(c, s)$为单维的淡出函数。

$$h(c, s) = \begin{cases} \left[\frac{\max(2s, 0.4)^3}{0.16 \times 2s}\right]^{\frac{1}{3}} & d_{bound} \geq s \wedge d_{bound} \geq 0.4 \\ \left[\frac{d_{bound}}{2}\left(\frac{d_{bound}}{0.4}\right)^2\right]^{\frac{1}{3}} & \text{otherwise} \end{cases}$$

当扩展对象的一部分开始移出房间时，对象内部的所有虚拟源都开始消失，除了边界处的那些。当一个对象到达一个边界时，只有边界增益对范围增益起作用。d_{bound}是到边界的最小距离。

归一化步骤应用于g_j^{extent}

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

7.3.11.6 结合范围增益和点增益

然后，范围增益与点增益相结合，并将它们之间的交叉淡出作为范围的函数：

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

其中：

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

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

且$s_{fade} = 0.2$。

这确保了对象的平滑平移和平滑增长，提供了在最小和最大可能范围之间的良好过渡。最后对增益进行归一化处理：

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

7.3.12 极区域排除

通过将增益计算器中早先产生的扬声器增益向量进行下混合，以避免将输出发送到排除区域中的扬声器。这可以分为两个部分：在第 7.3.12.1 节中，确定哪些扬声器在排除区域内，在第 7.3.12.2 节中，计算从排除的扬声器出去的下行混音。

排除扬声器的选择和下混矩阵的计算都只考虑扬声器的标称位置，因此扬声器位置的微小变化不会影响区域排除的行为。

7.3.12.1 选择排除的扬声器

扬声器的选择是通过处理 `ExclusionZone` 对象的列表来实现的，该列表为每个扬声器生成一个布尔标志，如果扬声器在任何排除区域内，则该布尔标志为真，因此应排除在外。

对于 `CartesianZone` 对象，以下表达式用于确定扬声器是否在该区域内，其中$\{x, y, z\}$是扬声器的标称位置，由半径为 1 的极坐标转换而来：

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

其中$\epsilon = 10^{-6}$是在极坐标和笛卡尔坐标之间转换时允许舍入误差的安全裕度。

对于PolarZone对象，以下表达式用于确定扬声器是否在该区域内，其中φ和θ表示扬声器的标称方位角和仰角。

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

IAR是指inside_angle_range函数；见第6.2节。

扬声器的仰角必须在允许的范围内，而方位角只有在绝对仰角小于90度时才必须在允许的范围内。

这是在 core.objectbased.gain_calc.ZoneExclusionHandler.get_excluded 中实现的。

7.3.12.2 排除扬声器的下混

一旦确定了区域内的扬声器，就设计了一个下混矩阵，将增益从这些扬声器中分离出来。

区域排除声像定位器与布局中输出扬声器组列表中的每个扬声器相关联。下混矩阵使得从排除的扬声器获得的增益被路由到第一组中的所有不排除的扬声器，其中存在不排除的扬声器。此功能将在接下来的两节中进行更详细的描述。

例如，表 3 显示了 4+5+0 中的扬声器组。第一行显示，如果不包括 M+030，则该扬声器的输出将被路由到 M+000，除非不包括 M+000，在这种情况下，它将被路由到 M−030 等直到 U−110。

一个更复杂的例子是 M+000。如果排除了这一点，则该频道将在{M + 030, M − 030}中的非排除扬声器之间被分割，除非这两个扬声器都被排除在外，在这种情况下，它将被路由到{M + 110, M − 110}中的非排除扬声器等。

表3

4+5+0的扬声器关联示例

输入	输出组
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}

此功能是在 core.objectbased.zone.ZoneExclusionDownmix 和 core.objectbased.gain_calc.ZoneExclusionHandler 中实现的。

7.3.12.2.1 扬声器组的测定

在初始化过程中，将确定每个扬声器的输出扬声器组。

对于每个输入扬声器，每个输出扬声器被分配一个称为 *key* 的浮点数元组。然后，输出组由按键排序的输出扬声器组成，并用相似的键收集成组。因此，排序和分组主要由键函数定义。

输入和输出扬声器的键由四个键组成：

– 整数层优先级，如果两个扬声器都在同一层上，则为零，并且随着输入层和输出层的分离而增加，它更倾向于在较低层之前从较高层选择扬声器。图层优先级从表4中得出。

– 整数前/后优先级，如果输入和输出扬声器都在侦听器的前面、侧面或后面，则该优先级较低。给定输入和输出扬声器在转换为笛卡尔$\mathrm{y_i}$和$\mathrm{y_o}$后极性标称位置的y分量，计算如下：

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

– 两个扬声器的标称位置之间的矢量距离，以便选择较小的运动。

– 两个扬声器之间标称y坐标的绝对差，以便分割围绕yz或xz平面不对称的组。

表4

两个扬声器之间的层优先级值

输入层	底部	中间	上边	顶部
底部	0	1	2	3
中间	3	0	1	2
上边	3	2	0	1
顶部	3	2	1	0

7.3.12.2.2 区域排除的应用

一组排除的扬声器E的下混矩阵计算如下：

– 对于N个扬声器，从$\mathrm{N} \times \mathrm{N}$下混矩阵D开始，每个元素初始化为0。

– 对于每个输入扬声器i，考虑分组表i行中的每组候选扬声器索引C。

- 如果组中的所有扬声器都位于忽略的扬声器组（即$C \subseteq E$）中，请转到下一组。
- 否则，对于$C \setminus E$中的每个j（组中未排除的一组扬声器），设置：

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

然后转到下一个扬声器。

如果排除了所有扬声器，则将**D**设置为标识矩阵。

然后将**D**应用于输入增益矢量**G**以产生**G′**，通过：

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

7.4 去相关滤波器

当渲染 *diffuse* 参数大于 0 的对象时，使用对象渲染器的漫反射路径，每个扬声器输出有一个去相关滤波器。

使用的滤波器是$N = 512$采样长随机相位全通 FIR 滤波器。给定输出的筛选器生成如下：

– 使用MT19937伪随机数产生器生成长度为$\frac{\mathrm{N}}{2} - 1$的值在[0,1)范围内的伪随机向量r，该伪随机数产生器与信道名称的索引一起播种在布局中所有信道名称的排序列表中。

– 长度$\frac{\mathrm{N}}{2} + 1$的相位矢量p定义为：

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

– 对应的频率向量 x被定义为$\mathrm{x_n} = \exp(\mathrm{ip_n})$。

– 对 x中的非负频率分量进行逆实值傅里叶变换（`irfft`函数），得到时域滤波器。

这是在 `core.objectbased.decorrelate.design_decorrelators` 中实现的。

这些滤波器引入的延迟与直接路径中的$\frac{(N-1)}{2}$采样延迟相匹配。

8 使用typeDefinition==DirectSpeakers的渲染项

若要渲染具有 *typeDefintion==DirectSpeakers* 的 *audioChannelFormats*，则将其路由到匹配的扬声器。如果这是不可能的，PSP 将作为一个备份。

基本算法如下：

1. 对于使用通用定义指定的输入，该*audioPackFormats*描述了ITU-R BS.2051-2建议书中指定的布局，则适用根据第8.1节的映射规则。
2. 确定元数据是否引用LFE信道（见第8.2节）。如果是，则只考虑LFE输出，如果不是，则只考虑非LFE输出。
3. 如果任何*speakerLabels*与扬声器匹配（参见第8.3节），则该频道将被路由到匹配的第一个扬声器。如果没有匹配的*speakerLabel*，请继续下一步。
4. 如果指定`screenEdgeLock`锁定，则标称位置将移动到屏幕的水平和/或垂直边缘。最小和最大边界保持不变（参见第8.4节）。
5. 如果任何扬声器的标称位置在指定的位置范围内（参见第8.5节），则将信道路由至最接近指定标称位置的扬声器。使用的扬声器位置由第8.5节中的位置类型决定。如果边界内没有扬声器，或距离标称位置最近的扬声器不唯一，请继续下一步。
6. 如果元数据引用LFE路由到`LFE1`的信道（如果存在），或者丢弃它。如果元数据引用非LFE信道，则使用与用于定义其位置的坐标类型相对应的PSP将信道在其标称位置渲染。

以下小节将更详细地描述各个步骤。

这是在 `core.direct_speakers.panner.DirectSpeakersPanner` 中实现的。

8.1 映射规则

– 如果type_metadata.audioPackFormats中列出的最后一个*audioPackFormat*不是通用定义包格式（即，它是在输入元数据中指定的，不是从通用定义文件中读取的），不应用映射规则。

– 查找表15中type_metadata.audioPackFormats中列出的最后一个*audioPackFormat*的ID，以确定input_layout。如果没有列出，就不要应用映射规则。

– 尝试依次应用表16中列出的每个规则。如果应用了任何规则，则使用列出的第一个匹配规则的增益来再现该信道。如果没有匹配的规则，请继续下一步。如果满足所有这些条件，则规则匹配：

  • rule.speakerLabel等于应用第8.3节所述归一化后的第一个（且仅限于）*speakerLabel*。

  • input_layout（如上所述）列在rule.input_layouts中，若已列出。

  • 输出扬声器布局的名称，layout.name，列在rule.output_layouts中（如果已列出）。

  • 所有列在rule.gains中的信道名称存于layout.channel_names。

8.2 LFE测定

如果音频信道格式中的频率元素具有≤ 200 Hz的低通（见第 6.3 节），或者如果存在指向 LFE 信道的 *speakerLabel*（在应用了下面描述的匹配过程之后，LFE1 或 LFE2），则信道被视为 LFE 信道。

8.3 扬声器标签匹配

speakerLabels 的匹配仅适用于 ITU-R BS.2051-2 建议书（例如 M+030）和 ITU-R BS.2094-1 建议书（例如 urn:itu:bs:2051:0:speaker:M+030）中指定的 ADM 通用定义文件中使用的 URNs。ITU-R BS.2051-2 建议书规定了 LFE1 和 LFE2 的标签，当 ADM 文件使用以下 *speakerLabels* 时，采用了一些替换：

– LFE → LFE1

– LFEL → LFE1

– LFER → LFE2

8.4 屏幕边缘锁定

typedefinition==DirectSpeakers 的 *screenEdgeLock* 实现重用了用于 *typeDefintion==Objects* 的 ScreenEdgeLockHandler，如 7.3.4 节所述。它只用于转换标称位置；最小和最大界限将保持不变。

这意味着，如果指定了界限，那么它们将被解释为绝对界限，而与屏幕位置无关；源将仅锁定到原始指定范围内的信道。如果未指定界限，则点源声像定位器行为将被激活，从而导致源锁定到屏幕边缘，而不管是否有扬声器。建议不要同时使用 *screenEdgeLock* 和坐标界限。

8.5 界限匹配

规定的最小或最大界限将允许范围从标称位置扩大。如果未指定最小或最大界限，则将其设置为标称坐标。如果所有的坐标都在指定的 *bounds* 内，则扬声器匹配。除了一个例外，在极点（例如 T+000）有极坐标的扬声器匹配任何方位范围，因为它们有一个不确定的方位角。

具有标称极位置speaker的扬声器与极坐标中指定的界限匹配，如果

$$\begin{pmatrix} \text{IAR}(\texttt{speaker.azimuth}, \texttt{azimuth.min}, \texttt{azimuth.max}, \epsilon) \\ \vee \quad |\texttt{speaker.elevation}| \geq 90° - \epsilon \end{pmatrix}$$
$$\wedge \quad \texttt{elevation.min} - \epsilon \leq \texttt{speaker.elevation} \leq \texttt{elevation.max} + \epsilon$$
$$\wedge \quad \texttt{distance.min} - \epsilon \leq \texttt{speaker.distance} \leq \texttt{distance.max} + \epsilon$$

其中IAR是 inside_angle_range 函数（见第 6.2 节），且$\epsilon = 10^{-5}$是允许舍入误差的安全裕度。

使用第7.3.9节所述将笛卡尔位置speaker转换为笛卡尔坐标的扬声器与使用笛卡尔坐标指定的界限匹配，如果

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

则正确。

9 使用typeDefinition==HOA的渲染项

9.1 支持的HOA 格式

9.1.1 HOA等级

ITU-R BS.2076-1 建议书所定义的 HOA 信号可以渲染到 50 级（参见下面的详细信息）。在 ADM 中，HOA 信道通过相应的 HOA 类型子元素以其 *order* 和 *degree* 分别表示。因此，全 3D HOA 场景（由每一等级l和m组成，直至给定等级L），2D HOA 场景（由每一个 HOA 元素组成，以致$|m| = l$直至给定等级L），而且混合等级的 HOA 场景可以渲染。

但是，如果两个HOA信号具有相同的等级，则会引发异常，并且不会渲染信号。

9.1.2 归一化

HOA 信号归一化通过*归一化* HOA 类型子元素显示。所有三种可能的归一化（N3D、SN3D 和 FuMa）由此渲染器支持。ADM 中，HOA 归一化分别单独指定为每个 HOA 信号，因此，理论上可以定义 HOA 场景，其中不同的信号使用不同的正规化。但是，此渲染器不支持这种情况：*audioBlockFormat* 中的所有 HOA 信道必须共享相同的归一化。最后，请注意，FuMa 规范化仅支持 3 阶。

9.2 不支持的子元素

以下三个 HOA 类型的子元素目前没有在渲染器中解释：

– *nfcRefDist*，表示扬声器的参考距离。近场补偿（NFC）效应补偿扬声器参考距离和扬声器位于回放布局中的距离之间的不匹配，在该渲染器中未实现。在HOA渲染中实现这种效果显著地增加渲染器的计算复杂度，同时对听众对音频内容的感知具有相对较小的影响。

– *screenRef*，指示HOA组件是否与屏幕相关。在HOA语境中，此子元素的预期用途不明确；因此在渲染时不考虑它。

– *equation*，它是用来替换*order*和*degree*子元素的。当前的ADM标准没有提供有关用于指定数学公式的格式的精确规则。因此，无法可靠地支持此子元素。

注意，与 *normalization* 子元素类似，*audioBlockFormat* 中的所有 HOA 信道必须共享要渲染的相同 n*fcRefDist* 和 *screenRef* 值。

9.3 扬声器上HOA信号的渲染

图15

HOA渲染流程图

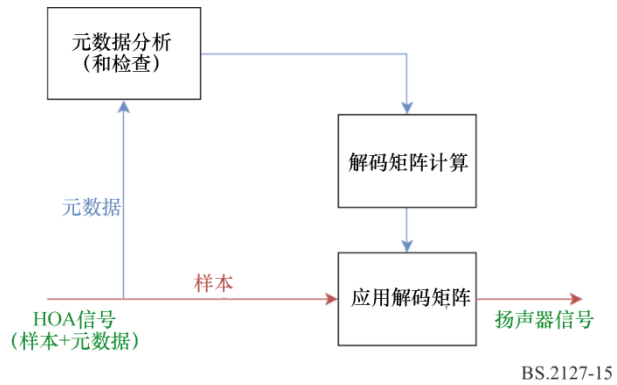


图 15 总结了通过扬声器渲染 HOA 信号的过程。首先，分析 ADM 元数据，以确定 HOA 对象的格式，并检查信号是否可以清晰地渲染。具体来说，如上所述，*audioBlockFormat* 中的所有 HOA 信道必须共享相同的 *normalization*、*nfcRefDist* 和 *screenRef* 子元素值。然后，计算扬声器解码矩阵并将其应用于 HOA 信号。这由以下方程式表示：

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

其中：

$\mathbf{S}_{\mathrm{spk}}$ 扬声器信号矩阵，维度$N_{\mathrm{spk}} \times N_{\mathrm{samp}}$

$\mathbf{S}_{\mathrm{HOA}}$ HOA信号矩阵，维度$N_{\mathrm{HOA}} \times N_{\mathrm{samp}}$

$\mathbf{D}$ 实值矩阵，维度$N_{\mathrm{spk}} \times N_{\mathrm{HOA}}$且被称为HOA解码矩阵

N_{HOA}, N_{spk}和N_{samp}分别表示 HOA 信号、扬声器信号和采样次数。

本节表达了 ACN 信道顺序中的解码矩阵计算，但是所使用的信道分配是在 *audioBlockFormat* 中的 *order* 参数中具体规定的。

解码矩阵通过使用第 6.4 节所述的块处理信道结构来应用。具体来说，对于每个传入的 `HOATypeMetadata` 对象，生成单个 `FixedMatrix` 处理块，该处理块在第 6.5 节确定的时间之间应用解码矩阵。

9.3.1 HOA解码矩阵计算

渲染器实现了 AllRAD HOA 解码技术[1]。该方法在不规则扬声器布局（如 ITU-R BS.2051-2 建议书中所述）上提供稳健的 HOA 解码。解码矩阵的计算在 `core.scenebased.design.HOADecoderDesign` 中完成。

从概念上讲，AllRAD 解码方法相当于:

1. 将HOA信号解码到均匀分布在球体上的虚拟扬声器网格，以及
2. 在实际扬声器上平移虚拟扬声器信号。

从数学上讲，这可以表示为：

$$\mathbf{D}' = \nu\,\mathbf{G}\,\mathbf{D}_{\mathrm{virt}}$$
$$\mathbf{D} = \mathbf{D}'\mathrm{diag}(\mathbf{n}^{-1})$$

其中$\mathbf{D}'$表示 N3D 归一化的 HOA 解码矩阵，$\mathbf{G}$是平移增益矩阵，$\mathbf{D}_{\mathrm{virt}}$是虚拟扬声器解码矩阵，$\nu$是能量归一化因素。$\mathbf{D}$是在将 HOA 归一化向量$\mathbf{n}$应用到$\mathbf{D}'$以应用所需的归一化后完成的解码矩阵。

9.3.1.1 虚拟扬声器位置

为了便于解码矩阵的计算，虚拟扬声器的角位置必须尽可能均匀地分布在球体上。此外，根据经验，虚拟扬声器的位置应该是 HOA 信号的两倍左右。

在该渲染器中，虚拟扬声器位置构成 5200 点 *spherical-T* 设计，这使得它非常适合解码高达 50 阶的 HOA 信号。

9.3.1.2 虚拟扬声器解码矩阵的计算

为了计算虚拟扬声器的解码矩阵，首先计算虚拟扬声器的 HOA 系数矩阵$\mathbf{Y}_{\mathrm{virt}}$。该矩阵由以下公式给出：

$$\mathbf{Y}_{\mathrm{virt}} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N_{\mathrm{virt}}}]$$
$$\mathbf{y}_n = [Y_0^0(\theta_n, \phi_n), Y_1^{-1}(\theta_n, \phi_n), \ldots]^{\mathrm{T}}$$

式中(θ_n, ϕ_n)表示第n个虚拟扬声器的仰角和方位角（使用ITU-R BS.2076-1建议书中定义的HOA坐标系和符号），Y_l^m表示N3D归一化的实值阶l和阶m球谐函数。注意，每个$Y_l^m(\theta, \phi)$项的值取决于每个HOA信道的*order*和*degree*子元素。

然后将虚扬声器HOA解码矩阵计算为$\mathbf{Y}_{\mathrm{virt}}$的转置：

$$\mathbf{D}_{\mathrm{virt}} = N_{\mathrm{samp}}^{-1}\mathbf{Y}_{\mathrm{virt}}^{\mathrm{T}}$$

对于虚拟扬声器位置的选择和N3D归一化，这相当于采用$\mathbf{Y}_{\mathrm{virt}}$.的转置。

9.3.1.3 平移增益矩阵的计算

在AllRAD HOA解码方法中，通常采用VBAP平移来计算平移增益矩阵。在这个渲染器的实现中，用于计算平移增益的方法只是为平移点源对象（core.point_source）提供的方法。

9.3.1.4 能量归一化

将HOA解码矩阵归一化，使得在HOA场景由单点源组成的情况下，对于球体上的每个可能的源位置，扬声器信号的总功率平均等于源信号的总功率。

从数学上讲，归一化因子ν计算如下:

$$\nu = \frac{\sqrt{N_{\mathrm{virt}}}}{\left\| \mathbf{G}\,\mathbf{D}_{\mathrm{virt}}\,\mathbf{Y}_{\mathrm{virt}} \right\|_{\mathrm{F}}}$$

其中，$\|\cdot\|_{\mathrm{F}}$表示Frobenius定量。

9.3.1.5 HOA 归一化

解码矩阵除以向量**n**，以便将信号转换为N3D归一化，**D**′是为N3D归一化设计的。对于给定的*normalization*参数norm，**n**定义为：

$$\mathbf{n}_n^m = \frac{N_{\mathrm{norm}}{}_n^{|m|}}{N_{\mathrm{N3D}}{}_n^{|m|}}$$

$$\mathbf{n} = [\mathbf{n}_0^0, \mathbf{n}_1^{-1}, \ldots]$$

10 元数据转换

本部分指定了在*typeDefinition*==*Objects*的*audioBlockFormat*s中的极坐标和笛卡尔参数之间进行转换的方法。元数据转换本质上不能是精确的；转换的结果将与不转换的结果不完全匹配。因此，应监测转换结果。注意，范围转换是不可逆的，所以应该避免极坐标和笛卡尔坐标之间的来回转换。

转换功能的接口如下：

```
AudioBlockFormat to_cartesian(AudioBlockFormat input);
AudioBlockFormat to_polar(AudioBlockFormat input);
```

在设置input.cartesian的情况下，使用AudioBlockFormat input输入调用to_cartesian时，输入将按原样返回。相反，如果input.cartesian不是，则使用AudioBlockFormat input输入调用to_polar，input按原样返回。

否则，在这两种情况下，都会反转input.cartesian，并在返回input之前对其进行以下更改:

– input.position根据第10.1节进行转换。

– input.width、input.height和input.depth根据第10.2节进行转换。

– input.objectDivergence根据第10.3节进行转换。

转换是在core.objectbased.conversion中实现的。

10.1 *position*转换

转换位置，使得4+5+0布局中扬声器的极坐标映射到笛卡尔点源声像定位器中使用的扬声器的笛卡尔坐标，如表8所示。

请注意，无论渲染器信道布局如何，都使用相同的转换，即基于4+5+0信道配置的转换。这样做是为了确保转换结果始终保持一致，即使在转换期间不知道正在使用的渲染器再现布局的使用情况下也是如此。选择4+5+0布局主要是为了确保对使用0+5+0创作的内容进行良好的转换。

本节介绍了双向转换所用的通用定义。转换函数本身在第10.1.1和10.1.2节中进行了描述。

map_linear_to_az和map_az_to_linear定义了一对具有方位角φ_l和$azimuth_r$的扬声器之间的方位角φ和线性坐标x之间的源位置的可逆映射，考虑到用于极坐标和笛卡尔坐标的点源声像定位器的平移曲线。

例如，极性位置φ_o在0°和−30°之间，其x位置由下式给出:

$$x = \mathrm{map\_az\_to\_linear}(0, -30, \varphi_o)$$

线性到方位角的映射定义为:

$$\mathrm{map\_linear\_to\_az}(\varphi_l, \varphi_r, x) = \varphi_{\mathrm{mid}} + \varphi_{\mathrm{rel}}$$

其中：

$$\begin{aligned}
\varphi_{\mathrm{mid}} &= \frac{\varphi_l + \varphi_r}{2} \\
\varphi_{\mathrm{range}} &= \varphi_r - \varphi_{\mathrm{mid}} \\
g'_l &= \cos\frac{x\pi}{2} \\
g'_r &= \sin\frac{x\pi}{2} \\
g_r &= \frac{g'_r}{g'_l + g'_r} \\
\varphi_{\mathrm{rel}} &= \frac{180}{\pi}\arctan\left(2\left(g_r - \frac{1}{2}\right)\tan\left(\frac{\pi}{180}\varphi_{\mathrm{range}}\right)\right)
\end{aligned}$$

逆函数定义为：

$$\mathrm{map\_az\_to\_linear}(\varphi_l, \varphi_r, \varphi) = \frac{2}{\pi}\mathrm{atan2}(g_r, 1 - g_r)$$

其中：

$$\begin{aligned}
\varphi_{\mathrm{mid}} &= \frac{\varphi_l + \varphi_r}{2} \\
\varphi_{\mathrm{range}} &= \varphi_r - \varphi_{\mathrm{mid}} \\
\varphi_{\mathrm{rel}} &= \varphi - \varphi_{\mathrm{mid}} \\
g_r &= \frac{1}{2} + \frac{\tan\left(\frac{\pi}{180}\varphi_{\mathrm{rel}}\right)}{2\tan\left(\frac{\pi}{180}\varphi_{\mathrm{range}}\right)}
\end{aligned}$$

该映射应用于中间层扬声器位置之间，根据以下规则给出左和右方位角，以及给定输入方位角的左和右x和y位置：

$$find_{sector}(\varphi)\begin{cases}\{30,0,\{-1,1\},\{0,1\}\} & IAR(\varphi,0,30)\\ \{0,-30,\{0,1\},\{1,1\}\} & IAR(\varphi,-30,0)\\ \{-30,-110,\{1,1\},\{1,-1\}\} & IAR(\varphi,-110,-30)\\ \{-110,110,\{1,-1\},\{-1,-1\}\} & IAR(\varphi,110,-110)\\ \{110,30,\{-1,-1\},\{-1,1\}\} & IAR(\varphi,30,110)\end{cases}$$

$$find\ \ (\varphi)\begin{cases}\{30,0,\{-1,1\},\{0,1\}\} & IAR(\varphi,0,45)\\ \{0,-30,\{0,1\},\{1,1\}\} & IAR(\varphi,-45,0)\\ \{-30,-110,\{1,1\},\{1,-1\}\} & IAR(\varphi,-135,-45)\\ \{-110,110,\{1,-1\},\{-1,-1\}\} & IAR(\varphi,135,-135)\\ \{110,30,\{-1,-1\},\{-1,1\}\} & IAR(\varphi,45,135)\end{cases}$$

式中，IAR是第6.2节所述的`inside_angle_range`的函数。

以下参数对于两个转换方向都是通用的：

$$\begin{aligned}\theta_{top} &= 30\\ \theta'_{top} &= 45\\ \epsilon &= 1\times10^{-10}\end{aligned}$$

10.1.1 极坐标到笛卡尔坐标

将具有方位角φ、仰角θ和距离d的极坐标转换为笛卡尔坐标的函数

$$\texttt{point\_polar\_to\_cart}(\varphi,\theta,d)=x,y,z$$

当$|\theta|>\theta_{top}$时，使用：

$$\begin{aligned}\theta' &= \theta'_{top}+(90-\theta'_{top})\frac{|\theta|-\theta_{top}}{90-\theta_{top}}\\ z &= d\,\mathrm{sgn}(\theta)\\ r_{xy} &= d\tan\left(\frac{\pi}{180}(90-\theta')\right)\end{aligned}$$

反之：

$$\begin{aligned}\theta' &= \theta'_{top}\frac{\theta}{\theta_{top}}\\ z &= d\tan\left(\frac{\pi}{180}\theta'\right)\\ r_{xy} &= d\end{aligned}$$

得出：

$$\begin{aligned}\{\varphi_l,\varphi_r,\{x_l,y_l\},\{x_r,y_r\}\} &= \texttt{find\_sector}(\varphi)\\ \varphi' &= \texttt{relative\_angle}(\varphi_r,\varphi)\\ \varphi'_l &= \texttt{relative\_angle}(\varphi_r,\varphi_l)\\ p &= \texttt{map\_az\_to\_linear}(\varphi'_l,\varphi_r,\varphi')\\ x &= r_{xy}(x_l+p(x_r-x_l))\\ y &= r_{xy}(y_l+p(y_r-y_l))\end{aligned}$$

`relative_angle`在第 6.7节中有说明。

10.1.2 笛卡尔坐标到极坐标

要将坐标为x, y和z的笛卡尔位置转换为极坐标，函数

$$\texttt{point\_cart\_to\_polar}(x, y, z) = \varphi, \theta, d$$

如果$|x| < \epsilon$和$|y| < \epsilon$，则使用：

$$\{\varphi, \theta, d\} = \begin{cases} \{0,0,0\} & |z| < \epsilon \\ \{0,90\text{sgn}(z), |z|\} & \text{otherwise} \end{cases}$$

否则，继续：

$$\begin{aligned} \varphi' &= -\frac{180}{\pi}\text{atan2}(x, y) \\ \{\varphi_l, \varphi_r, \{x_l, y_l\}, \{x_r, y_r\}\} &= \texttt{find\_cart\_sector}(\varphi') \\ [g_l \quad g_r] &= [x \quad y] \cdot \begin{bmatrix} x_l & y_l \\ x_r & y_r \end{bmatrix}^{-1} \\ r_{xy} &= g_l + g_r \\ \varphi'_l &= \texttt{relative\_angle}(\varphi_r, \varphi_l) \\ \varphi_{\text{rel}} &= \texttt{map\_linear\_to\_az}\left(\varphi'_l, \varphi_r, \frac{g_r}{r_{xy}}\right) \\ \varphi &= \texttt{relative\_angle}(-180, \varphi_{\text{rel}}) \\ \theta' &= \frac{180}{\pi}\arctan\frac{z}{r_{xy}} \end{aligned}$$

如果$|\theta'| > \theta'_{\text{top}}$，则：

$$\begin{aligned} |\theta| &= \theta_{\text{top}} + (90 - \theta_{\text{top}})\frac{|\theta'| - \theta'_{\text{top}}}{90 - \theta'_{\text{top}}} \\ \theta &= |\theta|\text{sgn}\theta' \\ d &= |z| \end{aligned}$$

否则：

$$\begin{aligned} \theta &= \theta'\frac{\theta_{\text{top}}}{\theta'_{\text{top}}} \\ d &= r_{xy} \end{aligned}$$

local_coordinate_system在第6.8节有说明。

10.2 范围转换

范围参数的转换分为两部分:

– $whd2xyz$和$xyz2whd$：在笛卡尔坐标系和极坐标系之间转换范围参数的函数，假设源位置直接位于半径为1的侦听器正前方。

– point_polar_to_cart和point_cart_to_polar：进行位置和范围转换的函数。使用第10.1节描述的方法转换位置。范围转换使用转换使用$whd2xyz$和$xyz2whd$，旋转笛卡尔范围以匹配位置。

请注意，范围转换通常不可逆。

10.2.1 极坐标到笛卡尔坐标

extent_polar_to_cart以方位角、仰角和距离以及极宽、高度和深度的形式获取极性源位置，并返回笛卡尔x, y和z坐标以及笛卡尔x, y和z大小：

$$\texttt{extent\_polar\_to\_cart}(\varphi, \theta, d, \texttt{width}, \texttt{height}, \texttt{depth}) = \{x, y, z, s_x, s_y, s_z\}$$

其中：

$$\begin{aligned} \{x, y, z\} &= \texttt{point\_polar\_to\_cart}(\varphi, \theta, d) \\ \{s_{x,f}, s_{y,f}, s_{z,f}\} &= \texttt{whd2xyz}(\texttt{width}, \texttt{height}, \texttt{depth}) \\ [\mathbf{M}_x \quad \mathbf{M}_y \quad \mathbf{M}_z] &= \text{diag}([s_{x,f}, s_{y,f}, s_{z,f}]) \cdot \texttt{local\_coordinate\_system}(\varphi, \theta) \\ s_x &= \| \mathbf{M}_x \|_2 \\ s_y &= \| \mathbf{M}_y \|_2 \\ s_z &= \| \mathbf{M}_z \|_2 \end{aligned}$$

并且

$$whd2xyz(width, height, depth) = \{s_{x,w}, max(s_{y,w}, s_{y,h}, s_{y,d}), s_{z,h}\}$$

其中：

$$\begin{aligned} s_{x,w} &= \begin{cases} \sin\frac{\pi}{180}\frac{\texttt{width}}{2} & \texttt{width} < 180 \\ 1 & \text{otherwise} \end{cases} \\ s_{y,w} &= \frac{1-\cos\frac{\pi\ \texttt{width}}{180\quad 2}}{2} \\ s_{z,h} &= \begin{cases} \sin\frac{\pi}{180}\frac{\texttt{height}}{2} & \texttt{height} < 180 \\ 1 & \text{otherwise} \end{cases} \\ s_{y,h} &= \frac{1-\cos\frac{\pi\ \texttt{height}}{180\quad 2}}{2} \\ s_{y,d} &= \texttt{depth} \end{aligned}$$

10.2.2 笛卡尔坐标到极坐标

extent_cart_to_polar以x, y和z坐标的形式获取笛卡尔源位置，以x, y和z大小的形式获取笛卡尔范围，并以方位角、高程、距离和宽度、高度和深度的形式返回极位置和范围：

$$\texttt{extent\_cart\_to\_polar}(x, y, z, s_x, s_y, s_z) = \{\varphi, \theta, d, \texttt{width}, \texttt{height}, \texttt{depth}\}$$

其中：

$$\begin{aligned} \{\varphi, \theta, d\} &= \texttt{point\_cart\_to\_polar}(x, y, z) \\ [\mathbf{M}_x \quad \mathbf{M}_y \quad \mathbf{M}_z] &= \text{diag}([s_x, s_y, s_z]) \cdot \texttt{local\_coordinate\_system}(\varphi, \theta)^T \\ s_{x,f} &= \| \mathbf{M}_x \|_2 \\ s_{y,f} &= \| \mathbf{M}_y \|_2 \\ s_{z,f} &= \| \mathbf{M}_z \|_2 \\ \{\texttt{width}, \texttt{height}, \texttt{depth}\} &= \texttt{xyz2whd}(s_{x,f}, s_{y,f}, s_{z,f}) \end{aligned}$$

和

$$xyz2whd(s_x, s_y, s_z) = \{w, h, d\}$$

其中：

$$\begin{aligned} w_{sx} &= 2\frac{180}{\pi}\arcsin s_x \\ w_{sy} &= 2\frac{180}{\pi}\arccos(1-2s_y) \\ w &= w_{sx} + s_x\max(w_{sy}-w_{sx},0) \\ h_{sz} &= 2\frac{180}{\pi}\arcsin s_z \\ h_{sy} &= 2\frac{180}{\pi}\arccos(1-2s_y) \\ h &= h_{sz} + s_z\max(h_{sy}-h_{sz},0) \\ \{s_{x,\mathrm{eq}}, s_{y,\mathrm{eq}}, s_{z,\mathrm{eq}}\} &= \mathrm{whd2xyz}(w,h,0) \\ d &= \max(0, s_y - s_{y,\mathrm{eq}},) \end{aligned}$$

10.3 objectDivergence转换

azimuthRange和positionRange根据以下关系转换:

$$positionRange = tan\frac{270\times azimuthRange}{\pi}$$

11 数据结构和表

11.1 内部元数据结构

11.1.1 共享结构

```
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 输入元数据

```
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 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 复制环境数据

```
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 同心扬声器位置

该数据以机器可读的形式在iar/core/data/allo_positions.yaml中可用，但此处将其包括在内以供参考。

表5

0 + 2 + 0的同心扬声器位置

信道	X	Y	Z
M+030	−1	1	0
M-030	1	1	0

表6

0+5+0的同心扬声器位置

信道	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

表7

2+ 5 + 0的同心扬声器位置

信道	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

表8

4+ 5 + 0的同心扬声器位置

信道	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

表9

4+ 5 + 1的同心扬声器位置

信道	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

表10

3+ 7+ 10的同心扬声器位置

信道	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

表11

4+ 9+ 0的同心扬声器位置

信道	X	Y	Z
M+030	−1	1	0
M-030	1	1	0
M+000	0	1	0
M+090	−1	0	0

表 11（结束）

信道	X	Y	Z
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

表12

9 + 10 + 3的同心扬声器位置

信道	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
B+045	−1	1	−1
B-045	1	1	−1

LFE1	-1	1	-1
LFE2	1	1	-1

表13

0+ 7+ 0的同心扬声器位置

信道	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

表14

4 + 7 + 0的同心扬声器位置

信道	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 DirectSpeakers映射数据

此数据以机器可读形式存在于`core.direct_speakers.panner.itu_packs`和`core.direct_speakers.panner.rules`中，但此处包含供参考。

表15

从通用定义audioPackFormatID到布局名称的映射（请参见第8.1节）

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

表16

DirectSpeakers的映射规则（请参见第8.1节）

输入 *speakerLabel*	**输出再现增益**	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		
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	

表16（续）

输入 *speakerLabel*	输出再现增益	input_layouts	output_layouts
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 = $\sqrt{\frac{1}{2}}$		
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		

表16（续）

输入 *speakerLabel*	**输出再现增益**	input_layouts	output_layouts
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 = $\sqrt{\frac{1}{2}}$		
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		

表16（续）

输入 *speakerLabel*	**输出再现增益**	input_layouts	output_layouts
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 = $\sqrt{\frac{1}{2}}$		
U-135	M-030 = $\sqrt{\frac{1}{2}}$		
U+180	U+180 = 1		
U+180	UH+180 = 1		
U+180	U+135 = U-135 = $\sqrt{\frac{1}{2}}$		

表16（续）

输入 *speakerLabel*	**输出再现增益**	input_layouts	output_layouts
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 = $\sqrt{\frac{1}{4}}$		
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 = $\sqrt{\frac{1}{2}}$		

表16（结束）

输入 *speakerLabel*	**输出再现增益**	input_layouts	output_layouts
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		

参考书目

[1] F. Zotter and 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.

附件1的附录1
（资料性）

ADM元数据规范相应部分指南

A1.1 跨ITU-R ADM渲染器的ADM元数据

下表的目的是提供渲染器关键元素的摘要列表，以及它们在附件1中给出的规范中的位置。规范应取自所列参考文献。

ADM元数据 子元素（属性）[坐标系]	**ITU-R BS.2076-1 建议书**	**本建议书的附件1**
typeDefinition == “DirectSpeakers”	§ 5.4.3.1 表11	§ 8
speakerLabel		§ 8.2
位置（方位角、仰角、距离、screenEdgeLock）		§ 8
typeDefinition == “Matrix”	§ 5.4.3.2	§ 5.2.6.1.1 § 5.2.6.4
outputChannelIDRef	表12	§ 5.2.6.1.1
矩阵→系数（增益、gainVar、相位、phaseVar、延迟、delayVar）	表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
位置（方位角、仰角、距离、screenEdgeLock）[极坐标]	表14	§ 6.1 § 7 § 7.3.4
位置（X，Y，Z，screenEdgeLock）[笛卡尔坐标]	表15	§ 6.1 § 7 § 7.3.10
宽度、高度、深度[极坐标]	表14	§ 7.3.8
宽度、高度、深度[笛卡尔坐标]	表15	§ 7.3.11
笛卡尔坐标	表16	§ 7.3.1 § 7.3.2
增益		§ 7.3.1
扩散		§ 7.3.1 § 7.4
channelLock (maxDistance)		§ 7.3.6
objectDivergence (azimuthRange, positionRange) [极坐标]		§ 7.3.7 § 7.3.1
objectDivergence (azimuthRange, positionRange) [笛卡尔坐标]		§ 7.3.7 § 7.3.1
jumpPosition (interpolationLength)		§ 7.2

ADM元数据 子元素（属性）[坐标系]	ITU-R BS.2076-1 建议书	本建议书的 附件1
zoneExclusion → zone (minX, maxX, minY, maxY, minZ, maxZ, minElevation, maxElevation, minAzimuth, maxAzimuth)		§ 7.3.5 § 7.3.12
screenRef		§ 7.3.3
重要性		§ 5.3.1 § 5.2.7.1.1
typeDefinition == “HOA”	§ 5.4.3.4	§ 9 § 5.2.7.3
公式	表17	§ 9.2
级别		§ 9.1.1 § 9.3.1.2
等级		§ 9.1.1 § 9.3.1.2
归一化	§ 5.4.3.4	§ 9.1.2 § 9.3.1.5
nfcRefDist	表17	§ 9.2
screenRef		§ 9.2
typeDefinition == “Binaural”	§ 5.4.3.5	–

附件1的附录2
（资料性）

另一种虚拟扬声器配置

A2.1 可选虚拟扬声器配置规范

与第6.1.3.1节中规定的配置相比，另一种VBAP虚拟扬声器配置描述了不位于极点上的虚拟扬声器的位置及其折减系数。ADM元数据的处理方式与本建议书正文中规定的相同，不需要额外的元数据。另一种虚拟扬声器的位置及其折减系数是基于耳部优化的。下文描述了该另一种虚拟扬声器配置。

A2.1.1 配置流程

配置过程遵循第6.1.3.1节中所述的步骤，但第二步除外，该步骤如下：

2) 通过首先查找第A2.1.2节中定义的表来确定虚拟扬声器。第A2.1.2节中的每个小节都定义了虚拟扬声器配置及其针对ITU-R BS.2051-2建议书中定义的特定布局的折减系数。

其他配置过程步骤（步骤（1）和步骤（3）至步骤（6））仍保留在第6.1.3.1中所述。下面是此可选虚拟扬声器配置的说明。

A2.1.2 虚拟扬声器和折减表

在下表中，虚拟扬声器（指定为方位角和仰角）在第一行中，物理扬声器在第一列中。虚拟扬声器位置具有相同的标称位置和实际位置。该表显示了从虚拟扬声器到物理扬声器的折减系数。

系统A: 0+2+0

对于系统A：0+2+0，使用第6.1.2.4节中所述的基于从系统B:0+5+0到系统A:0+2+0的下混方法。为了获得0+5+0信道，如下所述使用系统B:0+5+0的虚拟扬声器。

系统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

系统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						

系统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				

系统E: 4+5+1

此布局同时具有上部和下部扬声器。外壳完整，因此不需要虚拟扬声器。

系统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						

系统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				

系统H: 9+10+3

包含上下半球的扬声器；外壳完整，因此不需要虚拟扬声器。

系统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.	

系统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				