ETSI TS 103 584 V1.1.1 (2018-01)

TECHNICAL SPECIFICATION

DTS-UHD Point Source Renderer

EBU
OPERATING EUROVISION

Reference
DTS/JTC-DTS-PSR

Keywords
audio, broadcast, codec, digital, object audio

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

List of Tables

List of Figures

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document defines an audio renderer associated with the DTS-UHD codec defined in ETSI TS 103 491 [1]. The inputs to the renderer are one or more sets of audio waveforms along with mixing instructions, and the output is a single set of waveforms mapped to a defined speaker configuration. Each set of waveforms may represent either audio channels or audio objects. The mixing instructions may vary with time and they come from metadata carried in the DTS-UHD bitstream and from other application interfaces described herein.

2 References

2.1 Normative References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 103 491: "DTS-UHD Audio Format; Delivery of Channels, Objects and Ambisonic Sound Fields", version 1.1.1.

[2] Pulkki, Ville: "Virtual Sound Source Positioning Using Vector Base Amplitude Panning", JAES Volume 45 Issue 6 pp. 456-466; June 1997.

2.2 Informative References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

Not applicable.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

audio object: waveform or set of waveforms that have been assigned a location in space

hull: collection of points forming a surface mesh to be used as a basis for panning

output feeds: set of audio waveforms mapping one to one to the physical speakers

panning: distributing a point source location into a speaker layout

physical speaker: speaker that physically exists in the listening space

point: location on the surface of the unit sphere

point source: audio waveform with defined spatial coordinates within the listening room

NOTE: A point source is treated (mathematically) as a uniform spherical radiation pattern, though in reality most speakers have a pattern of less than 180 degrees on the face of the speaker. This reality is accounted for by limiting the panner range.

render: combine waveforms with their gain contributions to produce audio waveforms in a specified speaker configuration

speaker: transducer that converts an electrical signal into soundwaves

speaker configuration: defined list of physical speakers in the listening space

triplet: set of three points

virtual speaker: speaker that does not physically exist in the listening space

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

∪	Union operator
3D	three dimensional
C	Centre speaker
Ch	Centre high speaker
Chr	Centre high rear speaker
Clf	Centre low front speaker
Cs	Centre surround speaker
dB	decibel
EMA	Exponential Moving Average
L	Left speaker
Lc	Left centre speaker
LFE	Low Frequency Effects
LFE1	Low Frequency Effects - 1 speaker
LFE2	Low Frequency Effects - 2 speaker
Lh	Left high speaker
Lhr	Left high rear speaker
Lhs	Left high side speaker
Llf	Left low front speaker
LR	Left Right
Ls	Left surround speaker
Lsr	Left surround rear speaker
Lss	Left side surround speaker
Ltf	Left top front speaker
Ltr	Left top rear speaker
Lw	Left wide speaker
N/A	Not Applicable
Oh	Overhead speaker
R	Right speaker
Rc	Right centre speaker
Rh	Right high speaker
Rhr	Right high rear speaker
Rhs	Right high side speaker
Rlf	Right low front speaker
Rs	Right surround speaker
Rsr	Right surround rear speaker
Rss	Right side surround speaker

Rtf	Right top front speaker
Rtr	Right top rear speaker
Rw	Right wide speaker

4 Renderer Metadata

4.1 Overview of the Renderer Inputs

The DTS-UHD point source object renderer is a system that computes the waveform associated with a specified speaker configuration, given a collection of audio objects as input, as shown in Figure 1. The renderer metadata assumes the model of a sphere with the listener's head at the centre (an ego-centric model). All speakers are treated as point sources and are assumed to be on the surface of a unit sphere, and thus have a radius of 1.

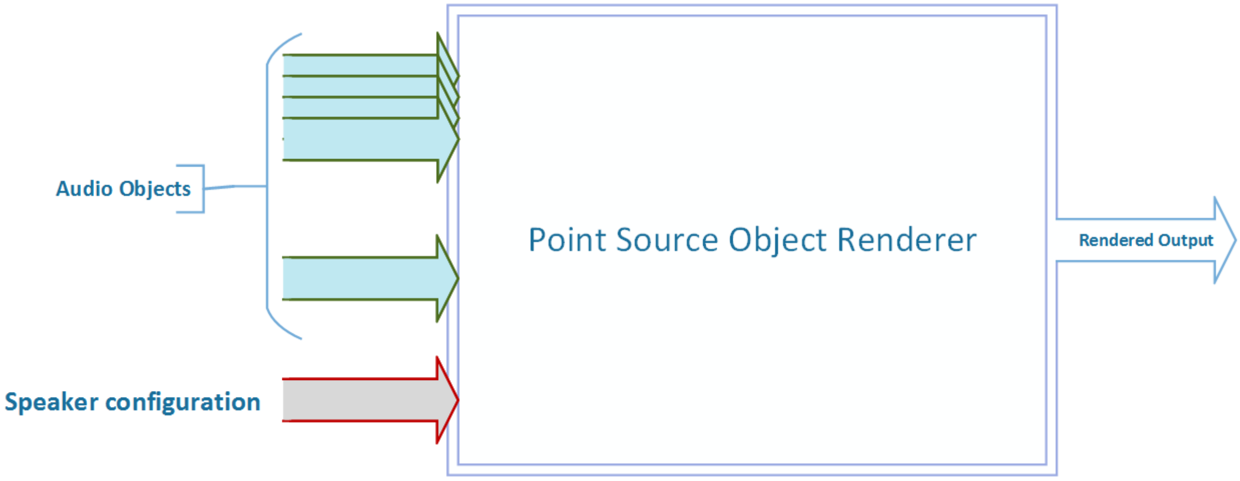

Figure 1: Point Source Object Renderer System

DTS-UHD audio objects consist of metadata plus one or more waveforms. The metadata always includes spatial location information, and may include additional parameters, such as gain information.

Audio objects can be manipulated by changing the gain and position metadata. For example, when the renderer metadata is coming from a DTS-UHD bitstream, the metadata can be updated as often as every 512 clock periods, where the clock shall be defined by `m_unClockRateInHz` in ETSI TS 103 491 [1]. If the audio within the DTS-UHD stream is carrying audio compressed using ACE, each ACE frame has a duration of 1 024 samples, therefore the mixing metadata can be updated twice per ACE frame. For audio with a sampling frequency of 48 KHz, this translates to a minimum update interval of 10,67 ms.

It is also possible for object metadata to be modified by user intervention. In this case, the bitstream metadata is being overridden until control is given back to the bitstream metadata.

The present document describes these in details to provide a comprehensive understanding of the DTS-UHD Point Source Object Renderer.

4.2 DTS-UHD Bitstream Metadata

DTS-UHD metadata is defined in ETSI TS 103 491 [1]. Since the renderer described in the present document is intended to be used in conjunction with the DTS-UHD bitstream, the DTS-UHD metadata parameters relating to the renderer are shown in Table 1.

Note that Table 1 accommodates restrictions inherent in DTS-UHD metadata, but this does not imply such restrictions are imposed on the renderer capabilities.

Table 1: DTS-UHD Metadata Parameters used for Rendering

Metadata Parameter	Reference in ETSI TS 103 491 [1]	Description	Object Renderer Parameter
m_unObjectID	7.8.7	Identifier to access and modify an object state.	`AudioObject::object_id`
m_bMonoObjWithMultipleSourcesFlag (or bObjMS for short)	7.8.11.2	Indicates whether the AudioObject with one MonoObject is associated with multiple 3D points.	used with m_ucNum3DSourcesInObj (below)
m_ucNumWaveFormsInObj	7.8.11.17	Specifies the number of waveforms (MonoObjects) in the AudioObject.	`AudioObject::` `mono_objects[].size()`
m_bPerSamplPeriodObjMDUpdFlag	7.8.11.22.2	Flag indicates that there are metadata updates per processing block within a larger audio block.	N/A
m_ppObjGain	7.8.11.22.3	Specifies the AudioObject gain.	`AudioObject::` `object_gain`

m_ucNum3DSourcesInObj	7.8.11.22.4	Specifies the total number of point source locations in the AudioObject.
		Note that DTS-UHD metadata restricts an AudioObject to have either multiple MonoObjects each with a single point source location, or a single MonoObject with multiple point source locations.
			`if (bObjMS)`
			`{`
			`MonoObject::`
			`positions[].size()`
			`AudioObject::`
			`mono_objects[].size() = 1`
			`}`
			`else`
			`{`
			`AudioObject::`
			`mono_objects[].size()`
			`MonoObject::`
			`positions[].size() = 1`
			`}`

unsrc_index	7.8.11.22.5	Identifies the source index within the AudioObject.	N/A
m_rPerObjExpWinLambda	7.8.11.23	Specifies the smoothing factor for the AudioObject.	`AudioObject::` `obj_lambda`

m_3DSrcRadius	7.8.11.28.2	Specifies radius of the point source location.	`AudioObject::`
			`mono_objects[].`
			`positions[].radius`

m_3DSrcAzimuth	7.8.11.28.3	Specifies azimuth of the point source location. Range is $[-180°, 180°]$.	`AudioObject::`
			`mono_objects[].`
			`positions[].azimuth`

m_3DSrcElevation	7.8.11.28.4	Specifies elevation of the point source location. Range is $[-90°, 90°]$.	`AudioObject::`
			`mono_objects[].`
			`positions[].elevation`

4.3 DTS-UHD Object Interactivity Manager

Renderer APIs may allow the object metadata from the DTS-UHD bitstream to be overridden by the user during playback. Metadata in the bitstream may also be set to limit or disable a user interaction. The object interactivity manager enforces these rules and applies any user changes to the metadata before calling the renderer. Figure 2 shows that the object interactivity manager sits just before the renderer, where it handles the user input and the limit rules specified by the bitstream creator.

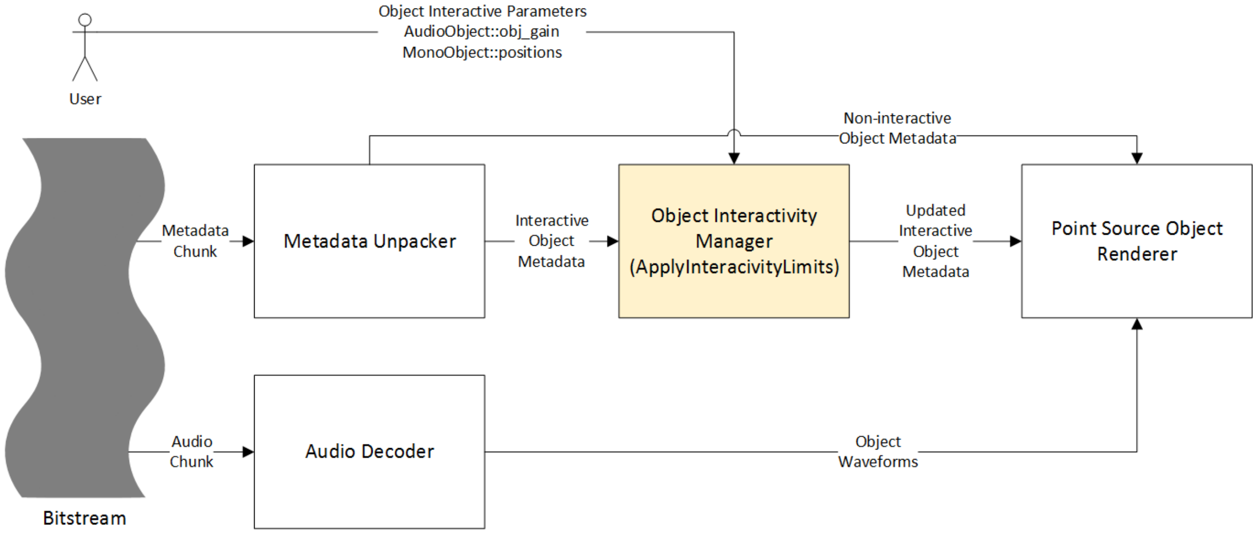

Figure 2: Object Interactivity Manager

Table 2 shows the interactive parameters specified in the DTS-UHD bitstream.

Table 2: Object Interactive Parameters Specified in the Bitstream

Input	Reference in ETSI TS 103 491 [1]	Description
m_bObjInteractiveFlag	7.8.11.12.1	Determines whether parameters can interactively change.
m_bObjInterLimitsFlag	7.8.11.12.2	Determines whether parameters have range limits constraints.
m_unMaxInterObjGainBoostdB	7.8.11.12.3	Specifies the maximum gain boost permitted.
m_unMaxInterObjGainAttendB	7.8.11.12.4	Specifies the maximum gain attenuation permitted.
m_unObjInterPosMaxDeltaAzim	7.8.11.12.5	Specifies the maximum allowed change in azimuth.
m_unObjInterPosMaxDeltaElev	7.8.11.12.6	Specifies the maximum allowed change in elevation.

The following pseudocode defines the function of an object interactivity manager.

```
ApplyInteractivityRules()
{
    // For each object in the bitstream
    for ( bitstream_obj : bitstream_obj_list )
    {
        // Only if this flag is true, the object is allowed to be interactive
        if ( bitstream_obj.m_bObjInteractiveFlag )
        {
            if ( bitstream_obj.m_bObjInterLimitsFlag )
            { // If this flag is set there are limits to the interactive parameters

                gain_max = bitstream_obj.m_ppObjGain +
                    bitstream_obj.m_unMaxInterObjGainBoostdB;
                gain_min = bitstream_obj.m_ppObjGain -
                    bitstream_obj.m_unMaxInterObjGainAttendB;

                user_interactive_obj_gain =
                    clip( user_interactive_obj_gain, gain_min, gain_max );
            }

            // Fetch PointSourceObjectRenderer's AudioObject to be updated
            AO = GetAudioObject( bitstream_obj.m_unObjectID );

            if ( has_user_specified_obj_gain ) {
                // Update gain
                AO.obj_gain = user_interactive_obj_gain;
            }

            // for each MonoObject
            for ( MO : AO.mono_objects )
            {
```

```
                // for each point in the MonoObject
                for ( p : MO.positions )
                {
                    if ( bitstream_obj.m_bObjInterLimitsFlag )
                    { // If this flag is set there are limits to the
                        // interactive params

                        // Limit azimuth
                        az_min = p.azimuth -
                            bitstream_obj.m_unObjInterPosMaxDeltaAzim;
                        az_max = p.azimuth +
                            bitstream_obj.m_unObjInterPosMaxDeltaAzim;
                        user_interactive_obj_position.azimuth =
                            clip( user_interactive_obj_position.azimuth, az_min, az_max );

                        // Limit Elevation
                        el_min = p.elevation -
                            bitstream_obj.m_unObjInterPosMaxDeltaElev;
                        el_max = p.elevation +
                            bitstream_obj.m_unObjInterPosMaxDeltaElev;
                        user_interactive_obj_position.elevation =
                            clip( user_interactive_obj_position.elevation, el_min, el_max );
                    }

                    if (has_user_specified_obj_position)
                    {
                        // update position
                        p = user_interactive_obj_position;
                    }
                }
            }
        }
    }
}
```

5 Variables and Class Types

5.1 Global Variables

The following data types and variables are used throughout the present document:

- **AO:** An object of type AudioObject.
- **H:** An object of type Hull.
- **MO:** An object of type MonoObject.
- **PSOR:** An object of type PointSourceObjectRenderer.
- **SC:** An object of type SpeakerConfiguration. A speaker configuration consists of a list of speaker locations that physically exist in the listening space.
- **V:** List of all virtual speakers or $V = V_e \cup V_u \cup V_l$.
- **V_e** or `V_e`: List of virtual speakers in the equatorial plane.
- **V_l** or `V_l`: List of virtual speakers in the lower hemisphere.
- **V_u** or `V_u`: List of virtual speakers in the upper hemisphere.
- **VBP:** An object of type VectorBasedPanner.

5.2 Renderer Object Classes

5.2.1 Point

A point is a location of any entity (for example, speaker or audio object) in the listening space. It is specified by polar coordinates - azimuth (θ) elevation (φ) and radius (r). For the purposes of the present document, only the points on the unit sphere are needed, which means the radius shall be equal to 1. The listener is at the origin ($\theta = 0, \varphi = 0, r = 0$), facing the location ($\theta = 0, \varphi = 0, r = 1$).

Figure 3 shows the coordinate system of the renderer. Note that the equatorial plane is represented with $\varphi = 0$. The upper hemisphere is specified as $\varphi > 0$, and lower hemisphere is specified as $\varphi < 0$.

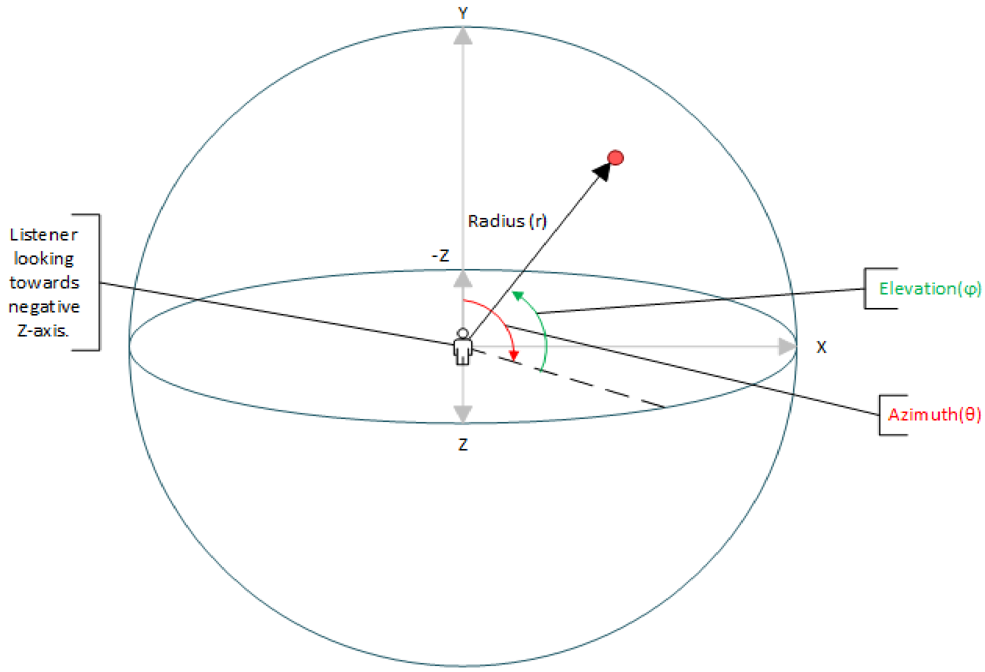

Figure 3: Point Source Object Renderer Coordinate System

The following pseudocode defines a point class. For convenience in determining the distance between points, a conversion to Cartesian coordinates is provided below:

```
class Point {
    float azimuth;
    float elevation;
    float radius;

    Cartesian();
};

// Converting to Cartesian coordinate system.
Point::Cartesian()
{
    float x = radius * sin(azimuth) * cos(elevation);
    float y = radius * sin(elevation);
    float z = radius * -cos(azimuth) * cos(elevation);
    return (x,y,z);
}
```

5.2.2 Edge

An Edge is a line segment connecting two points. For the purposes of the present document, the edge structure shall be a set of two Points.

```
struct Edge
{
    // Constructor
    Edge(Point p1, Point p2) : p1(p1), p2(p2) {}

    // The two points representing the edge.
    Point p1, p2;

    // Equals operator
    bool operator==(Edge other)
    {
        return ((p1 == other.p1 && p2 == other.p2) || (p1 == other.p2 && p2 == other.p1));
    }
};
```

5.2.3 SpeakerConfiguration

A speaker configuration lists the location of physical speakers in the listening space. This shall be established when initializing the renderer. Speaker configuration is represented as follows:

```
class SpeakerConfiguration
{
    Point speaker_locations[];
    uint32 GetChannelMask();
};

SpeakerConfiguration::GetChannelMask()
{
    uint32 channel_mask = 0;
    for ( sl : speaker_locations ) {

    // GetChannelMask(sl) fetches channel mask for predefined set of locations in Table 3.
    // If the location is not specified, then returns 0.

        uint32 location_ch_mask = GetChannelMask(sl);

        if (location_ch_mask)
            channel_mask |= location_ch_mask;
        else {
            channel_mask = 0;
            break;
        }
    }
    return channel_mask;
}
```

5.2.4 EMASmoother

`EMASmoother` is a class supporting an Exponential Moving Average (EMA) smoothing function. The renderer processes each audio object in blocks. A block contains a set of audio samples with associated metadata. The size of the block, defined as the number of audio samples in the block, is specified at the rendering stage. The block size is dynamic, adapting to the frequency of metadata updates. The audio object properties, such as object gain or mono object locations, can change from block to block. This dynamic change results in a change of gain coefficients between blocks. At these transitions, the renderer uses an exponential moving average smoothing algorithm, defined as `Smooth()`, to smooth the gains. In Figure 4 the position of `MonoObject` changes between the block at time *t-1* and the block at time *t*, affecting the gain contributions. For example, the gain contribution of `MonoObject` to the first speaker in the output may change from $g1(t-1) = 0$ to $g1(t) = 1$. The EMASmoother curve for this transition is shown in Figure 4. The curve may span multiple blocks to settle on the new gain depending on `lambda` and `block_size`.

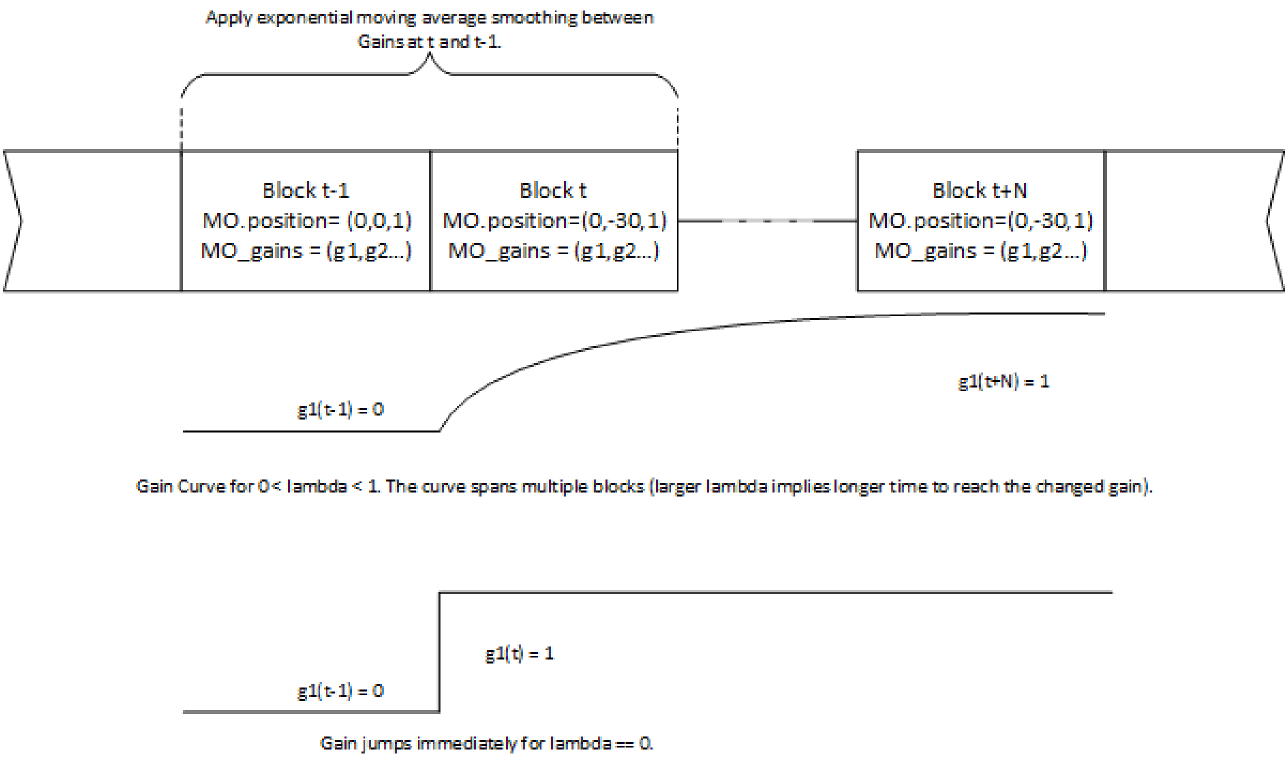

Figure 4: EMASmoother Sample Application

The pseudocode below provides the implementation of the exponential moving average smoother. The smoothing factor, lambda, may change at every block.

The previous gain value (`prev_gain`) is initialized to 0 when the object is created. This results in a ramp up to the new value when calculated. The user can choose not to ramp up by setting lambda to 0.

```
class EMASmoother
{
    float lambda;
    float prev_gain;
    Smooth(float new_gain, uint block_size);
};

EMASmoother::Smooth(float new_gain, uint block_size)
{
    float smoothed_gain_over_a_block[block_size];

    for ( uint t : 0 to block_size )
    {
        smoothed_gain_over_a_block[t] = new_gain * (1-lambda) + prev_gain * lambda;
        prev_gain = smoothed_gain_over_a_block[t];
    }
    return smoothed_gain_over_a_block;
}
```

5.2.5 MonoObject

A mono object consists of a single waveform with a list of associated points. A mono object can have more than one point source location. The following structure defines the mono object class.

```
class MonoObject
{
    // List of point sources locations.
    Point positions[];

    // Single waveform - array of audio samples for the block (an audio frame)
    float waveform[];
    // Smoother list - a persistent parameter that exist through the lifetime of the MonoObject.
    // It is of length equal to the SpeakerConfiguration::speaker_locations.size().
    EMASmoother smoother_list[];
```

```
};
```

5.2.6 AudioObject

An audio object is defined as a set of mono objects. An audio object consists of object properties such as, `object_id`, `object_gain`, `object_lambda` (smoothing factor) along with a list of `MonoObjects`. The following structure defines the `AudioObject` class:

```
class AudioObject
{
    MonoObject mono_objects[];
    float obj_lambda;
    float obj_gain;
    uint object_id;
};
```

5.2.7 Vector

The definition of the Vector class is provided below without an implementation. It supports basic vector operations, such as dot product and cross products.

```
// A Vector class defined without implementation.
class Vector
{
    Dot( Vector other ); // Computes dot product
    Cross( Vector other ); // Computes cross product
    int data[]; // internal data accessed through operator[]
};
```

5.2.8 Matrix

The definition of the Matrix class is provided below without an implementation. It supports basic matrix multiplication and inverse functions. The Matrix class shall have fixed dimensions of three rows by three columns.

```
// A matrix class defined without implementation.
class Matrix
{
    Column(col); // Peek the column col
    Row(row); // Peek at the row
    IsInvertible(); // True if the matrix is invertible.
    Invert(); // Returns inverse of a matrix
    Operator*( Matrix other ); // Multiply with matrix 'other'.
    int data[][]; // internal data accessed through operator[]
};
```

5.2.9 Triplet

A triplet is a spherical triangle formed on the unit-sphere surface specified by a set of three non-collinear points. Triplets are used within `Hull` (clause 5.2.10) to represent its surface mesh. Triplet is defined as follows:

```
class Triplet
{
    Point p1;
    Point p2;
    Point p3;

    // return Matrix object
    GetTripletMatrix();

    // rearrange points so that triangle formed by them are positively oriented
    ConvertToPositiveOrientation();

    // Triplet in equator or upper hemisphere
    FullyInUpperHemisphere();

    // Triplet in equator or lower hemisphere
    FullyInLowerHemisphere();

    // Triplet in equator
    FullyInEquatorialPlane();
```

```
};
```

Triplet::GetTripletMatrix

```
Triplet::GetTripletMatrix()
{
    Matrix M;
    M.Column(0) = p1.Cartesian();
    M.Column(1) = p2.Cartesian();
    M.Column(2) = p3.Cartesian();
    return M;
}
```

Triplet::ConvertToPositiveOrientation

```
// Convert triplets into positive-oriented convex polygon.
Triplet::ConvertToPositiveOrientation()
{
    Vector v1 = p1.Cartesian();
    Vector v2 = p2.Cartesian();
    Vector v3 = p3.Cartesian();

    Vector V[] = {v1, v2, v3, v1}; // cyclic

    // Find the sign of first Vector
    Vector n = V[0].Cross(V[1]);
    dots = [];
    for ( i : range(V.size()) )
        dots[i] = V[i].Dot(n);
    sign = GetSign(dots);

    // Match the sign with other Vectors
    for ( i : range(1, V.size()-1) )
    {
        n = V[i].cross(V[i+1]);
        for ( i : range(V.size()) )
            dots[i] = V[i].Dot(n);

        if ( sign != GetSign(dots) )
            sign = 0;
    }

    // negative oriented reverse the order of p1,p2,p3
    if (sign == -1)
        Swap(p1, p3);
}

// Return +1 if all positive
// Return -1 if all negative
// Return  0 if mixed or all zero
GetSign(float arr[])
{
    all_pos = true;
    all_neq = true;
    for ( a : arr ) {
        if ( a < 0 )
            all_pos = false;
        if ( a > 0 )
            all_neg = false;
    }
    sign = 0;
    if ( all_pos && !all_neg )
        sign = 1;
    if ( !all_pos && all_neg )
        sign = -1;
    return sign;
};
```

Triplet::FullyInUpperHemisphere

For the purposes of this function, the upper hemisphere includes the equator.

```
Triplet::FullyInUpperHemisphere()
{
    return (p1.elevation >= 0 && p2.elevation >= 0 && p3.elevation >= 0);
}
```

Triplet::FullyInLowerHemisphere

For the purposes of this function, the lower hemisphere includes the equator.

```
Triplet::FullyInLowerHemisphere()
{
    return (p1.elevation <= 0 && p2.elevation <= 0 && p3.elevation <= 0);
}
```

Triplet::FullyInEquatorialPlane

```
Triplet::FullyInEquatorialPlane()
{
    return (p1.elevation == 0 && p2.elevation == 0 && p3.elevation == 0);
}
```

5.2.10 Hull

A hull is a set of points formed by combining the physical loud speaker locations and the virtual speaker locations. These points are used to form a surface mesh by grouping them into triplets. The structure below defines the `Hull` class.

```
class Hull
{
    Triplet T[]; // list of triplets
    Point hull_points[]; // list of points that form the hull
    Float epsilon;  // epsilon is used to handle floating point errors

    // Initialize the hull
    Init(Point hull_points[])

    // Pan the point p into the hull
    Pan(Point P);
};
```

5.2.11 Virtual Speakers

A virtual speaker for a given speaker configuration is specified by the position on the unit sphere and the fold-down contributions to the specified physical speaker configuration. Virtual speakers enable the vector based panner to create a complete three-dimensional hull. The vector based panner is supported by two virtual speaker classes.

The first class is `PredefinedVirtualSpeakers`, which supports virtual speaker generation for a specific set of output speaker configurations. The structure below defines the class `PredefinedVirtualSpeakers`. The definition and use of the predefined virtual speakers is further defined in clause 6.3.2.

```
class PredefinedVirtualSpeakers
{
    // Check if the speaker configuration is supported
    IsSupported(SpeakerConfiguration SC);

    // Find virtual speakers and fold-down coefficients
    GetVirtualSpeakersAndFolddownCoeffs(SpeakerConfiguration SC);
}The other class of virtual speaker is AutoVirtualSpeaker. When the speaker configuration is not
supported by predefined virtual speakers, this second algorithm is used. This algorithm can support
arbitrary speaker configurations. The structure below defines the class AutoVirtualSpeakers. The
algorithm and use of auto virtual speakers is further defined in clause 6.3.3.
class AutoVirtualSpeakers
{
    // Find virtual speakers and fold-down coefficients
    GetVirtualSpeakersAndFolddownCoeffs(SpeakerConfiguration SC);
}
```

5.2.12 VectorBasedPanner

The renderer uses vector based panning to calculate gain contributions of every point source location to the given output speaker configuration. The algorithm for the vector based panner is further defined in clause 6.5.2.

The following pseudocode defines the `VectorBasedPanner` class.

```
class VectorBasedPanner
{
    SpeakerConfiguration SC;
    Point V[];              // virtual points
    float FD[][];           // fold-down coefficients for each virtual point
    Hull H;                 // Hull used for panning
```

```
    // Initialize the panner
    Init(SpeakerConfiguration SC);

    // Pan the point p
    Pan (Point p);
};
```

5.2.13 PointSourceObjectRenderer

The class `PointSourceObjectRenderer` shows the interface to the renderer. There are two public APIs:

- `Init()` - renderer initialization
- `RenderAllObjects()` - rendering objects

The calls to the renderer are defined below:

```
class PointSourceObjectRenderer
{
public:
    // Initialize the renderer.
    Init(SpeakerConfiguration sc);

    // Render the objects in AO_list.
    RenderAllObjects(AudioObject AO_list[], uint block_size );

private:

    // Render a single AudioObject.
    RenderAudioObject(AudioObject AO, uint block_size );

    // Render a MonoObject.
    RenderMonoObject(
        MonoObject MO,
        uint block_size,
        float obj_gain,
        float obj_lambda );

    // Speaker configuration this instance of object renderer is initialized with.
    SpeakerConfiguration sc;

    // Vector based panner for a given speaker configuration.
    VectorBasedPanner vbp;

    // The output waveforms in the specified speaker configuration.
    float out_waveforms[][];
};
```

5.3 Other Helper Functions

5.3.1 Join

`Join()` combines multiple collections of objects into one container. A simple pseudocode for `Join()` is provided below:

```
// Join lists in a two dimensional array
// or arrays specified in comma separated list.
Join( collection[][] )
{
    res = [];
    for ( i : collection.size() )
    {
        for ( val : collection[i].size() )
            res += val;
    }
    return res;
}
```

5.3.2 MapAzimuthTo0_360

`MapAzimuthTo0_360()` converts azimuth to all positive angles. Angles in the range [-180° : 0°] are mapped to [180° : 360°].

```
MapAzimuthTo0_360(Point points[])
{
    modified_points = []
    for (p : points)
    {
        new_az = (p.azimuth - floor(p.azimuth / 360) * 360);
        modified_points += { new_az, p.elevation, p.radius };
    }
    return modified_points;
}
```

5.3.3 GetEquatorialPoints

`GetEquatorialPoints()` returns the points on the equator.

```
GetEquatorialPoints(Points points[])
{
    equatorial_points = [];

    for ( p : points )
    {
        if ( p.elevation == 0 )
            equatorial_points += p;
    }
    return equatorial_points;
}
```

5.3.4 NumUpperHemispherePoints

`NumUpperHemispherePoints()` returns the number of points in the upper hemisphere. For the purposes of this function, this excludes points on the equator.

```
NumUpperHemispherePoints(Point points[])
{
    count = 0;

    for ( p : points )
    {
        if (p.elevation > 0)
            ++count;
    }
    return count;
}
```

5.3.5 NumLowerHemispherePoints

`NumLowerHemispherePoints()` returns the number of points in the lower hemisphere. For the purposes of this function, this excludes points on the equator.

```
NumLowerHemispherePoints(Point points[])
{
    count = 0;

    for ( p : points )
    {
        if (p.elevation < 0)
            ++count;
    }
    return count;
}
```

6 Point Source Object Renderer

6.1 Operation of the Renderer

The rendering system shall be initialized with an output `SpeakerConfiguration` before it is used. After initialization, a list of audio objects is rendered one block at a time. The underlying system that enables initialization and rendering is shown in Figure 5. The interface to the renderer consists of two APIs:

- `Init()`
  - Initializes VectorBasedPanner
- `RenderAllObjects()`
  - Pans using vector based panner
  - Applies exponential smoothing
  - Renders the objects into the output feeds

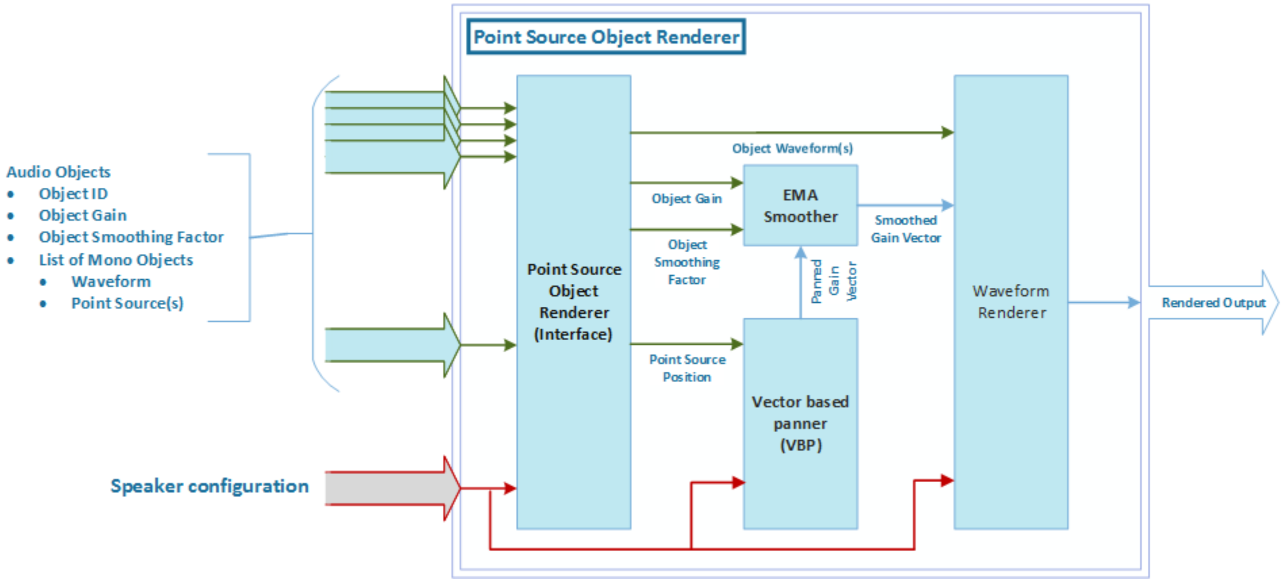

Figure 5: Point Source Object Renderer Block Diagram

A higher-level usage of `PointSourceObjectRenderer` is shown in the pseudocode below. Here the renderer is initialized with a speaker configuration object followed by the actual rendering.

```
// Declare PointSourceObjectRenderer
PointSourceObjectRenderer PSOR;

// Define SpeakerConfiguration
SpeakerConfiguration SC = {
    speaker_location1,
    speaker_location2,
    ...
    speaker_locationN
};

// Initialize renderer
PSOR.Init(SC);

// Run renderer for all blocks
for ( audio_objects[] : audio_object_blocks )
{
    // Render all objects
    PSOR.RenderAllObjects(audio_objects);
}
```

6.2 Renderer Initialization

The initialization of the renderer involves identifying virtual speakers and setting up the hull for panning, both of which are handled during initialization of `VectorBasedPanner`. The `PointSourceObjectRenderer::Init()` is initializing `VectorBasedPanner` as shown in the pseudocode below:

```
PointSourceObjectRenderer::Init(SpeakerConfiguration SC)
{
    this->SC = SC;
    VBP.Init(SC);
}
```

Initialization of `VectorBasedPanner` takes the speaker configuration and determines the number and location of any required virtual speakers and their fold-down coefficients. The virtual speakers are then used to initialize the convex hull, refer to the pseudocode below:

```
VectorBasedPanner::Init(SpeakerConfiguration SC)
{
    this->SC = SC;

    if (PredefinedVirtualSpeakers::IsSupported(SC))
        V, FD = PredefinedVirtualSpeakers::GetVirtualSpeakersAndFolddownCoeffs(SC);
    else
        V, FD = AutoVirtualSpeakers::GetVirtualSpeakersAndFolddownCoeffs(SC);

    hull_points = Join( SC.speaker_locations, V );
    H.Init( hull_points );
}
```

6.3 Virtual Speakers

6.3.1 Virtual Speakers Positioning

For listening layouts with large angular spacing between the speakers, it may be impossible to create the necessary convex hull for the proper operation of vector based 3D rendering. Successful creation of a convex hull over a set of physical speakers is achieved by carefully placing additional virtual speakers. In this case, the VBP rendering is done over a set of physical and virtual speakers and the fold-down of the virtual speakers into the physical speakers is done as a post VBP processing step. The number and position of the virtual speakers and the corresponding fold-down coefficients can greatly influence the spatial impression of the rendered sounds. `PointSourceObjectRenderer` uses two methods for placing virtual speakers: `PredefinedVirtualSpeakers` and `AutoVirtualSpeakers` in that order. If the output configuration can be supported by the `PredefinedVirtualSpeaker` algorithm, then `PredefinedVirtualSpeakers` are used, otherwise the `AutoVirtualSpeaker` algorithm is used.

6.3.2 Predefined Virtual Speakers

6.3.2.1 Predefined Virtual Speakers Layouts

Predefined virtual speakers have been carefully designed for specific sets of physical speaker layouts. The supported sets of physical speaker layouts are classified into groups which are enumerated in clauses 7.3 and 7.4. The predefined virtual speakers consist of 10 predefined locations, a subset of which is chosen depending on the physical speaker layout.

The predefined virtual speakers in the equatorial plane are located as follows:

$$V_e = [(-135°, 0°, 1), (135°, 0°, 1)]$$

The predefined virtual speakers in the upper hemisphere are located as follows:

$$V_u = [(-45°, 45°, 1), (45°, 45°, 1), (-135°, 45°, 1), (135°, 45°, 1)]$$

The predefined virtual speakers in the lower hemisphere are located as follows:

$$V_l = [(-45°, -45°, 1), (45°, -45°, 1), (-135°, -45°, 1), (135°, -45°, 1)]$$

The actual virtual speakers chosen are subsets of these speakers and vary depending on the physical speaker layout. The subsets for the equatorial plane and upper hemisphere are given in clause 7.3, and for the lower hemisphere in clause 7.4.

Figure 6 shows how the virtual speakers are added for a stereo (LR) configuration. Predefined virtual speakers (yellow vertices) for the stereo destination layout (red vertices). The figure shows both the upper and lower hemisphere. The physical stereo speakers are indicated by the red vertices, and the virtual speakers in yellow are added in the equatorial plane, the upper hemisphere, and the lower hemisphere. For the stereo configuration, all ten of the possible predefined virtual speakers are used.

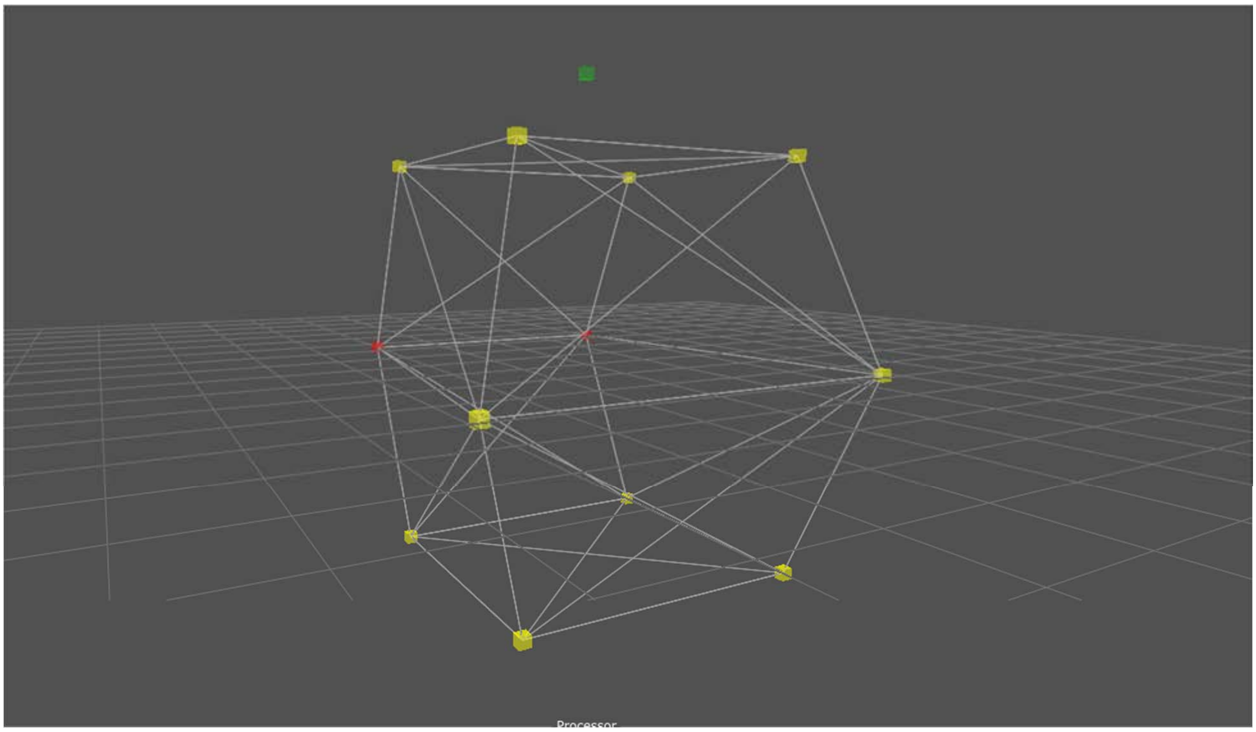

Figure 6: Stereo (LR) Configuration with Predefined Virtual Speakers

Figure 7 shows the virtual speaker setup for the C+L+R+Lss+Rss+Lsr+Rsr configuration. The figure displays virtual speakers in both the upper and lower hemispheres with yellow vertices. For this configuration, all the possible predefined virtual speakers in the upper and lower hemisphere are used.

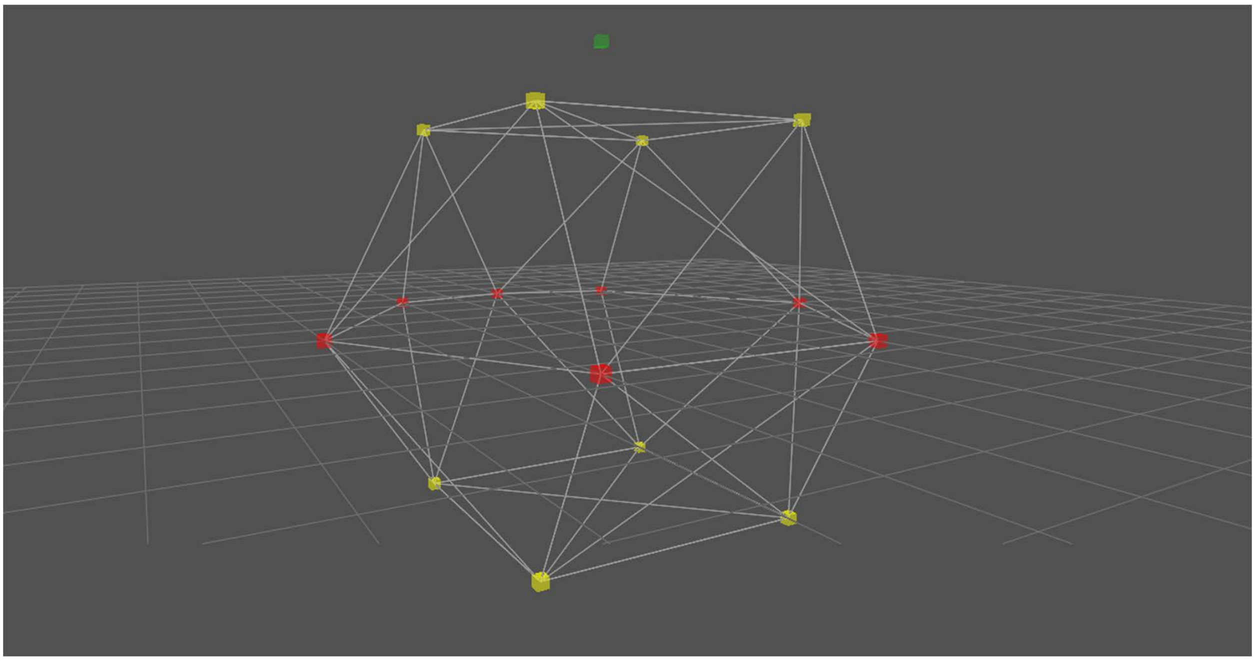

Figure 7: 7.x Output Configuration with Predefined Virtual Speakers

The fold-down coefficients for predefined virtual speakers vary depending on the output layout configuration. The predefined fold-down coefficients are tabulated in clause 7.3 for the equatorial plane and upper hemisphere, and in clause 7.4 for the lower hemisphere.

6.3.2.2 PredefinedVirtualSpeakers::IsSupported

The pseudocode below determines whether the `PredefinedVirtualSpeakers` algorithm can support the specified speaker configuration.

```
PredefinedVirtualSpeakers::IsSupported(SpeakerConfiguration SC)
{
    // Extract channel layout mask from speaker configuration
    speaker_layout_channel_mask = SC.GetChannelMask();

    // Mask away channels not needed for equatorial plane and upper hemisphere.
    channel_mask_for_eqtr_uphemi =
        GetChannelMaskForGroups(speaker_layout_channel_mask);

    for ( group_eqtr_uphemi : list_of_all_groups_equator_or_upper)
    {
        if (group_eqtr_uphemi.list_of_channel_masks.exist(channel_mask_for_eqtr_uphemi))
        {
            return true; // Supported if the channel mask exists
        }
    }
    return false; // Unsupported
}
```

6.3.2.3 PredefinedVirtualSpeakers::GetVirtualSpeakersAndFolddownCoeffs

The pseudocode below shows the logic of finding the virtual speakers and their fold-down coefficients. The input to the function is the physical speaker configuration.

```
PredefinedVirtualSpeakers::GetVirtualSpeakersAndFolddownCoeffs(
    SpeakerConfiguration config )
{
    // Extract channel layout mask from speaker configuration
    speaker_layout_channel_mask = config.GetChannelMask();
    // Extract speaker locations from speaker configuration
    speaker_locations = config.speaker_locations;

    channel_mask_for_eqtr_uphemi =
        GetChannelMaskForGroups(speaker_layout_channel_mask);
```

```
    // Find virtual sources and fold-down coefficients for equatorial and upper hemisphere.
    for ( group_eqtr_uphemi : list_of_all_groups_equator_or_upper)
    {
        if (group_eqtr_uphemi.list_of_channel_masks.exist(channel_mask_for_eqtr_uphemi))
        {
            V_e = group_eqtr_uphemi.eqtr_virt_srcs;
            V_u = group_eqtr_uphemi.uphemi_virt_srcs;

            FD_e = group_eqtr_uphemi.GetLinearGainMatrixEqtr( speaker_locations );
            FD_u = group_eqtr_uphemi.GetLinearGainMatrixUpHemi( speaker_locations );
        }
    }

    channel_mask_for_lower_hemi =
        GetChannelLayoutMaskForLowerHemisphere(physical_speaker_layout_channel_mask);

    // Find virtual sources and fold-down coefficients for lower hemisphere.
    for ( group_lohemi : list_of_all_groups_lower )
    {
        if ( group_lohemi.list_of_channel_masks.exist(channel_mask_for_lower_hemi) )
        {
            V_l = group_lohemi.lohemi_virt_srcs;

            FD_l = group_lohemi.GetLinearGainMatrixLoHemi( speaker_locations );
        }
    }

    // Creating a combined list for virtual speakers
    V = Join( V_e, V_u, V_l );

    // Creating a combined list of fold-down coefficients
    FD = Join( FD_e, FD_u, FD_l );

    // return both the list of virtual speakers and their fold-down coefficients
    return (V, FD);
}
```

6.3.3 Auto Virtual Speakers

6.3.3.1 Overview of Automatic Virtual Speaker Mapping

When the `SpeakerConfiguration` is not supported by the `PredefinedVirtualSpeaker` algorithm, then the `AutoVirtualSpeaker` algorithm is employed. There are multiple stages in this algorithm:

- Add virtual speakers in the equatorial plane
- Add virtual speakers in the upper hemisphere
- Add virtual speakers in the lower hemisphere
- Calculate fold-down coefficients for equatorial virtual speakers
- Calculate fold-down coefficients for the upper and lower hemisphere

6.3.3.2 Add Virtual Speakers in the Equatorial Plane

Virtual speakers are added in the equatorial plane if they are spaced more than 180 degrees apart when measured in the clockwise direction. They are added at 1/3 and 2/3 of the azimuth gap between the furthest speakers.

```
GetEquatorialVirtualSpeakers(SpeakerConfiguration SC)
{
    equatorial_points = GetEquatorialPoints(SC.speaker_locations);

    // Map azimuth to be between 0 and 360
    equatorial_points = MapAzimuthTo0_360(equatorial_points);

    // Sort in the ascending order of the azimuth.
    // Since elevation = 0 and radius = 1 for all points in the equator
    // of a unit sphere, the sort will order according to azimuth.
    sorted_equatorial_points = Sort(equatorial_points);
    // Repeat the first point
    sorted_equatorial_points += sorted_equatorial_points[0];
```

```
    // Find the maximum azimuth gap and the point that has this gap.
    max_gap = 0;
    max_gap_p = NULL;
    for ( i = 1; i < sorted_equatorial_points.size(); ++i )
    {
        gap = sorted_equatorial_points[i].azimuth - sorted_equatorial_points[i-1].azimuth;
        if ( max_gap < gap )
        {
            max_gap = gap;
            max_gap_p = sorted_equatorial_points[i-1];
        }
    }

    // If the gap is larger than 180, add virtual speakers in equatorial plane
    // at 1/3rd and 2/3rd points between the furthest speakers.
    V_e = [];

    if ( max_gap > 180 )
    {
        // wraps around if the angle goes beyond the range of -180 to 180.
        one_third_az = Wrap( max_gap_p + max_gap / 3.0, -180, 180 );
        two_third_az = Wrap( max_gap_p + 2 * max_gap / 3.0, -180, 180 );

        // Add virtual speaker at the 1/3rd gap
        V_e += { one_third_az, 0, 1 };
        // Add virtual speaker at the 2/3rd gap
        V_e += { two_third_az, 0, 1 };
    }
    return V_e;
}
```

Figure 8 illustrates the process of adding virtual speakers at 1/3rd and 2/3rd of the azimuth gap in the equatorial plane.

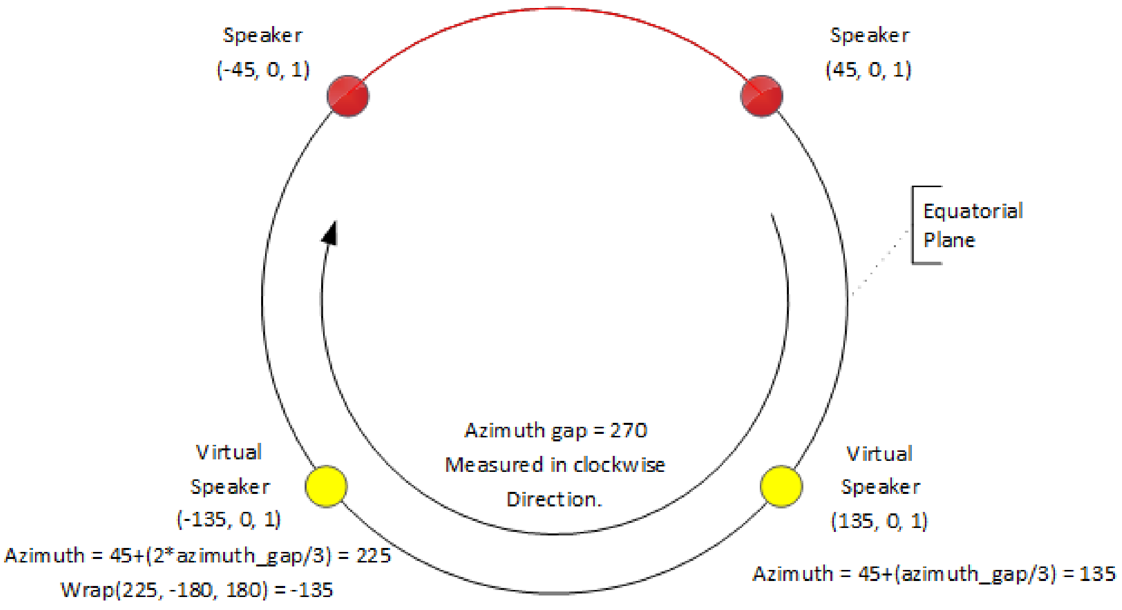

Figure 8: Adding Virtual Speakers in Equatorial Plane

6.3.3.3 Add Virtual Speakers in the Upper Hemisphere

Virtual speakers are added in the upper hemisphere only if there are no speakers with elevation above 0 degrees. If this condition is met, add a virtual speaker at an elevation of 45 degrees above every virtual and physical speaker in the equatorial plane.

```
GetUpperHemiVirtualSpeakers(SpeakerConfiguration SC, Point V_e[])
{
    equatorial_points = GetEquatorialPoints(SC.speaker_locations);

    // Combine both virtual and physical speakers in equatorial plane
    all_equatorial_points = Join( V_e, equatorial_points );
```

```
    // If there are no upper hemisphere speakers, add virtuals above every
    // equatorial plane speaker
    V_u = []
    if ( NumUpperHemispherePoints(SC.speaker_locations) == 0 )
    {
        for ( p : all_equatorial_points )
        {
            V_u += { p.azimuth, 45.0, p.radius };
        }
    }
    return V_u;
}
```

6.3.3.4 Add Virtual Speakers in the Lower Hemisphere

Virtual speakers are added in the lower hemisphere only if there are no speakers with elevation above 0 degrees. If this condition is met, add a virtual speaker below every virtual and physical speaker in the equatorial plane at an elevation of -45 degrees.

```
GetLowerHemiVirtualSpeakers(SpeakerConfiguration SC, Point V_e[])
{
    equatorial_points = GetEquatorialPoints(SC.speaker_locations);

    // Combine both virtual and physical speakers in equatorial plane
    all_equatorial_points = Join( V_e, equatorial_points );

    // If there are no lower hemisphere speakers, add virtuals below every
    // equatorial plane speaker
    V_l = []
    if ( NumLowerHemispherePoints(SC.speaker_locations) == 0 )
    {
        for ( p : all_equatorial_points )
        {
            V_l += { p.azimuth, 45.0, p.radius };
        }
    }
    return V_l;
}
```

Figure 9 shows the auto virtual speaker setup for the 7.x (C+L+R+Lss+Rss+Lsr+Rsr) output configuration. The yellow vertices indicate virtual speakers and the red vertices are the physical speakers.

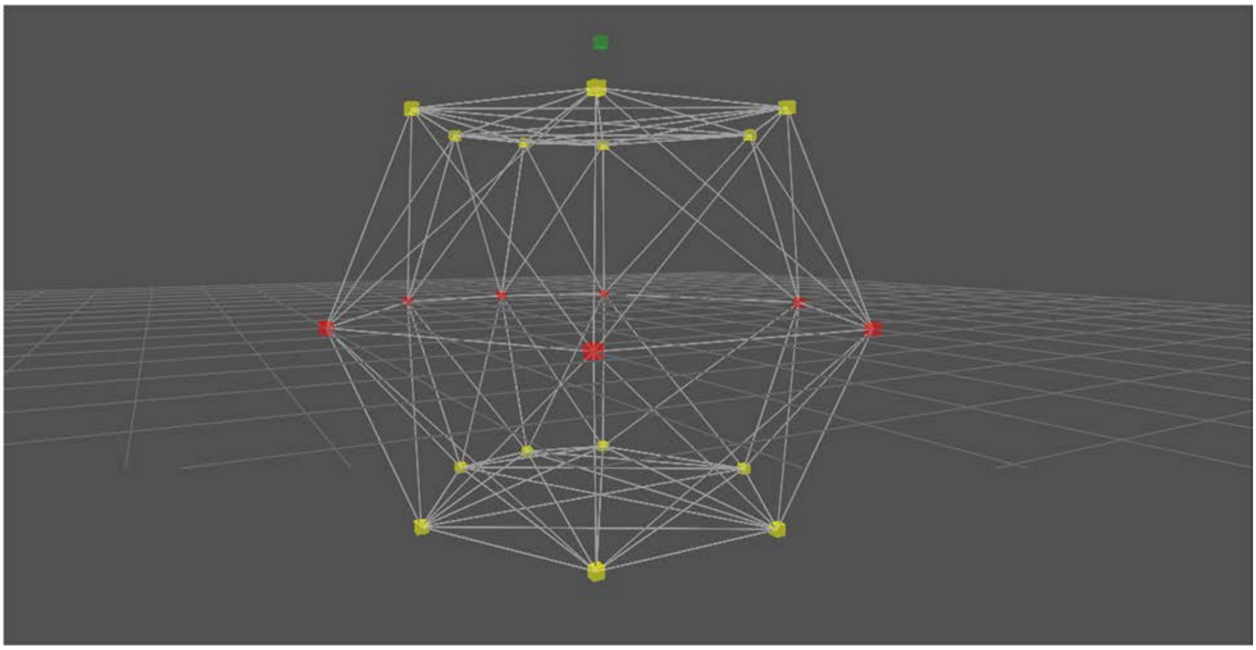

Figure 9: 7.x Output Configuration with Automatic Virtual Speakers

6.3.3.5 Calculate Fold-down Coefficients for Equatorial Virtual Speakers

If there are virtual speakers in the equatorial plane which were added at 1/3 and 2/3 of the azimuth gap between the furthest speakers, they are remapped between those physical speakers on the smaller arc on the equatorial circle and then panned pairwise to the new location for folding.

Remapping the azimuth of the virtual speaker between the physical speakers is done as shown in the pseudocode below:

```
// Remap equatorial virtual speaker (virt_az) between two furthest physical equatorial speakers
represented by min_az and max_az.
// The azimuths angles used for this function is wrapped to be between 0 to 360 (see
MapAzimuthTo0_360 for mapping logic).
RemapEquatorialVirtualSpkr (float virt_az, float min_az, float max_az )
{
    diff_az = max_az - min_az;
    return max_az - (((virt_az - max_az) / (360.0 - diff_az)) * diff_az);
}
```

Figure 10 shows the remapping of the virtual speakers (yellow) into the arc between physical speakers (red). The remapped positions are in green.

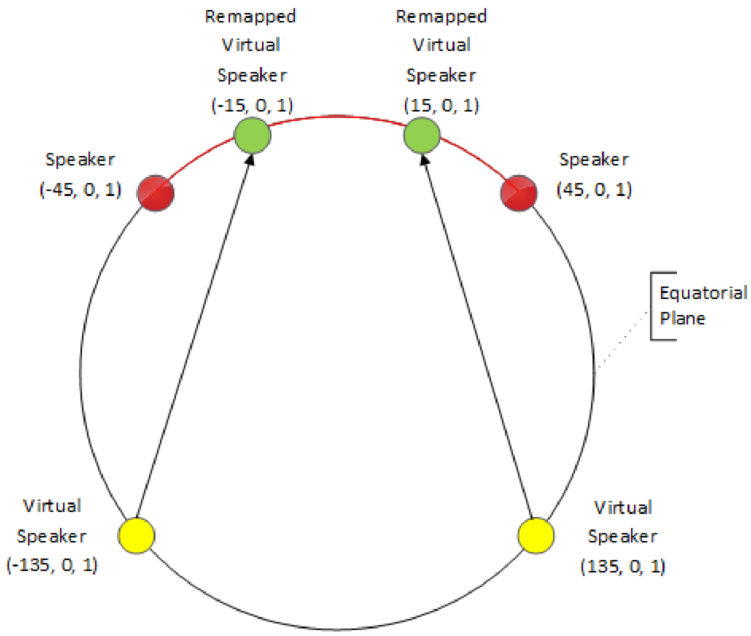

Figure 10: Remapping Virtual Speakers Between Physical Speakers

The algorithm for finding the fold-down coefficients for equatorial virtual speakers is shown below. Note that a hull object is used to determine the fold-down coefficients. A temporary hull object is constructed as shown in clause 6.3.4 to calculate the fold-down coefficients.

```
GetEquatorialFolddown(SpeakerConfiguration SC, Point V_e[], Hull H)
{
    FD_e = [][];

    // Calculate fold-down coefficients only if we have equatorial virtual speakers.
    if (V_e.size() > 0)
    {
        equatorial_points = GetEquatorialPoints(SC.speaker_locations);

        // Sort in the ascending order of the azimuth.
        // Since elevation = 0 and radius = 1 for all points in the equator
        // of a unit sphere, the sort will order according to azimuth.
        sorted_equatorial_points = Sort(equatorial_points);
        // Repeat the first point
        sorted_equatorial_points += sorted_equatorial_points[0];

        // Find the maximum azimuth gap and the points that have this gap.
        max_gap = 0;
        max_gap_p1 = NULL;
        max_gap_p2 = NULL;
        for ( i = 1; i < sorted_equatorial_points.size(); ++i )
```

```
        {
            gap = sorted_equatorial_points[i].azimuth - sorted_equatorial_points[i-1].azimuth;
            if ( max_gap < gap )
            {
                max_gap = gap;
                max_gap_p1 = sorted_equatorial_points[i-1];
                max_gap_p2 = sorted_equatorial_points[i];
            }
        }

        // Calculate the folding coefficients
        for ( p : V_e )
        {
            p_tmp = MapAzimuthTo0_360(p);
            min_p1p2 = min( max_gap_p1, max_gap_p2 );
            max_p1p2 = max( max_gap_p1, max_gap_p2 );
            // Find the remapped azimuth
            remap_az = RemapEquatorialVirtualSpkr ( p_tmp.azimuth, min_p1p2, max_p1p2);

            // Pan the new point
            new_p = { remap_az, p.elevation, p.radius };
            gain_v = H.Pan( new_p );

            // Extract gain coefficients for the physical speakers
            FD_e += gain_v[SC.speaker_locations];
        }
    }
    return FD_e;
}
```

6.3.3.6 Calculate Fold-down Coefficients for the Upper and Lower Hemisphere

The fold-down coefficients for folding the upper and lower hemisphere virtual speakers into a speaker configuration is calculated by first collapsing the virtual speakers into the equator with a small spread. The virtual speakers are dropped down (or pulled up) by moving every point in the upper or lower hemisphere virtual speakers to a new point at elevation equal to 0 degrees. A spread is added by splitting each point into 3 points on the equator with the new azimuth = $\{\theta,\ \theta + 45°,\ \theta - 45°\}$. Each of these are further scaled by { -3 dB, 0 dB, -3 dB }. This logic of calculating fold-down coefficients for the upper and lower hemisphere is illustrated in the pseudocode below. Note that a hull object is used to determine the fold-down coefficients. A temporary hull object is constructed to calculate the fold-down coefficients, as shown in clause 6.3.4.

```
GetNonEquatorialFolddown( SpeakerConfiguration SC, Point V_ul[], Point V_e float FD_e[][], Hull H)
{
    // Same algorithm works for both upper and lower hemisphere virtual speakers.
    FD_ul = [][];

    // Calculate fold-down only if virtual speakers exist
    if (V_ul.size() > 0)
    {
        float spread_angle[] = {-45, 0, 45};
        float spread_gain[] = {-3.0, 0.0, -3.0};

        for ( v : V_ul )
        {
            FD_v = [0] // 0 vector of size H.hull_points.size();
            for ( i : range(spread_angle.size()) )
            {
                p = { v.azimuth + spread_angle[i], 0, v.radius };
                // Calculate the hull gain vector
                hull_g = H.Pan(p);
                // Scale hull gain vector by the spread_gain
                mod_hull_g = hull_g * db2mag( spread_gain[i] );
                // Merge the hull gain with FD_v
                for ( h : H.hull_points )
                {
                    FD_v[h] = sqrt( pow( FD_v[h], 2 ) + pow( mod_hull_g[h], 2 ) );
                }
            }
            // The above folds virtual speakers from either
            // hemisphere to equatorial plane.
            // Now, fold the equatorial virtual speakers to physical speakers.
            for ( ve : V_e )
            {
                gain_ve = FD_v[ve]; // Extract gain contribution to 've'
                fd_ve = FD_e[ve]; // Extract fold-down vector for virtual speaker 've'
```

```
                for ( sl : SC.speaker_locations )
                    FD_v[sl] = FD_v[sl] + gain_ve * fd_ve[sl];
            }
            // Add the extracted gain coefficient to fold-down matrix
            FD_ul += FD_v[SC.speaker_locations];
        }
    }

    return FD_ul;
}
```

6.3.4 GetVirtualSpeakersAndFolddownCoeffs

The multiple steps described in clause 6.3.3.1 are combined in the implementation of `GetVirtualSpeakersAndFolddownCoeffs()`. A temporary hull object is created using the generated virtual speakers and the speaker configuration.

```
AutoVirtualSpeakers::GetVirtualSpeakersAndFolddownCoeffs(SpeakerConfiguration SC)
{
    // Get equatorial virtual speakers
    V_e = GetEquatorialVirtualSpeakers(SC);

    // Get upper hemisphere virtual speakers
    V_u = GetUpperHemiVirtualSpeakers(SC, V_e);

    // Get lower hemisphere virtual speakers
    V_l = GetLowerHemiVirtualSpeakers(SC, V_e)

    // List of all virtual points
    V = Union( V_e, V_u, V_l );

    // Create a temporary hull for finding fold-down coefficients
    hull_points = Union( V, SC.speaker_locations );
    Hull H = Hull( hull_points );

    // Get fold-down for equatorial virtual speakers
    FD_e = GetEquatorialFolddown(SC, V_e, H);

    // Get fold-down for upper hemisphere virtual speakers
    FD_u = GetNonEquatorialFolddown(SC, V_u, V_e, FD_e, H);

    // Get fold-down for lower hemisphere virtual speakers
    FD_l = GetNonEquatorialFolddown(SC, V_l, V_e, FD_e, H);

    // Combine all fold-down coefficients
    FD = Union( FD_e, FD_u, FD_l );

    return (V,FD);
}
```

6.3.5 Hull Initialization

Hull points are formed by combining physical and virtual speaker locations. At the initialization stage, a set of triplets are calculated, forming the surface mesh using the hull points. An exhaustive search is performed on all the triplets formed from the hull points in order to triangulate the hull. Each triplet is associated with an area on the positively oriented surface of the sphere. The positively oriented surface is the one facing away from the origin of the sphere. This triangulation is called hull initialization and is described below in the pseudocode. To summarize the pseudocode, it is finding all the triplets that do not have another speaker point within their associated surface. It does this by checking if there is another point `p_l` inside the surface area formed by `p_i`, `p_j` and `p_k`; and if no such point exists, then the triplet { `p_i`, `p_j`, `p_k` } is on the surface of the sphere and it is saved.

```
Hull::Init(Point hull_points[])
{
    // epsilon is used to handle floating point errors.
    // A generic epsilon value is used here, but should be changed depending on platform.
    epsilon = 1e-6;

    this->hull_points = hull_points;

    // Perform an exhaustive search to find the triplets
    // that form triangles of the hull.
```

```
    triplets_processed = [];
    for ( p_i : hull_points )
    {
        for ( p_j : hull_points[i+1, ...] )
        {
            for ( p_k : hull_points[j+1, ...] )
            {
                // Cannot form a triangle with two same points
                if ( p_i == p_j or p_j == p_k or p_i == p_k )
                    continue;

                // Construct a triplet
                Triplet t = { p_i, p_j, p_k };
                // Convert to be positively oriented
                t.ConvertToPositiveOrientation();

                // If the triplet is already processed, move on to the next triplet.
                if ( triplets_processed.contains(t) )
                    continue;

                // Avoid spanning hull's facets spanning hemispheres
                if ( !t.FullyInUpperHemisphere() && !t.FullyInLowerHemisphere() )
                    continue;

                Matrix M = t.GetTripletMatrix();
                // Avoid singular/degenerate matrices
                if ( !M.IsInvertible() )
                    continue;

                Matrix M_inv = M.Invert();

                is_facet = true;
                for ( p_l : hull_points )
                {
                    // Test vector should also be in the same hemisphere as the triplet
                    if ( !t.FullyInEquatorialPlane() &&
                        t.FullyInUpperHemisphere() && p_l.elevation < 0 )
                        continue;
                    if ( !t.FullyInEquatorialPlane() &&
                        t.FullyInLowerHemisphere() && p_l.elevation > 0 )
                        continue;

                    g = M_inv * p_l.Cartesian();

                    // This checks if the point p_l exists outside of the plane formed
                    // by p_i, p_j & p_k away from origin. The sum of gains will be > 1
                    // only for a point lying above the plane but within the triangle
                    // formed by p_i, p_j & p_k.
                    if (g[0] + g[1] + g[2] > 1 + epsilon)
                    {
                        is_facet = false;
                        break;
                    }
                }

                // If the triplet is a face, add it hull triplets
                if ( is_facet )
                    this->T += t;

                // Mark triplet as processed
                triplets_processed += t;
            } // for p_k
        } // for p_j
    } // for p_i
}
```

6.4 Predefined Virtual Speakers Organization

The predefined virtual speaker assignments and corresponding fold-down coefficients are organized by the channel bitmask. There are 21 non-overlapping groups of channel bitmasks for the equatorial plane and/or upper hemisphere regions that have defined virtual speakers with corresponding fold-down coefficients. These groups are defined in clause 7.3.

A subset of the channel masks that include speakers in the equatorial plane also have predefined virtual speakers in the lower hemisphere. This subset is further organized into five non-overlapping groups. These groups are defined in clause 7.4.

6.5 Rendering All Objects

6.5.1 Object Rendering

Rendering all objects involves panning all the objects in the specified `SpeakerConfiguration`, followed by smoothing and mixing them into the output feeds.

The pseudocode below illustrates the implementation of `RenderAllObjects()`, which renders one `AudioObject` at a time. Similarly, `RenderAudioObject()` renders one `MonoObject` at a time.

```
PointSourceObjectRenderer::RenderAllObjects(AudioObject AO_list[], uint block_size)
{
    // Prepare output buffers for rendering
    out_waveforms = float[SC.speaker_locations.size()][block_size];
    for (AudioObject AO : AO_list)
    {
        RenderAudioObject(AO, block_size);
    }
    return out_waveforms;
}

PointSourceObjectRenderer::RenderAudioObject(AudioObject AO, uint block_size)
{
    for (MonoObject MO : AO.mono_objects)
    {
        RenderMonoObject(MO, block_size, AO.obj_gain, AO.obj_lambda);
    }
};
```

A mono object is defined as a single waveform with an associated list of point source locations. To render a mono object, each of the point source locations is panned into the speaker configuration. The gain vectors of all the points are summed together to get a waveform's gain vector. It is then smoothed with the previous block's gain vector and applied on the waveform to render it into the output feeds. The pseudocode for `RenderMonoObject()` is below.

```
PointSourceObjectRenderer::RenderMonoObject(
MonoObject MO,
uint block_size,
float obj_gain,
float obj_lambda)
{
    // Initialize MonoObject gain array.
    float MO_gains[SC.speaker_locations.size()];

    for ( Point p : MO.positions )
    {
        // Calculate gains from the panner.
        float gains[] = VBP.Pan(p);
        for ( Point sl : SC.speaker_locations )
        {
            // Apply object gain to the VBP gains and accumulate into the MonoObject's gain.
            MO_gains[sl] += gains[sl] * obj_gain;
        }
    }
    for ( Point sl : SC.speaker_locations )
    {
        // set dynamic lambda for the smoothers.
        MO.smoother_list[sl].lambda = obj_lambda;
        // Run the smoothing filter.
        MO_smoothed_gain_over_a_block = MO.smoother_list[sl].Smooth(MO_gains[sl], block_size);
        for ( uint t : 0 to block_size )
        {
            // Apply smoothed gains to the MonoObject waveforms to render.
            out_waveforms[sl][t] += MO_smoothed_gain_over_a_block[t] * MO.waveform[t] * obj_gain;
        }
    }
}
```

6.5.2 Vector Based Panner

The vector based panner takes a point and pans it to the speaker configuration. VBP manages the hull, virtual speakers and their fold-down coefficients. The panning algorithm first pans the point to the hull and then fold-down coefficients are applied to the panned gain coefficients in the specified speaker configuration. The gain coefficients are normalized to preserve power. The pseudocode below implements this algorithm.

```
VectorBasedPanner::Pan(Point P)
{
    // Pan to hull
    gain_H = H.Pan(p);

    // Fold-down
    gain = [0...SC.speaker_locations] = 0; // zero vector
    for ( v : V ) // for each virtual speaker
    {
        gv = gain_H[v];
        for ( sl : SC.speaker_locations )
        {
            gain[sl] += gv * FD[v][sl];
        }
    }
    // normalize
    gain /= sqrt( gain.Dot(gain) );
    return gain;
}
```

6.5.3 Panning in a Hull

After the hull triplets are constructed (see clause 6.3.5), the panner is ready to pan any point on the surface of the sphere. For a given `Hull`, the panner renders a point source location in that hull by first finding set of triplets that contain the point source location and then pan the point in those triplets using `VectorBasedPanner`. The renderer uses vector based amplitude panning, which extends the well-known tangent law for pairwise panning on a plane to three-dimensions (see [2]). Figure 11 shows panning of a point source location into 7.x (C+L+R+Lss+Rss+Lsr+Rsr) speaker configuration. The red lines are the contribution line to each of the point in the triplet. The point source location (green vertex) is panned to a triplet within the 7.x output configuration. The white line is a vector in the direction of the point source from the origin.

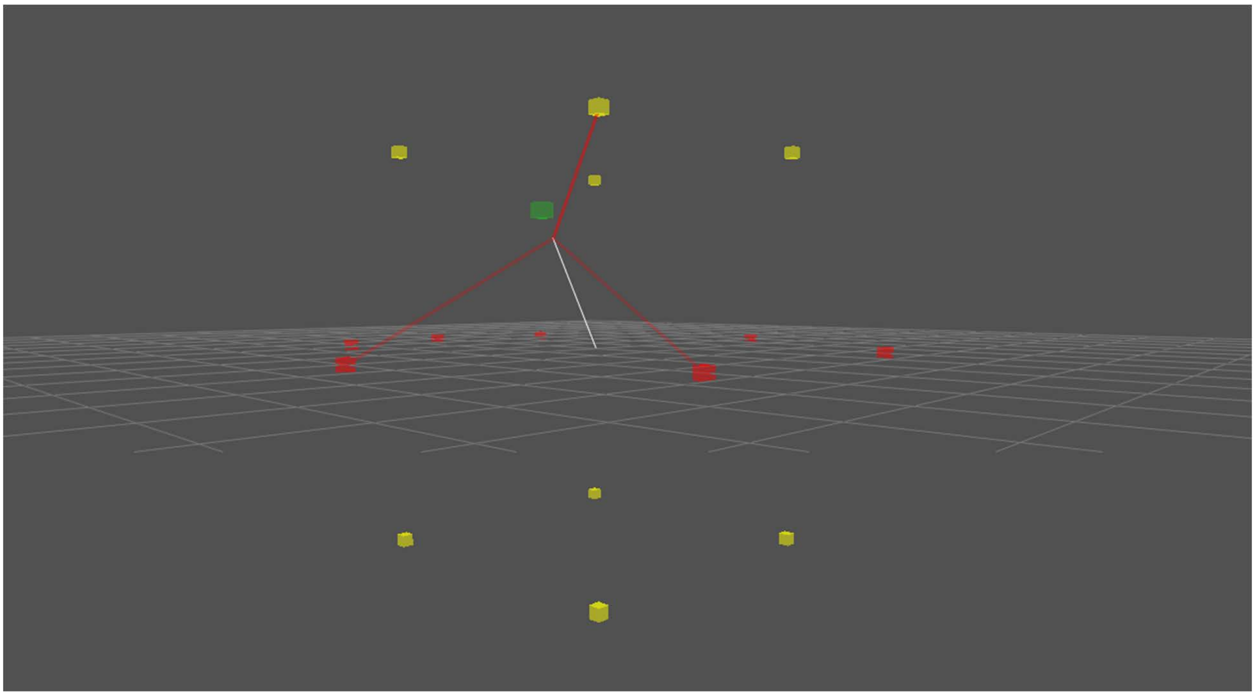

Figure 11: 7.x Speaker Configuration Point Source Location Panning

Multiple triplets can contain the same point source locations because of the following conditions:

- The point may lie on a speaker where it belongs to all the triplets formed with that speaker; or
- The point may be on the edge between two triplets; or
- The point may be inside an n-patch, a three-dimensional polygon that has overlapping triangles.

To address such cases, all the triplets are used and the gains are averaged over all the triplets. Figure 12 shows how the panner distributes gains for a point source location which spans over multiple triplets. The point source location (green vertex) is panned to a 4-patch within the 7.x output configuration.

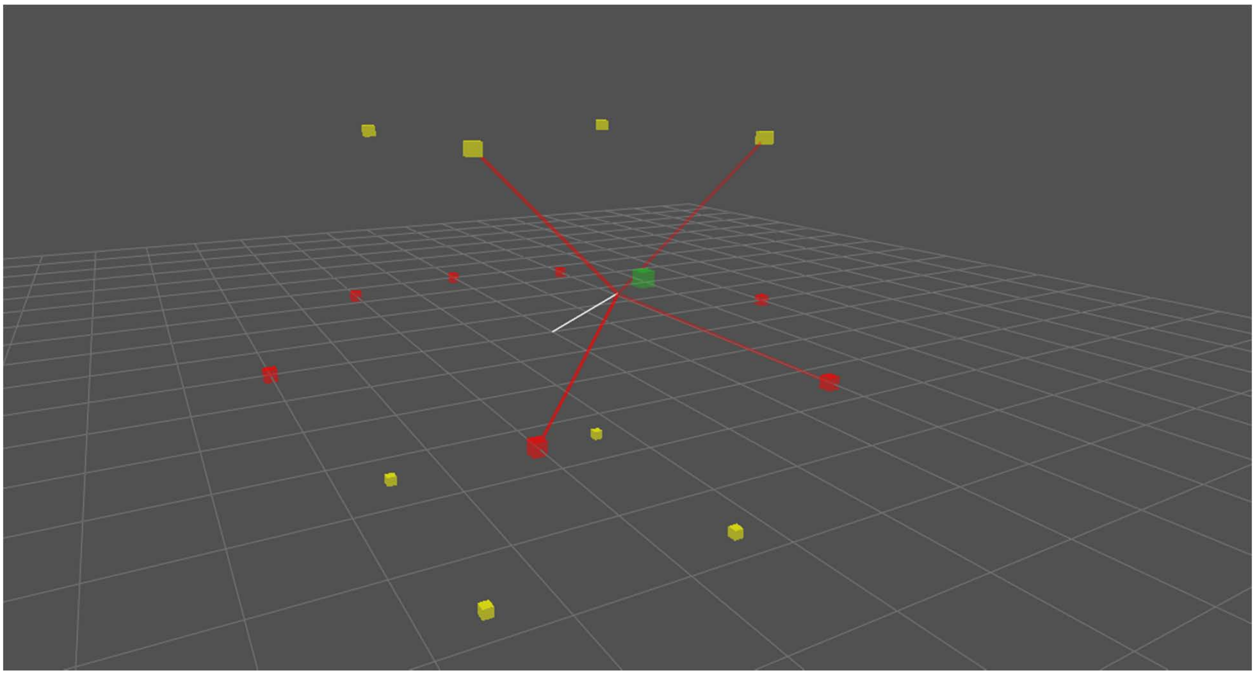

Figure 12: Point Source Panner Distribution of Gains

When a point is located on the edge between two triplets, in addition to the two triplets sharing the edge, it is possible that there may be a third triplet overlapping these two triplets fully containing the point. In such cases, it is necessary to prevent the speakers on the edge from getting a boost over the other speakers. That is, instead of computing the gain as $gain = (2 * edge + triplet)/3$ it is calculated as $gain = (edge + triplet)/2$. In this manner, a moving point source location has continuous gain changes. To handle this, the panning algorithm treats edge cases differently by storing them and ensuring that other triplets in the edge will not be used for panning.

Note that the panning pseudocode below uses a generic value for `epsilon`, initialized in `Hull::Init()`, to handle the floating point errors. An actual implementation may need to change this value or handle floating point errors differently.

Given a point, panning to the hull is implemented in the pseudocode below.

```
Hull::Pan(Point p)
{
     // gain vector for all points in the hull; initialized to 0
    gain_H = [0...hull_points.size()];

    // List of all the significant edges that were panned.
    all_significant_edge_list = [];

    // List of all the edges already panned.
    all_panned_edge_list = [];

    triplet_count = 0;
    for ( t : T ) // For all triplets
    {
        Matrix M_inv = t.GetTripletMatrix().Inverse();

        // Calculate gain vector by panning p to this triplet
        Vector g = M_inv * p.Cartesian();
```

```
        // Check if the point p lies in this triplet
        if ( g[0] >= -epsilon && g[1] >= -epsilon && g[2] >= -epsilon )
        {
            if (g[0] < 0) g[0] = 0;
            if (g[1] < 0) g[1] = 0;
            if (g[2] < 0) g[2] = 0;

            significance_count = ( g[0] > epsilon ) + ( g[1] > epsilon ) + ( g[2] > epsilon );

            // Special case when the point lies on the edge of a triangle.
            // Since this edge would appear in two triangles, doubling the contribution
            // of the edge. To avoid this, we ensure that a significant edge is not panned twice.
            if ( significance_count == 2 ) {
                Edge significant_edge;
                if (g[0] > epsilon && g[1] > epsilon)
                    significant_edge = Edge(t.p1, t.p2);
                else if (g[0] > epsilon && g[2] > epsilon)
                    significant_edge = Edge(t.p1, t.p3);
                else
                    significant_edge = Edge(t.p2, t.p3);

                // If the significant edge is already panned then move on to the next triplet.
                if ( all_panned_edge_list.exist(significant_edge) )
                    continue;

                // Add significant edge to the special list
                all_significant_edge_list += significant_edge;
            }

            // Triplets' edges
            Edge triplet_edge_list[] = { Edge(t.p1, t.p2), Edge(t.p1, t.p3), Edge(t.p2, t.p3) };

            // Special case when any of the edge has been already marked as significant.
            // This would occur due floating point errors and hence, once an edge is marked
            // significant, it remains significant. And so, we shall render that edge only once.
            edge_is_significant_and_panned = false;
            for ( Edge e : triplet_edge_list )
                if (all_significant_edge_list.exist(e) && all_panned_edge_list.exist(e)) {
                    edge_is_significant_and_panned = true;
                    break;
                }
            if ( edge_is_significant_and_panned )
                continue; // if edge is significant and already panned, move on to next triplet.

            // Add the edges in the triplet to the list of panned edges.
            all_panned_edge_list += triplet_edge_list;

            // Add the triplet contribution to hull gain
            gain_H[ t.p1 ] += g[0];
            gain_H[ t.p2 ] += g[1];
            gain_H[ t.p3 ] += g[2];
            ++triplet_count;
        }
    }

    // Normalize the gains
    gain_H /= triplet_count;

    return gain_H;
}
```

7 Predefined Virtual Speaker Mapping

7.1 Channel Bitmask

The channel layouts defined in clause 7.3 and clause 7.4 contain the entire set supported by the predefined virtual speaker mode. The channel masks are specified by the channel layout bitmask which is formed by taking bitwise disjunction of individual channels' bitmasks as defined in Table 3.

Note that there is some exclusion of bitmasks while listing the channel masks. For example, in the clauses of 7.3, the LFE channels are not included in the channel mask listing. In clause 7.4, the LFE channels and all of the upper hemisphere channels are excluded from the channel mask listing. See clause 7.3.1.1 and clause 7.4.2.1 for the channel mask conversion functions.

Table 3: Channel Definitions and Bitmask

Channel Description	Abbreviation	Azimuth (degrees)	Elevation (degrees)	Channel Bitmask
Centre	C	0,0	0,0	0x00000001
Left	L	-30,0	0,0	0x00000002
Right	R	30,0	0.0	0x00000004
Left Surround	Ls	-110,0	0,0	0x00000008
Right Surround	Rs	110,0	0,0	0x00000010
Low Frequency Effects - 1	LFE1	0,0	0,0	0x00000020
Centre Surround	Cs	180,0	0,0	0x00000040
Left Surround Rear	Lsr	-150,0	0,0	0x00000080
Right Surround Rear	Rsr	150,0	0,0	0x00000100
Left Side Surround	Lss	-90,0	0,0	0x00000200
Right Side Surround	Rss	90,0	0,0	0x00000400
Left Centre	Lc	-15,0	0,0	0x00000800
Right Centre	Rc	15,0	0,0	0x00001000
Left High	Lh	-30,0	45,0	0x00002000
Centre High	Ch	0,0	45,0	0x00004000
Right High	Rh	30,0	45,0	0x00008000
Low Frequency Effects - 2	LFE2	0,0	0,0	0x00010000
Left Wide	Lw	-60,0	0,0	0x00020000
Right Wide	Rw	60,0	0,0	0x00040000
Overhead	Oh	0,0	90,0	0x00080000
Left High Side	Lhs	-90,0	45,0	0x00100000
Right High Side	Rhs	90,0	45,0	0x00200000
Centre high Rear	Chr	180,0	45,0	0x00400000
Left High Rear	Lhr	-150,0	45,0	0x00800000
Right High Rear	Rhr	150,0	45,0	0x01000000
Centre Low Front	Clf	0,0	-30,0	0x02000000
Left Low Front	Llf	-30,0	-30,0	0x04000000
Right Low Front	Rlf	30,0	-30,0	0x08000000
Left Top Front	Ltf	-45,0	60,0	0x10000000
Right Top Front	Rtf	45,0	60,0	0x20000000
Left Top Rear	Ltr	-135,0	60,0	0x40000000
Right Top Rear	Rtr	135,0	60,0	0x80000000

7.2 General Mapping Functions

7.2.1 GetLinearGainVectorFromTable

This function converts the fold-down coefficients tables defined in clause 7.3 and clause 7.4 into gain vectors.

```
// Get fold-down coefficients for a virtual point in linear scale.
GetLinearGainVectorFromTable(
    GainCoefficient_dB table,
    FunctionP mapping_functions[],
    uint virtual_p_idx,
    Point speaker_locations[] )
{
    out_gain_vector = []
    for ( i : range(speaker_locations.size()) )
    out_gain_vector[i] = 0;

    for ( func_p : mapping_functions )
    out_gain_vector[func_p()] = db2mag(table[in_virtual_p_idx][func_p()]);

    return out_gain_vector;
}
```

7.2.2 Physical Speaker Search Functions

7.2.2.1 Search Functions Overview

A physical speaker is identified by the location of the speaker in the listening space. This location can be specified as a tuple of azimuth θ, elevation φ and radius r. As with the object positions, azimuth varies from -180 degrees to 180 degrees; a negative angle indicates speaker on the left hemisphere and the positive angle indicates speaker on the right hemisphere. The elevation varies from -90 degrees to 90 degrees; where a negative angle indicates lower hemisphere and the positive angle indicates upper hemisphere. A virtual speaker is defined in the same manner as a physical speaker, but they do not physically exist in the listening space. The following functions are used to identify the physical speaker location. All of the functions listed below are using the set of physical speaker locations, `SC.speaker_locations`.

7.2.2.2 ClosestAzEq

`ClosestAzEq(θ)` returns the physical speaker index from the set `SC.speaker_locations[]` that meets two conditions:

1) It is located in equatorial plane ($\varphi = 0$).

2) It is closest to the specified azimuth (θ).

7.2.2.3 ClosestAzUH

`ClosestAzUH(θ)` returns the physical speaker index from the set `SC.speaker_locations[]` that meets two conditions:

1) It is located in the equatorial plane or upper hemisphere ($\varphi \geq 0$).

2) It is closest to the specified azimuth (θ).

7.2.2.4 ClosestElv

`ClosestElv(φ, S)` returns the physical speaker index from the set `S`, which is a subset of `SC.speaker_locations[]` whose elevation is closest to the specified elevation φ.

7.2.2.5 LeftLargestEqtr

`LeftLargestEqtr()` returns the physical speaker index from set `SC.speaker_locations[]` that meets two conditions:

1) It is located on the equatorial plane ($\varphi = 0$).

2) It has largest magnitude azimuth in the left hemisphere ($0 > \theta > -180°$).

7.2.2.6 RightLargestEqtr

`RightLargestEqtr()` returns the physical speaker index from set in `SC.speaker_locations[]` that meets two conditions:

1) It is located on the equatorial plane ($\varphi = 0$).

2) It has largest azimuth in the right hemisphere ($0 < \theta < 180°$).

7.2.2.7 PrefOrder

`PrefOrder(Spkr1, Spkr2,...)` is used to specify the preference order of speakers to contribute to. It returns the index of the first speakers from the input ordered speaker list that exist in the physical speakers set. It is used in cases where all of `Spkr1, Spkr1+ Spkr2 & Spkr2` are possible output layouts. To resolve such cases, the contribution preference order is used, `Spkr1` gets the contribution if it is present or else `Spkr2` gets the contribution, and so on. Note that the function takes an arbitrary number of arguments; as such, any number of speakers may be used while calling this function. Speakers can also be identified by calling other search functions.

7.2.2.8 Split

`Split(Spkr1,Spkr2,…,Gain_dB)` is used to specify the split of contribution between speakers `Spkr1, Spkr2 … SpkrN by Gain_dB` gain. The splitting only happens if all the speakers in the list exist.

7.2.2.9 ExclusiveOr

`ExclusiveOr(Spkr1, Spkr2, …)` is used to specify disjunction choice of the speakers. In such cases only one of the speakers exists and hence, apply the contribution specified to that speaker.

7.3 Equatorial Plane and Upper Hemisphere

7.3.1 Equatorial Plane and Height Functions

7.3.1.1 GetChannelMaskForEqtrUpHemiGroups

This function masks the LFE channels to zero from the physical channel layouts speakers.

```
GetChannelMaskForEqtrUpHemiGroups( physical_speaker_channel_layout_mask )
{
    // Defining masking flags that will be masked to zero for channel layout groups.
    mask_lfe = LFE1 | LFE2;

    // Take negation of lfe masks
    mask_to_zero = (~mask_lfe);

    // Apply conjunction to make lfe's channel masks zero.
    channel_layout_mask_for_groups =
    physical_speaker_channel_layout_mask & mask_to_zero;

    // return the resultant channel mask.
    return channel_layout_mask_for_groups;
}
```

7.3.1.2 GroupEquatorOrUpper

The following structure represents a group in the equatorial plane or the upper hemisphere virtual speakers.

```
struct GroupEquatorOrUpper
{
    // An array listing all the channel masks supported by the group.
    uint32 list_of_channel_masks[];

    // An array of equatorial virtual speakers used by the group.
    Point eqtr_virt_srcs[];
    // An array of upper hemisphere virtual speakers used by the group.
    Point uphemi_virt_srcs[];

    // An array of function pointers to determine physical speaker index.
    FunctionP eqtr_functions[];
    // An array of function pointers to determine upper hemisphere virtual speaker index.
    FunctionP uphemi_functions[];

    // Table of fold-down coefficients for the equatorial virtual speakers.
    GainCoefficient_dB eqtr_folddown[][];
    // Table of fold-down coefficients for the upper hemisphere virtual speakers.
    GainCoefficient_dB uphemi_folddown[][];
```

```
    // Get linear gain vector for virtual sources in equatorial plane
    GetLinearGainMatrixEqtr(Point in_output_locations[]);
    // Get linear gain vector for virtual sources in upper hemisphere
    GetLinearGainMatrixUpHemi(Point speaker_locations[]);
}

GroupEquatorOrUpper::GetLinearGainMatrixEqtr(Point speaker_locations[])
{
    out_gain_matrix = [][];

    for ( virtual_p_idx : range(eqtr_virt_srcs.size()) )
    {
        out_gain_matrix[virtual_p_idx] =
            GetLinearGainVectorFromTable(
                eqtr_virt_srcs,
                eqtr_functions,
                virtual_p_idx,
                speaker_locations );
    }
    return out_gain_matrix;
}

GroupEquatorOrUpper:: GetLinearGainMatrixUpHemi(Point speaker_locations[])
{
    out_gain_matrix = [][];

    for ( virtual_p_idx : range(uphemi_virt_srcs.size()) )
    {
        out_gain_matrix[virtual_p_idx] =
            GetLinearGainVectorFromTable(
                uphemi_virt_srcs,
                uphemi_functions,
                virtual_p_idx,
                speaker_locations );
    }
    return out_gain_matrix;
}
```

7.3.1.3 list_of_all_groups_equator_or_upper

The following list defines all the supported groups for custom virtual speakers in the upper hemisphere and the equatorial plane.

```
list_of_all_groups_equator_or_upper = {
    GroupEquatorOrUpper1,
    GroupEquatorOrUpper2,
    …
    GroupEquatorOrUpper21
};
```

7.3.2 GroupEquatorOrUpper1

7.3.2.1 Channel Layouts in GroupEquatorOrUpper1

Channel layout bitmasks that use `GroupEquatorOrUpper1` fold-down coefficients are listed in Table 4.

Table 4: GroupEquatorOrUpper1 list_of_channel_masks[]

0x00000006	0x00001800	0x00001E00	0x00060000	0x00060600	0x00061800	0x00061E00
0x00000007	0x00001801	0x00001E01	0x00060001	0x00060601	0x00061801	0x00061E01
0x00000606	0x00001806	0x00001E06	0x00060006	0x00060606	0x00061806	0x00061E06
0x00000607	0x00001807	0x00001E07	0x00060007	0x00060607	0x00061807	0x00061E07

7.3.2.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper1

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 5.

Table 5: GroupEquatorOrUpper1 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	LeftLargestEqtr()	RightLargestEqtr()
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]	
(-135, 0, 1)	0 dB	
(135, 0, 1)		0 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 6.

Table 6: GroupEquatorOrUpper1 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	ClosestAzEq(-40)	ClosestAzEq(40)	RightLargestEqtr()	RightLargestEqtr()
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]			
(-45, 45, 1)	0 dB			
(45, 45, 1)		0 dB		
(-135, 45, 1)			0 dB	
(135, 45, 1)				0 dB

Fold-down coefficients for virtual speakers in lower hemisphere are defined in clauses 7.4.3 and 7.4.4.

7.3.3 GroupEquatorOrUpper2

7.3.3.1 Channel Layouts in GroupEquatorOrUpper2

Channel layout bitmasks that use `GroupEquatorOrUpper2` fold-down coefficients are listed in Table 7.

Table 7: GroupEquatorOrUpper2 list_of_channel_masks[]

0x01800006	0x01861E00	0x01C61800	0xC0060600	0xC0460000	0xC1801E00	0xC1C01800
0x01800007	0x01861E01	0x01C61801	0xC0060601	0xC0460001	0xC1801E01	0xC1C01801
0x01800606	0x01861E06	0x01C61806	0xC0060606	0xC0460006	0xC1801E06	0xC1C01806
0x01800607	0x01861E07	0x01C61807	0xC0060607	0xC0460007	0xC1801E07	0xC1C01807
0x01801800	0x01C00006	0x01C61E00	0xC0061800	0xC0460600	0xC1860000	0xC1C01E00
0x01801801	0x01C00007	0x01C61E01	0xC0061801	0xC0460601	0xC1860001	0xC1C01E01
0x01801806	0x01C00606	0x01C61E06	0xC0061806	0xC0460606	0xC1860006	0xC1C01E06
0x01801807	0x01C00607	0x01C61E07	0xC0061807	0xC0460607	0xC1860007	0xC1C01E07
0x01801E00	0x01C01800	0xC0000006	0xC0061E00	0xC0461800	0xC1860600	0xC1C60000
0x01801E01	0x01C01801	0xC0000007	0xC0061E01	0xC0461801	0xC1860601	0xC1C60001
0x01801E06	0x01C01806	0xC0000606	0xC0061E06	0xC0461806	0xC1860606	0xC1C60006
0x01801E07	0x01C01807	0xC0000607	0xC0061E07	0xC0461807	0xC1860607	0xC1C60007
0x01860000	0x01C01E00	0xC0001800	0xC0400006	0xC0461E00	0xC1861800	0xC1C60600
0x01860001	0x01C01E01	0xC0001801	0xC0400007	0xC0461E01	0xC1861801	0xC1C60601
0x01860006	0x01C01E06	0xC0001806	0xC0400606	0xC0461E06	0xC1861806	0xC1C60606
0x01860007	0x01C01E07	0xC0001807	0xC0400607	0xC0461E07	0xC1861807	0xC1C60607
0x01860600	0x01C60000	0xC0001E00	0xC0401800	0xC1800006	0xC1861E00	0xC1C61800
0x01860601	0x01C60001	0xC0001E01	0xC0401801	0xC1800007	0xC1861E01	0xC1C61801
0x01860606	0x01C60006	0xC0001E06	0xC0401806	0xC1800606	0xC1861E06	0xC1C61806
0x01860607	0x01C60007	0xC0001E07	0xC0401807	0xC1800607	0xC1861E07	0xC1C61807
0x01861800	0x01C60600	0xC0060000	0xC0401E00	0xC1801800	0xC1C00006	0xC1C61E00
0x01861801	0x01C60601	0xC0060001	0xC0401E01	0xC1801801	0xC1C00007	0xC1C61E01
0x01861806	0x01C60606	0xC0060006	0xC0401E06	0xC1801806	0xC1C00606	0xC1C61E06
0x01861807	0x01C60607	0xC0060007	0xC0401E07	0xC1801807	0xC1C00607	0xC1C61E07

7.3.3.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper2

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 8.

Table 8: GroupEquatorOrUpper2 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	LeftLargestEqtr()	RightLargestEqtr()	PrefOrder(Lhr,Ltr)	PrefOrder(Rhr,Rtr)
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]			
(-135,0,1)	-3,0103 dB		-3,0103 dB	
(135,0,1)		-3,0103 dB		-3,0103 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 9.

Table 9: GroupEquatorOrUpper2 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	ClosestAzEq(-40)	ClosestAzEq(40)	PrefOrder(Ltr,Lhr)	PrefOrder(Rtr,Rhr)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]			
(-45,45,1)	-3,0103 dB		-3,0103 dB	
(45,45,1)		-3,0103 dB		-3,0103 dB

7.3.4 GroupEquatorOrUpper3

7.3.4.1 Channel Layouts in GroupEquatorOrUpper3

Channel layout bitmasks that use `GroupEquatorOrUpper3` fold-down coefficients are listed in Table 10.

Table 10: GroupEquatorOrUpper3 list_of_channel_masks[]

0x00404006	0x00405800	0x00405E00	0x00464000	0x00464600	0x00465800	0x00465E00
0x00404007	0x00405801	0x00405E01	0x00464001	0x00464601	0x00465801	0x00465E01
0x00404606	0x00405806	0x00405E06	0x00464006	0x00464606	0x00465806	0x00465E06
0x00404607	0x00405807	0x00405E07	0x00464007	0x00464607	0x00465807	0x00465E07

7.3.4.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper3

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 11.

Table 11: GroupEquatorOrUpper3 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	LeftLargestEqtr()	RightLargestEqtr()	Chr
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]		
(-135,0,1)	-3,0103 dB		-3,0103 dB
(135,0,1)		-3,0103 dB	-3,0103 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 12.

Table 12: GroupEquatorOrUpper3 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	ClosestAzEq(-40)	ClosestAzEq(40)	Ch
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]		
(-45,45,1)	-3,0103 dB		-3,0103 dB
(45,45,1)		-3,0103 dB	-3,0103 dB

7.3.5 GroupEquatorOrUpper4

7.3.5.1 Channel Layouts in GroupEquatorOrUpper4

Channel layout bitmasks that use `GroupEquatorOrUpper4` fold-down coefficients are listed in Table 13.

Table 13: GroupEquatorOrUpper4 list_of_channel_masks[]

0x00300006	0x00301800	0x00301E00	0x00360000	0x00360600	0x00361800	0x00361E00
0x00300007	0x00301801	0x00301E01	0x00360001	0x00360601	0x00361801	0x00361E01
0x00300606	0x00301806	0x00301E06	0x00360006	0x00360606	0x00361806	0x00361E06
0x00300607	0x00301807	0x00301E07	0x00360007	0x00360607	0x00361807	0x00361E07

7.3.5.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper4

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 14.

Table 14: GroupEquatorOrUpper4 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	PrefOrder (Lss,Lhs,Ltm)	PrefOrder (Rss,Rhs,Rtm)
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]	
(-135,0,1)	0 dB	
(135,0,1)		0 dB

Table 15: GroupEquatorOrUpper4 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	ClosestAzEq(-40)	ClosestAzEq(40)	ExclusiveOr (Lhs,Ltm)	ExclusiveOr (Rhs,Rtm)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]			
(-45,45,1)	-3,0103 dB		-3,0103 dB	
(45,45,1)		-3,0103 dB		-3,0103 dB

7.3.6 GroupEquatorOrUpper5

7.3.6.1 Channel Layouts in GroupEquatorOrUpper5

Channel layout bitmasks that use `GroupEquatorOrUpper5` fold-down coefficients are listed in Table 16.

Table 16: GroupEquatorOrUpper5 list_of_channel_masks[]

0x00400006	0x00401800	0x00401E00	0x00460000	0x00460600	0x00461800	0x00461E00
0x00400007	0x00401801	0x00401E01	0x00460001	0x00460601	0x00461801	0x00461E01
0x00400606	0x00401806	0x00401E06	0x00460006	0x00460606	0x00461806	0x00461E06
0x00400607	0x00401807	0x00401E07	0x00460007	0x00460607	0x00461807	0x00461E07

7.3.6.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper5

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 17.

Table 17: GroupEquatorOrUpper5 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	LeftLargestEqtr()	RightLargestEqtr()	Chr
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]		
(-135,0,1)	-12,3045 dB		-0,2633 dB
(135,0,1)		-12,3045 dB	-0,2633 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 18.

Table 18: GroupEquatorOrUpper5 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	ClosestAzEq(-40)	ClosestAzEq(40)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]	
(-45,45,1)	0 dB	
(45,45,1)		0 dB

7.3.7 GroupEquatorOrUpper6

7.3.7.1 Channel Layouts in GroupEquatorOrUpper6

Channel layout bitmasks that use `GroupEquatorOrUpper6` fold-down coefficients are listed in Table 19.

Table 19: GroupEquatorOrUpper6 list_of_channel_masks[]

0x00004006	0x00005800	0x00005E00	0x00064000	0x00064600	0x00065800	0x00065E00
0x00004007	0x00005801	0x00005E01	0x00064001	0x00064601	0x00065801	0x00065E01
0x00004606	0x00005806	0x00005E06	0x00064006	0x00064606	0x00065806	0x00065E06
0x00004607	0x00005807	0x00005E07	0x00064007	0x00064607	0x00065807	0x00065E07

7.3.7.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper6

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 20.

Table 20: GroupEquatorOrUpper6 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	LeftLargestEqtr()	RightLargestEqtr()
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]	
(-135,0,1)	0 dB	
(135,0,1)		0 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 21.

Table 21: GroupEquatorOrUpper6 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	LeftLargestEqtr()	RightLargestEqtr()	Ch
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]		
(-135, 45,1)	-3,0103 dB		-3,0103 dB
(135, 45,1)		-3,0103 dB	-3,0103 dB

7.3.8 GroupEquatorOrUpper7

7.3.8.1 Channel Layouts in GroupEquatorOrUpper7

Channel layout bitmasks that use `GroupEquatorOrUpper7` fold-down coefficients are listed in Table 22.

Table 22: GroupEquatorOrUpper7 list_of_channel_masks[]

0x0000A006	0x0006A000	0x0006F800	0x30005800	0x3000F800	0x30064000	0x3006B800
0x0000A007	0x0006A001	0x0006F801	0x30005801	0x3000F801	0x30064001	0x3006B801
0x0000A606	0x0006A006	0x0006F806	0x30005806	0x3000F806	0x30064006	0x3006B806
0x0000A607	0x0006A007	0x0006F807	0x30005807	0x3000F807	0x30064007	0x3006B807
0x0000B800	0x0006A600	0x0006FE00	0x30005E00	0x3000FE00	0x30064600	0x3006BE00
0x0000B801	0x0006A601	0x0006FE01	0x30005E01	0x3000FE01	0x30064601	0x3006BE01
0x0000B806	0x0006A606	0x0006FE06	0x30005E06	0x3000FE06	0x30064606	0x3006BE06
0x0000B807	0x0006A607	0x0006FE07	0x30005E07	0x3000FE07	0x30064607	0x3006BE07
0x0000BE00	0x0006B800	0x30000006	0x3000A006	0x30060000	0x30065800	0x3006E000
0x0000BE01	0x0006B801	0x30000007	0x3000A007	0x30060001	0x30065801	0x3006E001
0x0000BE06	0x0006B806	0x30000606	0x3000A606	0x30060006	0x30065806	0x3006E006
0x0000BE07	0x0006B807	0x30000607	0x3000A607	0x30060007	0x30065807	0x3006E007
0x0000E006	0x0006BE00	0x30001800	0x3000B800	0x30060600	0x30065E00	0x3006E600
0x0000E007	0x0006BE01	0x30001801	0x3000B801	0x30060601	0x30065E01	0x3006E601
0x0000E606	0x0006BE06	0x30001806	0x3000B806	0x30060606	0x30065E06	0x3006E606
0x0000E607	0x0006BE07	0x30001807	0x3000B807	0x30060607	0x30065E07	0x3006E607
0x0000F800	0x0006E000	0x30001E00	0x3000BE00	0x30061800	0x3006A000	0x3006F800
0x0000F801	0x0006E001	0x30001E01	0x3000BE01	0x30061801	0x3006A001	0x3006F801
0x0000F806	0x0006E006	0x30001E06	0x3000BE06	0x30061806	0x3006A006	0x3006F806
0x0000F807	0x0006E007	0x30001E07	0x3000BE07	0x30061807	0x3006A007	0x3006F807
0x0000FE00	0x0006E600	0x30004006	0x3000E006	0x30061E00	0x3006A600	0x3006FE00
0x0000FE01	0x0006E601	0x30004007	0x3000E007	0x30061E01	0x3006A601	0x3006FE01
0x0000FE06	0x0006E606	0x30004606	0x3000E606	0x30061E06	0x3006A606	0x3006FE06
0x0000FE07	0x0006E607	0x30004607	0x3000E607	0x30061E07	0x3006A607	0x3006FE07

7.3.8.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper7

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 23.

Table 23: GroupEquatorOrUpper7 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	LeftLargestEqtr()	RightLargestEqtr()
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]	
(-135,0,1)	0 dB	
(135,0,1)		0 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 24.

Table 24: GroupEquatorOrUpper7 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	PrefOrder (Ltf, Lh)	PrefOrder (Rtf, Rh)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]	
(-135, 45,1)	0 dB	
(135, 45,1)		0 dB

7.3.9 GroupEquatorOrUpper8

7.3.9.1 Channel Layouts in GroupEquatorOrUpper8

Channel layout bitmasks that use `GroupEquatorOrUpper8` fold-down coefficients are listed in Table 25.

Table 25: GroupEquatorOrUpper8 list_of_channel_masks[]

list_of_channel_masks[]						
0x00304006	0x0030F800	0x0036A600	0x30300006	0x3030B800	0x30361800	0x3036A600
0x00304007	0x0030F801	0x0036A601	0x30300007	0x3030B801	0x30361801	0x3036A601
0x00304606	0x0030F806	0x0036A606	0x30300606	0x3030B806	0x30361806	0x3036A606
0x00304607	0x0030F807	0x0036A607	0x30300607	0x3030B807	0x30361807	0x3036A607
0x00305800	0x0030FE00	0x0036B800	0x30301800	0x3030BE00	0x30361E00	0x3036B800
0x00305801	0x0030FE01	0x0036B801	0x30301801	0x3030BE01	0x30361E01	0x3036B801
0x00305806	0x0030FE06	0x0036B806	0x30301806	0x3030BE06	0x30361E06	0x3036B806
0x00305807	0x0030FE07	0x0036B807	0x30301807	0x3030BE07	0x30361E07	0x3036B807
0x00305E00	0x00364000	0x0036BE00	0x30301E00	0x3030E006	0x30364000	0x3036BE00
0x00305E01	0x00364001	0x0036BE01	0x30301E01	0x3030E007	0x30364001	0x3036BE01
0x00305E06	0x00364006	0x0036BE06	0x30301E06	0x3030E606	0x30364006	0x3036BE06
0x00305E07	0x00364007	0x0036BE07	0x30301E07	0x3030E607	0x30364007	0x3036BE07
0x0030A006	0x00364600	0x0036E000	0x30304006	0x3030F800	0x30364600	0x3036E000
0x0030A007	0x00364601	0x0036E001	0x30304007	0x3030F801	0x30364601	0x3036E001
0x0030A606	0x00364606	0x0036E006	0x30304606	0x3030F806	0x30364606	0x3036E006
0x0030A607	0x00364607	0x0036E007	0x30304607	0x3030F807	0x30364607	0x3036E007
0x0030B800	0x00365800	0x0036E600	0x30305800	0x3030FE00	0x30365800	0x3036E600
0x0030B801	0x00365801	0x0036E601	0x30305801	0x3030FE01	0x30365801	0x3036E601
0x0030B806	0x00365806	0x0036E606	0x30305806	0x3030FE06	0x30365806	0x3036E606
0x0030B807	0x00365807	0x0036E607	0x30305807	0x3030FE07	0x30365807	0x3036E607
0x0030BE00	0x00365E00	0x0036F800	0x30305E00	0x30360000	0x30365E00	0x3036F800
0x0030BE01	0x00365E01	0x0036F801	0x30305E01	0x30360001	0x30365E01	0x3036F801
0x0030BE06	0x00365E06	0x0036F806	0x30305E06	0x30360006	0x30365E06	0x3036F806
0x0030BE07	0x00365E07	0x0036F807	0x30305E07	0x30360007	0x30365E07	0x3036F807
0x0030E006	0x0036A000	0x0036FE00	0x3030A006	0x30360600	0x3036A000	0x3036FE00
0x0030E007	0x0036A001	0x0036FE01	0x3030A007	0x30360601	0x3036A001	0x3036FE01
0x0030E606	0x0036A006	0x0036FE06	0x3030A606	0x30360606	0x3036A006	0x3036FE06
0x0030E607	0x0036A007	0x0036FE07	0x3030A607	0x30360607	0x3036A007	0x3036FE07

7.3.9.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper8

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 26.

Table 26: GroupEquatorOrUpper8 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	LeftLargestEqtr ()	RightLargestEqtr ()
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]	
(-135, 0, 1)	0 dB	
(135, 0, 1)		0 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 27.

Table 27: GroupEquatorOrUpper8 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	ExclusiveOr (Lhs, Ltm)	ExclusiveOr (Rhs, Rtm)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]	
(-135, 45, 1)	0 dB	
(135, 45, 1)		0 dB

7.3.10 GroupEquatorOrUpper9

7.3.10.1 Channel Layouts in GroupEquatorOrUpper9

Channel layout bitmasks that use `GroupEquatorOrUpper9` fold-down coefficients are listed in Table 28.

Table 28: GroupEquatorOrUpper9 list_of_channel_masks[]

0x00700006	0x01B00006	0x01F00006	0xC0300006	0xC0700006	0xC1B00006	0xC1F00006
0x00700007	0x01B00007	0x01F00007	0xC0300007	0xC0700007	0xC1B00007	0xC1F00007
0x00700606	0x01B00606	0x01F00606	0xC0300606	0xC0700606	0xC1B00606	0xC1F00606
0x00700607	0x01B00607	0x01F00607	0xC0300607	0xC0700607	0xC1B00607	0xC1F00607
0x00701800	0x01B01800	0x01F01800	0xC0301800	0xC0701800	0xC1B01800	0xC1F01800
0x00701801	0x01B01801	0x01F01801	0xC0301801	0xC0701801	0xC1B01801	0xC1F01801
0x00701806	0x01B01806	0x01F01806	0xC0301806	0xC0701806	0xC1B01806	0xC1F01806
0x00701807	0x01B01807	0x01F01807	0xC0301807	0xC0701807	0xC1B01807	0xC1F01807
0x00701E00	0x01B01E00	0x01F01E00	0xC0301E00	0xC0701E00	0xC1B01E00	0xC1F01E00
0x00701E01	0x01B01E01	0x01F01E01	0xC0301E01	0xC0701E01	0xC1B01E01	0xC1F01E01
0x00701E06	0x01B01E06	0x01F01E06	0xC0301E06	0xC0701E06	0xC1B01E06	0xC1F01E06
0x00701E07	0x01B01E07	0x01F01E07	0xC0301E07	0xC0701E07	0xC1B01E07	0xC1F01E07
0x00760000	0x01B60000	0x01F60000	0xC0360000	0xC0760000	0xC1B60000	0xC1F60000
0x00760001	0x01B60001	0x01F60001	0xC0360001	0xC0760001	0xC1B60001	0xC1F60001
0x00760006	0x01B60006	0x01F60006	0xC0360006	0xC0760006	0xC1B60006	0xC1F60006
0x00760007	0x01B60007	0x01F60007	0xC0360007	0xC0760007	0xC1B60007	0xC1F60007
0x00760600	0x01B60600	0x01F60600	0xC0360600	0xC0760600	0xC1B60600	0xC1F60600
0x00760601	0x01B60601	0x01F60601	0xC0360601	0xC0760601	0xC1B60601	0xC1F60601
0x00760606	0x01B60606	0x01F60606	0xC0360606	0xC0760606	0xC1B60606	0xC1F60606
0x00760607	0x01B60607	0x01F60607	0xC0360607	0xC0760607	0xC1B60607	0xC1F60607
0x00761800	0x01B61800	0x01F61800	0xC0361800	0xC0761800	0xC1B61800	0xC1F61800
0x00761801	0x01B61801	0x01F61801	0xC0361801	0xC0761801	0xC1B61801	0xC1F61801
0x00761806	0x01B61806	0x01F61806	0xC0361806	0xC0761806	0xC1B61806	0xC1F61806
0x00761807	0x01B61807	0x01F61807	0xC0361807	0xC0761807	0xC1B61807	0xC1F61807
0x00761E00	0x01B61E00	0x01F61E00	0xC0361E00	0xC0761E00	0xC1B61E00	0xC1F61E00
0x00761E01	0x01B61E01	0x01F61E01	0xC0361E01	0xC0761E01	0xC1B61E01	0xC1F61E01
0x00761E06	0x01B61E06	0x01F61E06	0xC0361E06	0xC0761E06	0xC1B61E06	0xC1F61E06
0x00761E07	0x01B61E07	0x01F61E07	0xC0361E07	0xC0761E07	0xC1B61E07	0xC1F61E07

7.3.10.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper9

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 29.

Table 29: GroupEquatorOrUpper9 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	PrefOrder (Lhr, Split (Lss,Chr,-3 dB), ClosestElv (0, ClosestAzUH(-90)))	PrefOrder (Rhr, Split (Rss,Chr,-3 dB), ClosestElv (0, ClosestAzUH(90)))
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]	
(-135,0,1)	0 dB	
(135,0,1)		0 dB

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 30.

Table 30: GroupEquatorOrUpper9 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	ClosestAzEq(-40)	ClosestAzEq(40)	ExclusiveOr (Lhs,Ltm)	ExclusiveOr (Rhs,Rtm)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]			
(-45, 45,1)	-3,0103 dB		-3,0103 dB	
(45, 45,1)		-3,0103 dB		-3,0103 dB

7.3.11 GroupEquatorOrUpper10

7.3.11.1 Channel Layouts in GroupEquatorOrUpper10

Channel layout bitmasks that use `GroupEquatorOrUpper10` fold-down coefficients are listed in Table 31.

Table 31: GroupEquatorOrUpper10 list_of_channel_masks[]

0x0040A006	0x01C6FE00	0x31865E00	0xC000B800	0xC186E000	0xF0301E00	0xF1861E00
0x0040A007	0x01C6FE01	0x31865E01	0xC000B801	0xC186E001	0xF0301E01	0xF1861E01
0x0040A606	0x01C6FE06	0x31865E06	0xC000B806	0xC186E006	0xF0301E06	0xF1861E06
0x0040A607	0x01C6FE07	0x31865E07	0xC000B807	0xC186E007	0xF0301E07	0xF1861E07
0x0040B800	0x01F04006	0x3186A000	0xC000BE00	0xC186E600	0xF0304006	0xF1864000
0x0040B801	0x01F04007	0x3186A001	0xC000BE01	0xC186E601	0xF0304007	0xF1864001
0x0040B806	0x01F04606	0x3186A006	0xC000BE06	0xC186E606	0xF0304606	0xF1864006
0x0040B807	0x01F04607	0x3186A007	0xC000BE07	0xC186E607	0xF0304607	0xF1864007
0x0040BE00	0x01F05800	0x3186A600	0xC000E006	0xC186F800	0xF0305800	0xF1864600
0x0040BE01	0x01F05801	0x3186A601	0xC000E007	0xC186F801	0xF0305801	0xF1864601
0x0040BE06	0x01F05806	0x3186A606	0xC000E606	0xC186F806	0xF0305806	0xF1864606
0x0040BE07	0x01F05807	0x3186A607	0xC000E607	0xC186F807	0xF0305807	0xF1864607
0x0040E006	0x01F05E00	0x3186B800	0xC000F800	0xC186FE00	0xF0305E00	0xF1865800
0x0040E007	0x01F05E01	0x3186B801	0xC000F801	0xC186FE01	0xF0305E01	0xF1865801
0x0040E606	0x01F05E06	0x3186B806	0xC000F806	0xC186FE06	0xF0305E06	0xF1865806
0x0040E607	0x01F05E07	0x3186B807	0xC000F807	0xC186FE07	0xF0305E07	0xF1865807
0x0040F800	0x01F0A006	0x3186BE00	0xC000FE00	0xC1B04006	0xF030A006	0xF1865E00
0x0040F801	0x01F0A007	0x3186BE01	0xC000FE01	0xC1B04007	0xF030A007	0xF1865E01
0x0040F806	0x01F0A606	0x3186BE06	0xC000FE06	0xC1B04606	0xF030A606	0xF1865E06
0x0040F807	0x01F0A607	0x3186BE07	0xC000FE07	0xC1B04607	0xF030A607	0xF1865E07
0x0040FE00	0x01F0B800	0x3186E000	0xC0064000	0xC1B05800	0xF030B800	0xF186A000
0x0040FE01	0x01F0B801	0x3186E001	0xC0064001	0xC1B05801	0xF030B801	0xF186A001
0x0040FE06	0x01F0B806	0x3186E006	0xC0064006	0xC1B05806	0xF030B806	0xF186A006
0x0040FE07	0x01F0B807	0x3186E007	0xC0064007	0xC1B05807	0xF030B807	0xF186A007
0x0046A000	0x01F0BE00	0x3186E600	0xC0064600	0xC1B05E00	0xF030BE00	0xF186A600
0x0046A001	0x01F0BE01	0x3186E601	0xC0064601	0xC1B05E01	0xF030BE01	0xF186A601
0x0046A006	0x01F0BE06	0x3186E606	0xC0064606	0xC1B05E06	0xF030BE06	0xF186A606
0x0046A007	0x01F0BE07	0x3186E607	0xC0064607	0xC1B05E07	0xF030BE07	0xF186A607
0x0046A600	0x01F0E006	0x3186F800	0xC0065800	0xC1B0A006	0xF030E006	0xF186B800
0x0046A601	0x01F0E007	0x3186F801	0xC0065801	0xC1B0A007	0xF030E007	0xF186B801

0x0046A606	0x01F0E606	0x3186F806	0xC0065806	0xC1B0A606	0xF030E606	0xF186B806
0x0046A607	0x01F0E607	0x3186F807	0xC0065807	0xC1B0A607	0xF030E607	0xF186B807
0x0046B800	0x01F0F800	0x3186FE00	0xC0065E00	0xC1B0B800	0xF030F800	0xF186BE00
0x0046B801	0x01F0F801	0x3186FE01	0xC0065E01	0xC1B0B801	0xF030F801	0xF186BE01
0x0046B806	0x01F0F806	0x3186FE06	0xC0065E06	0xC1B0B806	0xF030F806	0xF186BE06
0x0046B807	0x01F0F807	0x3186FE07	0xC0065E07	0xC1B0B807	0xF030F807	0xF186BE07
0x0046BE00	0x01F0FE00	0x31B00006	0xC006A000	0xC1B0BE00	0xF030FE00	0xF186E000
0x0046BE01	0x01F0FE01	0x31B00007	0xC006A001	0xC1B0BE01	0xF030FE01	0xF186E001
0x0046BE06	0x01F0FE06	0x31B00606	0xC006A006	0xC1B0BE06	0xF030FE06	0xF186E006
0x0046BE07	0x01F0FE07	0x31B00607	0xC006A007	0xC1B0BE07	0xF030FE07	0xF186E007
0x0046E000	0x01F64000	0x31B01800	0xC006A600	0xC1B0E006	0xF0360000	0xF186E600
0x0046E001	0x01F64001	0x31B01801	0xC006A601	0xC1B0E007	0xF0360001	0xF186E601
0x0046E006	0x01F64006	0x31B01806	0xC006A606	0xC1B0E606	0xF0360006	0xF186E606
0x0046E007	0x01F64007	0x31B01807	0xC006A607	0xC1B0E607	0xF0360007	0xF186E607
0x0046E600	0x01F64600	0x31B01E00	0xC006B800	0xC1B0F800	0xF0360600	0xF186F800
0x0046E601	0x01F64601	0x31B01E01	0xC006B801	0xC1B0F801	0xF0360601	0xF186F801
0x0046E606	0x01F64606	0x31B01E06	0xC006B806	0xC1B0F806	0xF0360606	0xF186F806
0x0046E607	0x01F64607	0x31B01E07	0xC006B807	0xC1B0F807	0xF0360607	0xF186F807
0x0046F800	0x01F65800	0x31B04006	0xC006BE00	0xC1B0FE00	0xF0361800	0xF186FE00
0x0046F801	0x01F65801	0x31B04007	0xC006BE01	0xC1B0FE01	0xF0361801	0xF186FE01
0x0046F806	0x01F65806	0x31B04606	0xC006BE06	0xC1B0FE06	0xF0361806	0xF186FE06
0x0046F807	0x01F65807	0x31B04607	0xC006BE07	0xC1B0FE07	0xF0361807	0xF186FE07
0x0046FE00	0x01F65E00	0x31B05800	0xC006E000	0xC1B64000	0xF0361E00	0xF1B00006
0x0046FE01	0x01F65E01	0x31B05801	0xC006E001	0xC1B64001	0xF0361E01	0xF1B00007
0x0046FE06	0x01F65E06	0x31B05806	0xC006E006	0xC1B64006	0xF0361E06	0xF1B00606
0x0046FE07	0x01F65E07	0x31B05807	0xC006E007	0xC1B64007	0xF0361E07	0xF1B00607
0x00704006	0x01F6A000	0x31B05E00	0xC006E600	0xC1B64600	0xF0364000	0xF1B01800
0x00704007	0x01F6A001	0x31B05E01	0xC006E601	0xC1B64601	0xF0364001	0xF1B01801
0x00704606	0x01F6A006	0x31B05E06	0xC006E606	0xC1B64606	0xF0364006	0xF1B01806
0x00704607	0x01F6A007	0x31B05E07	0xC006E607	0xC1B64607	0xF0364007	0xF1B01807
0x00705800	0x01F6A600	0x31B0A006	0xC006F800	0xC1B65800	0xF0364600	0xF1B01E00
0x00705801	0x01F6A601	0x31B0A007	0xC006F801	0xC1B65801	0xF0364601	0xF1B01E01
0x00705806	0x01F6A606	0x31B0A606	0xC006F806	0xC1B65806	0xF0364606	0xF1B01E06
0x00705807	0x01F6A607	0x31B0A607	0xC006F807	0xC1B65807	0xF0364607	0xF1B01E07
0x00705E00	0x01F6B800	0x31B0B800	0xC006FE00	0xC1B65E00	0xF0365800	0xF1B04006
0x00705E01	0x01F6B801	0x31B0B801	0xC006FE01	0xC1B65E01	0xF0365801	0xF1B04007
0x00705E06	0x01F6B806	0x31B0B806	0xC006FE06	0xC1B65E06	0xF0365806	0xF1B04606
0x00705E07	0x01F6B807	0x31B0B807	0xC006FE07	0xC1B65E07	0xF0365807	0xF1B04607
0x0070A006	0x01F6BE00	0x31B0BE00	0xC0304006	0xC1B6A000	0xF0365E00	0xF1B05800
0x0070A007	0x01F6BE01	0x31B0BE01	0xC0304007	0xC1B6A001	0xF0365E01	0xF1B05801
0x0070A606	0x01F6BE06	0x31B0BE06	0xC0304606	0xC1B6A006	0xF0365E06	0xF1B05806
0x0070A607	0x01F6BE07	0x31B0BE07	0xC0304607	0xC1B6A007	0xF0365E07	0xF1B05807
0x0070B800	0x01F6E000	0x31B0E006	0xC0305800	0xC1B6A600	0xF036A000	0xF1B05E00
0x0070B801	0x01F6E001	0x31B0E007	0xC0305801	0xC1B6A601	0xF036A001	0xF1B05E01
0x0070B806	0x01F6E006	0x31B0E606	0xC0305806	0xC1B6A606	0xF036A006	0xF1B05E06
0x0070B807	0x01F6E007	0x31B0E607	0xC0305807	0xC1B6A607	0xF036A007	0xF1B05E07
0x0070BE00	0x01F6E600	0x31B0F800	0xC0305E00	0xC1B6B800	0xF036A600	0xF1B0A006
0x0070BE01	0x01F6E601	0x31B0F801	0xC0305E01	0xC1B6B801	0xF036A601	0xF1B0A007
0x0070BE06	0x01F6E606	0x31B0F806	0xC0305E06	0xC1B6B806	0xF036A606	0xF1B0A606
0x0070BE07	0x01F6E607	0x31B0F807	0xC0305E07	0xC1B6B807	0xF036A607	0xF1B0A607
0x0070E006	0x01F6F800	0x31B0FE00	0xC030A006	0xC1B6BE00	0xF036B800	0xF1B0B800
0x0070E007	0x01F6F801	0x31B0FE01	0xC030A007	0xC1B6BE01	0xF036B801	0xF1B0B801
0x0070E606	0x01F6F806	0x31B0FE06	0xC030A606	0xC1B6BE06	0xF036B806	0xF1B0B806
0x0070E607	0x01F6F807	0x31B0FE07	0xC030A607	0xC1B6BE07	0xF036B807	0xF1B0B807
0x0070F800	0x01F6FE00	0x31B60000	0xC030B800	0xC1B6E000	0xF036BE00	0xF1B0BE00
0x0070F801	0x01F6FE01	0x31B60001	0xC030B801	0xC1B6E001	0xF036BE01	0xF1B0BE01
0x0070F806	0x01F6FE06	0x31B60006	0xC030B806	0xC1B6E006	0xF036BE06	0xF1B0BE06
0x0070F807	0x01F6FE07	0x31B60007	0xC030B807	0xC1B6E007	0xF036BE07	0xF1B0BE07
0x0070FE00	0x30400006	0x31B60600	0xC030BE00	0xC1B6E600	0xF036E000	0xF1B0E006
0x0070FE01	0x30400007	0x31B60601	0xC030BE01	0xC1B6E601	0xF036E001	0xF1B0E007
0x0070FE06	0x30400606	0x31B60606	0xC030BE06	0xC1B6E606	0xF036E006	0xF1B0E606
0x0070FE07	0x30400607	0x31B60607	0xC030BE07	0xC1B6E607	0xF036E007	0xF1B0E607
0x00764000	0x30401800	0x31B61800	0xC030E006	0xC1B6F800	0xF036E600	0xF1B0F800
0x00764001	0x30401801	0x31B61801	0xC030E007	0xC1B6F801	0xF036E601	0xF1B0F801
0x00764006	0x30401806	0x31B61806	0xC030E606	0xC1B6F806	0xF036E606	0xF1B0F806

0x00764007	0x30401807	0x31B61807	0xC030E607	0xC1B6F807	0xF036E607	0xF1B0F807
0x00764600	0x30401E00	0x31B61E00	0xC030F800	0xC1B6FE00	0xF036F800	0xF1B0FE00
0x00764601	0x30401E01	0x31B61E01	0xC030F801	0xC1B6FE01	0xF036F801	0xF1B0FE01
0x00764606	0x30401E06	0x31B61E06	0xC030F806	0xC1B6FE06	0xF036F806	0xF1B0FE06
0x00764607	0x30401E07	0x31B61E07	0xC030F807	0xC1B6FE07	0xF036F807	0xF1B0FE07
0x00765800	0x30404006	0x31B64000	0xC030FE00	0xC1C04006	0xF036FE00	0xF1B60000
0x00765801	0x30404007	0x31B64001	0xC030FE01	0xC1C04007	0xF036FE01	0xF1B60001
0x00765806	0x30404606	0x31B64006	0xC030FE06	0xC1C04606	0xF036FE06	0xF1B60006
0x00765807	0x30404607	0x31B64007	0xC030FE07	0xC1C04607	0xF036FE07	0xF1B60007
0x00765E00	0x30405800	0x31B64600	0xC0364000	0xC1C05800	0xF0400006	0xF1B60600
0x00765E01	0x30405801	0x31B64601	0xC0364001	0xC1C05801	0xF0400007	0xF1B60601
0x00765E06	0x30405806	0x31B64606	0xC0364006	0xC1C05806	0xF0400606	0xF1B60606
0x00765E07	0x30405807	0x31B64607	0xC0364007	0xC1C05807	0xF0400607	0xF1B60607
0x0076A000	0x30405E00	0x31B65800	0xC0364600	0xC1C05E00	0xF0401800	0xF1B61800
0x0076A001	0x30405E01	0x31B65801	0xC0364601	0xC1C05E01	0xF0401801	0xF1B61801
0x0076A006	0x30405E06	0x31B65806	0xC0364606	0xC1C05E06	0xF0401806	0xF1B61806
0x0076A007	0x30405E07	0x31B65807	0xC0364607	0xC1C05E07	0xF0401807	0xF1B61807
0x0076A600	0x3040A006	0x31B65E00	0xC0365800	0xC1C0A006	0xF0401E00	0xF1B61E00
0x0076A601	0x3040A007	0x31B65E01	0xC0365801	0xC1C0A007	0xF0401E01	0xF1B61E01
0x0076A606	0x3040A606	0x31B65E06	0xC0365806	0xC1C0A606	0xF0401E06	0xF1B61E06
0x0076A607	0x3040A607	0x31B65E07	0xC0365807	0xC1C0A607	0xF0401E07	0xF1B61E07
0x0076B800	0x3040B800	0x31B6A000	0xC0365E00	0xC1C0B800	0xF0404006	0xF1B64000
0x0076B801	0x3040B801	0x31B6A001	0xC0365E01	0xC1C0B801	0xF0404007	0xF1B64001
0x0076B806	0x3040B806	0x31B6A006	0xC0365E06	0xC1C0B806	0xF0404606	0xF1B64006
0x0076B807	0x3040B807	0x31B6A007	0xC0365E07	0xC1C0B807	0xF0404607	0xF1B64007
0x0076BE00	0x3040BE00	0x31B6A600	0xC036A000	0xC1C0BE00	0xF0405800	0xF1B64600
0x0076BE01	0x3040BE01	0x31B6A601	0xC036A001	0xC1C0BE01	0xF0405801	0xF1B64601
0x0076BE06	0x3040BE06	0x31B6A606	0xC036A006	0xC1C0BE06	0xF0405806	0xF1B64606
0x0076BE07	0x3040BE07	0x31B6A607	0xC036A007	0xC1C0BE07	0xF0405807	0xF1B64607
0x0076E000	0x3040E006	0x31B6B800	0xC036A600	0xC1C0E006	0xF0405E00	0xF1B65800
0x0076E001	0x3040E007	0x31B6B801	0xC036A601	0xC1C0E007	0xF0405E01	0xF1B65801
0x0076E006	0x3040E606	0x31B6B806	0xC036A606	0xC1C0E606	0xF0405E06	0xF1B65806
0x0076E007	0x3040E607	0x31B6B807	0xC036A607	0xC1C0E607	0xF0405E07	0xF1B65807
0x0076E600	0x3040F800	0x31B6BE00	0xC036B800	0xC1C0F800	0xF040A006	0xF1B65E00
0x0076E601	0x3040F801	0x31B6BE01	0xC036B801	0xC1C0F801	0xF040A007	0xF1B65E01
0x0076E606	0x3040F806	0x31B6BE06	0xC036B806	0xC1C0F806	0xF040A606	0xF1B65E06
0x0076E607	0x3040F807	0x31B6BE07	0xC036B807	0xC1C0F807	0xF040A607	0xF1B65E07
0x0076F800	0x3040FE00	0x31B6E000	0xC036BE00	0xC1C0FE00	0xF040B800	0xF1B6A000
0x0076F801	0x3040FE01	0x31B6E001	0xC036BE01	0xC1C0FE01	0xF040B801	0xF1B6A001
0x0076F806	0x3040FE06	0x31B6E006	0xC036BE06	0xC1C0FE06	0xF040B806	0xF1B6A006
0x0076F807	0x3040FE07	0x31B6E007	0xC036BE07	0xC1C0FE07	0xF040B807	0xF1B6A007
0x0076FE00	0x30460000	0x31B6E600	0xC036E000	0xC1C64000	0xF040BE00	0xF1B6A600
0x0076FE01	0x30460001	0x31B6E601	0xC036E001	0xC1C64001	0xF040BE01	0xF1B6A601
0x0076FE06	0x30460006	0x31B6E606	0xC036E006	0xC1C64006	0xF040BE06	0xF1B6A606
0x0076FE07	0x30460007	0x31B6E607	0xC036E007	0xC1C64007	0xF040BE07	0xF1B6A607
0x01804006	0x30460600	0x31B6F800	0xC036E600	0xC1C64600	0xF040E006	0xF1B6B800
0x01804007	0x30460601	0x31B6F801	0xC036E601	0xC1C64601	0xF040E007	0xF1B6B801
0x01804606	0x30460606	0x31B6F806	0xC036E606	0xC1C64606	0xF040E606	0xF1B6B806
0x01804607	0x30460607	0x31B6F807	0xC036E607	0xC1C64607	0xF040E607	0xF1B6B807
0x01805800	0x30461800	0x31B6FE00	0xC036F800	0xC1C65800	0xF040F800	0xF1B6BE00
0x01805801	0x30461801	0x31B6FE01	0xC036F801	0xC1C65801	0xF040F801	0xF1B6BE01
0x01805806	0x30461806	0x31B6FE06	0xC036F806	0xC1C65806	0xF040F806	0xF1B6BE06
0x01805807	0x30461807	0x31B6FE07	0xC036F807	0xC1C65807	0xF040F807	0xF1B6BE07
0x01805E00	0x30461E00	0x31C00006	0xC036FE00	0xC1C65E00	0xF040FE00	0xF1B6E000
0x01805E01	0x30461E01	0x31C00007	0xC036FE01	0xC1C65E01	0xF040FE01	0xF1B6E001
0x01805E06	0x30461E06	0x31C00606	0xC036FE06	0xC1C65E06	0xF040FE06	0xF1B6E006
0x01805E07	0x30461E07	0x31C00607	0xC036FE07	0xC1C65E07	0xF040FE07	0xF1B6E007
0x0180A006	0x30464000	0x31C01800	0xC0404006	0xC1C6A000	0xF0460000	0xF1B6E600
0x0180A007	0x30464001	0x31C01801	0xC0404007	0xC1C6A001	0xF0460001	0xF1B6E601
0x0180A606	0x30464006	0x31C01806	0xC0404606	0xC1C6A006	0xF0460006	0xF1B6E606
0x0180A607	0x30464007	0x31C01807	0xC0404607	0xC1C6A007	0xF0460007	0xF1B6E607
0x0180B800	0x30464600	0x31C01E00	0xC0405800	0xC1C6A600	0xF0460600	0xF1B6F800
0x0180B801	0x30464601	0x31C01E01	0xC0405801	0xC1C6A601	0xF0460601	0xF1B6F801
0x0180B806	0x30464606	0x31C01E06	0xC0405806	0xC1C6A606	0xF0460606	0xF1B6F806
0x0180B807	0x30464607	0x31C01E07	0xC0405807	0xC1C6A607	0xF0460607	0xF1B6F807

0x0180BE00	0x30465800	0x31C04006	0xC0405E00	0xC1C6B800	0xF0461800	0xF1B6FE00
0x0180BE01	0x30465801	0x31C04007	0xC0405E01	0xC1C6B801	0xF0461801	0xF1B6FE01
0x0180BE06	0x30465806	0x31C04606	0xC0405E06	0xC1C6B806	0xF0461806	0xF1B6FE06
0x0180BE07	0x30465807	0x31C04607	0xC0405E07	0xC1C6B807	0xF0461807	0xF1B6FE07
0x0180E006	0x30465E00	0x31C05800	0xC040A006	0xC1C6BE00	0xF0461E00	0xF1C00006
0x0180E007	0x30465E01	0x31C05801	0xC040A007	0xC1C6BE01	0xF0461E01	0xF1C00007
0x0180E606	0x30465E06	0x31C05806	0xC040A606	0xC1C6BE06	0xF0461E06	0xF1C00606
0x0180E607	0x30465E07	0x31C05807	0xC040A607	0xC1C6BE07	0xF0461E07	0xF1C00607
0x0180F800	0x3046A000	0x31C05E00	0xC040B800	0xC1C6E000	0xF0464000	0xF1C01800
0x0180F801	0x3046A001	0x31C05E01	0xC040B801	0xC1C6E001	0xF0464001	0xF1C01801
0x0180F806	0x3046A006	0x31C05E06	0xC040B806	0xC1C6E006	0xF0464006	0xF1C01806
0x0180F807	0x3046A007	0x31C05E07	0xC040B807	0xC1C6E007	0xF0464007	0xF1C01807
0x0180FE00	0x3046A600	0x31C0A006	0xC040BE00	0xC1C6E600	0xF0464600	0xF1C01E00
0x0180FE01	0x3046A601	0x31C0A007	0xC040BE01	0xC1C6E601	0xF0464601	0xF1C01E01
0x0180FE06	0x3046A606	0x31C0A606	0xC040BE06	0xC1C6E606	0xF0464606	0xF1C01E06
0x0180FE07	0x3046A607	0x31C0A607	0xC040BE07	0xC1C6E607	0xF0464607	0xF1C01E07
0x01864000	0x3046B800	0x31C0B800	0xC040E006	0xC1C6F800	0xF0465800	0xF1C04006
0x01864001	0x3046B801	0x31C0B801	0xC040E007	0xC1C6F801	0xF0465801	0xF1C04007
0x01864006	0x3046B806	0x31C0B806	0xC040E606	0xC1C6F806	0xF0465806	0xF1C04606
0x01864007	0x3046B807	0x31C0B807	0xC040E607	0xC1C6F807	0xF0465807	0xF1C04607
0x01864600	0x3046BE00	0x31C0BE00	0xC040F800	0xC1C6FE00	0xF0465E00	0xF1C05800
0x01864601	0x3046BE01	0x31C0BE01	0xC040F801	0xC1C6FE01	0xF0465E01	0xF1C05801
0x01864606	0x3046BE06	0x31C0BE06	0xC040F806	0xC1C6FE06	0xF0465E06	0xF1C05806
0x01864607	0x3046BE07	0x31C0BE07	0xC040F807	0xC1C6FE07	0xF0465E07	0xF1C05807
0x01865800	0x3046E000	0x31C0E006	0xC040FE00	0xC1F04006	0xF046A000	0xF1C05E00
0x01865801	0x3046E001	0x31C0E007	0xC040FE01	0xC1F04007	0xF046A001	0xF1C05E01
0x01865806	0x3046E006	0x31C0E606	0xC040FE06	0xC1F04606	0xF046A006	0xF1C05E06
0x01865807	0x3046E007	0x31C0E607	0xC040FE07	0xC1F04607	0xF046A007	0xF1C05E07
0x01865E00	0x3046E600	0x31C0F800	0xC0464000	0xC1F05800	0xF046A600	0xF1C0A006
0x01865E01	0x3046E601	0x31C0F801	0xC0464001	0xC1F05801	0xF046A601	0xF1C0A007
0x01865E06	0x3046E606	0x31C0F806	0xC0464006	0xC1F05806	0xF046A606	0xF1C0A606
0x01865E07	0x3046E607	0x31C0F807	0xC0464007	0xC1F05807	0xF046A607	0xF1C0A607
0x0186A000	0x3046F800	0x31C0FE00	0xC0464600	0xC1F05E00	0xF046B800	0xF1C0B800
0x0186A001	0x3046F801	0x31C0FE01	0xC0464601	0xC1F05E01	0xF046B801	0xF1C0B801
0x0186A006	0x3046F806	0x31C0FE06	0xC0464606	0xC1F05E06	0xF046B806	0xF1C0B806
0x0186A007	0x3046F807	0x31C0FE07	0xC0464607	0xC1F05E07	0xF046B807	0xF1C0B807
0x0186A600	0x3046FE00	0x31C60000	0xC0465800	0xC1F0A006	0xF046BE00	0xF1C0BE00
0x0186A601	0x3046FE01	0x31C60001	0xC0465801	0xC1F0A007	0xF046BE01	0xF1C0BE01
0x0186A606	0x3046FE06	0x31C60006	0xC0465806	0xC1F0A606	0xF046BE06	0xF1C0BE06
0x0186A607	0x3046FE07	0x31C60007	0xC0465807	0xC1F0A607	0xF046BE07	0xF1C0BE07
0x0186B800	0x30700006	0x31C60600	0xC0465E00	0xC1F0B800	0xF046E000	0xF1C0E006
0x0186B801	0x30700007	0x31C60601	0xC0465E01	0xC1F0B801	0xF046E001	0xF1C0E007
0x0186B806	0x30700606	0x31C60606	0xC0465E06	0xC1F0B806	0xF046E006	0xF1C0E606
0x0186B807	0x30700607	0x31C60607	0xC0465E07	0xC1F0B807	0xF046E007	0xF1C0E607
0x0186BE00	0x30701800	0x31C61800	0xC046A000	0xC1F0BE00	0xF046E600	0xF1C0F800
0x0186BE01	0x30701801	0x31C61801	0xC046A001	0xC1F0BE01	0xF046E601	0xF1C0F801
0x0186BE06	0x30701806	0x31C61806	0xC046A006	0xC1F0BE06	0xF046E606	0xF1C0F806
0x0186BE07	0x30701807	0x31C61807	0xC046A007	0xC1F0BE07	0xF046E607	0xF1C0F807
0x0186E000	0x30701E00	0x31C61E00	0xC046A600	0xC1F0E006	0xF046F800	0xF1C0FE00
0x0186E001	0x30701E01	0x31C61E01	0xC046A601	0xC1F0E007	0xF046F801	0xF1C0FE01
0x0186E006	0x30701E06	0x31C61E06	0xC046A606	0xC1F0E606	0xF046F806	0xF1C0FE06
0x0186E007	0x30701E07	0x31C61E07	0xC046A607	0xC1F0E607	0xF046F807	0xF1C0FE07
0x0186E600	0x30704006	0x31C64000	0xC046B800	0xC1F0F800	0xF046FE00	0xF1C60000
0x0186E601	0x30704007	0x31C64001	0xC046B801	0xC1F0F801	0xF046FE01	0xF1C60001
0x0186E606	0x30704606	0x31C64006	0xC046B806	0xC1F0F806	0xF046FE06	0xF1C60006
0x0186E607	0x30704607	0x31C64007	0xC046B807	0xC1F0F807	0xF046FE07	0xF1C60007
0x0186F800	0x30705800	0x31C64600	0xC046BE00	0xC1F0FE00	0xF0700006	0xF1C60600
0x0186F801	0x30705801	0x31C64601	0xC046BE01	0xC1F0FE01	0xF0700007	0xF1C60601
0x0186F806	0x30705806	0x31C64606	0xC046BE06	0xC1F0FE06	0xF0700606	0xF1C60606
0x0186F807	0x30705807	0x31C64607	0xC046BE07	0xC1F0FE07	0xF0700607	0xF1C60607
0x0186FE00	0x30705E00	0x31C65800	0xC046E000	0xC1F64000	0xF0701800	0xF1C61800
0x0186FE01	0x30705E01	0x31C65801	0xC046E001	0xC1F64001	0xF0701801	0xF1C61801
0x0186FE06	0x30705E06	0x31C65806	0xC046E006	0xC1F64006	0xF0701806	0xF1C61806
0x0186FE07	0x30705E07	0x31C65807	0xC046E007	0xC1F64007	0xF0701807	0xF1C61807
0x01B04006	0x3070A006	0x31C65E00	0xC046E600	0xC1F64600	0xF0701E00	0xF1C61E00

0x01B04007	0x3070A007	0x31C65E01	0xC046E601	0xC1F64601	0xF0701E01	0xF1C61E01
0x01B04606	0x3070A606	0x31C65E06	0xC046E606	0xC1F64606	0xF0701E06	0xF1C61E06
0x01B04607	0x3070A607	0x31C65E07	0xC046E607	0xC1F64607	0xF0701E07	0xF1C61E07
0x01B05800	0x3070B800	0x31C6A000	0xC046F800	0xC1F65800	0xF0704006	0xF1C64000
0x01B05801	0x3070B801	0x31C6A001	0xC046F801	0xC1F65801	0xF0704007	0xF1C64001
0x01B05806	0x3070B806	0x31C6A006	0xC046F806	0xC1F65806	0xF0704606	0xF1C64006
0x01B05807	0x3070B807	0x31C6A007	0xC046F807	0xC1F65807	0xF0704607	0xF1C64007
0x01B05E00	0x3070BE00	0x31C6A600	0xC046FE00	0xC1F65E00	0xF0705800	0xF1C64600
0x01B05E01	0x3070BE01	0x31C6A601	0xC046FE01	0xC1F65E01	0xF0705801	0xF1C64601
0x01B05E06	0x3070BE06	0x31C6A606	0xC046FE06	0xC1F65E06	0xF0705806	0xF1C64606
0x01B05E07	0x3070BE07	0x31C6A607	0xC046FE07	0xC1F65E07	0xF0705807	0xF1C64607
0x01B0A006	0x3070E006	0x31C6B800	0xC0704006	0xC1F6A000	0xF0705E00	0xF1C65800
0x01B0A007	0x3070E007	0x31C6B801	0xC0704007	0xC1F6A001	0xF0705E01	0xF1C65801
0x01B0A606	0x3070E606	0x31C6B806	0xC0704606	0xC1F6A006	0xF0705E06	0xF1C65806
0x01B0A607	0x3070E607	0x31C6B807	0xC0704607	0xC1F6A007	0xF0705E07	0xF1C65807
0x01B0B800	0x3070F800	0x31C6BE00	0xC0705800	0xC1F6A600	0xF070A006	0xF1C65E00
0x01B0B801	0x3070F801	0x31C6BE01	0xC0705801	0xC1F6A601	0xF070A007	0xF1C65E01
0x01B0B806	0x3070F806	0x31C6BE06	0xC0705806	0xC1F6A606	0xF070A606	0xF1C65E06
0x01B0B807	0x3070F807	0x31C6BE07	0xC0705807	0xC1F6A607	0xF070A607	0xF1C65E07
0x01B0BE00	0x3070FE00	0x31C6E000	0xC0705E00	0xC1F6B800	0xF070B800	0xF1C6A000
0x01B0BE01	0x3070FE01	0x31C6E001	0xC0705E01	0xC1F6B801	0xF070B801	0xF1C6A001
0x01B0BE06	0x3070FE06	0x31C6E006	0xC0705E06	0xC1F6B806	0xF070B806	0xF1C6A006
0x01B0BE07	0x3070FE07	0x31C6E007	0xC0705E07	0xC1F6B807	0xF070B807	0xF1C6A007
0x01B0E006	0x30760000	0x31C6E600	0xC070A006	0xC1F6BE00	0xF070BE00	0xF1C6A600
0x01B0E007	0x30760001	0x31C6E601	0xC070A007	0xC1F6BE01	0xF070BE01	0xF1C6A601
0x01B0E606	0x30760006	0x31C6E606	0xC070A606	0xC1F6BE06	0xF070BE06	0xF1C6A606
0x01B0E607	0x30760007	0x31C6E607	0xC070A607	0xC1F6BE07	0xF070BE07	0xF1C6A607
0x01B0F800	0x30760600	0x31C6F800	0xC070B800	0xC1F6E000	0xF070E006	0xF1C6B800
0x01B0F801	0x30760601	0x31C6F801	0xC070B801	0xC1F6E001	0xF070E007	0xF1C6B801
0x01B0F806	0x30760606	0x31C6F806	0xC070B806	0xC1F6E006	0xF070E606	0xF1C6B806
0x01B0F807	0x30760607	0x31C6F807	0xC070B807	0xC1F6E007	0xF070E607	0xF1C6B807
0x01B0FE00	0x30761800	0x31C6FE00	0xC070BE00	0xC1F6E600	0xF070F800	0xF1C6BE00
0x01B0FE01	0x30761801	0x31C6FE01	0xC070BE01	0xC1F6E601	0xF070F801	0xF1C6BE01
0x01B0FE06	0x30761806	0x31C6FE06	0xC070BE06	0xC1F6E606	0xF070F806	0xF1C6BE06
0x01B0FE07	0x30761807	0x31C6FE07	0xC070BE07	0xC1F6E607	0xF070F807	0xF1C6BE07
0x01B64000	0x30761E00	0x31F00006	0xC070E006	0xC1F6F800	0xF070FE00	0xF1C6E000
0x01B64001	0x30761E01	0x31F00007	0xC070E007	0xC1F6F801	0xF070FE01	0xF1C6E001
0x01B64006	0x30761E06	0x31F00606	0xC070E606	0xC1F6F806	0xF070FE06	0xF1C6E006
0x01B64007	0x30761E07	0x31F00607	0xC070E607	0xC1F6F807	0xF070FE07	0xF1C6E007
0x01B64600	0x30764000	0x31F01800	0xC070F800	0xC1F6FE00	0xF0760000	0xF1C6E600
0x01B64601	0x30764001	0x31F01801	0xC070F801	0xC1F6FE01	0xF0760001	0xF1C6E601
0x01B64606	0x30764006	0x31F01806	0xC070F806	0xC1F6FE06	0xF0760006	0xF1C6E606
0x01B64607	0x30764007	0x31F01807	0xC070F807	0xC1F6FE07	0xF0760007	0xF1C6E607
0x01B65800	0x30764600	0x31F01E00	0xC070FE00	0xF0000006	0xF0760600	0xF1C6F800
0x01B65801	0x30764601	0x31F01E01	0xC070FE01	0xF0000007	0xF0760601	0xF1C6F801
0x01B65806	0x30764606	0x31F01E06	0xC070FE06	0xF0000606	0xF0760606	0xF1C6F806
0x01B65807	0x30764607	0x31F01E07	0xC070FE07	0xF0000607	0xF0760607	0xF1C6F807
0x01B65E00	0x30765800	0x31F04006	0xC0764000	0xF0001800	0xF0761800	0xF1C6FE00
0x01B65E01	0x30765801	0x31F04007	0xC0764001	0xF0001801	0xF0761801	0xF1C6FE01
0x01B65E06	0x30765806	0x31F04606	0xC0764006	0xF0001806	0xF0761806	0xF1C6FE06
0x01B65E07	0x30765807	0x31F04607	0xC0764007	0xF0001807	0xF0761807	0xF1C6FE07
0x01B6A000	0x30765E00	0x31F05800	0xC0764600	0xF0001E00	0xF0761E00	0xF1F00006
0x01B6A001	0x30765E01	0x31F05801	0xC0764601	0xF0001E01	0xF0761E01	0xF1F00007
0x01B6A006	0x30765E06	0x31F05806	0xC0764606	0xF0001E06	0xF0761E06	0xF1F00606
0x01B6A007	0x30765E07	0x31F05807	0xC0764607	0xF0001E07	0xF0761E07	0xF1F00607
0x01B6A600	0x3076A000	0x31F05E00	0xC0765800	0xF0004006	0xF0764000	0xF1F01800
0x01B6A601	0x3076A001	0x31F05E01	0xC0765801	0xF0004007	0xF0764001	0xF1F01801
0x01B6A606	0x3076A006	0x31F05E06	0xC0765806	0xF0004606	0xF0764006	0xF1F01806
0x01B6A607	0x3076A007	0x31F05E07	0xC0765807	0xF0004607	0xF0764007	0xF1F01807
0x01B6B800	0x3076A600	0x31F0A006	0xC0765E00	0xF0005800	0xF0764600	0xF1F01E00
0x01B6B801	0x3076A601	0x31F0A007	0xC0765E01	0xF0005801	0xF0764601	0xF1F01E01
0x01B6B806	0x3076A606	0x31F0A606	0xC0765E06	0xF0005806	0xF0764606	0xF1F01E06
0x01B6B807	0x3076A607	0x31F0A607	0xC0765E07	0xF0005807	0xF0764607	0xF1F01E07
0x01B6BE00	0x3076B800	0x31F0B800	0xC076A000	0xF0005E00	0xF0765800	0xF1F04006
0x01B6BE01	0x3076B801	0x31F0B801	0xC076A001	0xF0005E01	0xF0765801	0xF1F04007

0x01B6BE06	0x3076B806	0x31F0B806	0xC076A006	0xF0005E06	0xF0765806	0xF1F04606
0x01B6BE07	0x3076B807	0x31F0B807	0xC076A007	0xF0005E07	0xF0765807	0xF1F04607
0x01B6E000	0x3076BE00	0x31F0BE00	0xC076A600	0xF000A006	0xF0765E00	0xF1F05800
0x01B6E001	0x3076BE01	0x31F0BE01	0xC076A601	0xF000A007	0xF0765E01	0xF1F05801
0x01B6E006	0x3076BE06	0x31F0BE06	0xC076A606	0xF000A606	0xF0765E06	0xF1F05806
0x01B6E007	0x3076BE07	0x31F0BE07	0xC076A607	0xF000A607	0xF0765E07	0xF1F05807
0x01B6E600	0x3076E000	0x31F0E006	0xC076B800	0xF000B800	0xF076A000	0xF1F05E00
0x01B6E601	0x3076E001	0x31F0E007	0xC076B801	0xF000B801	0xF076A001	0xF1F05E01
0x01B6E606	0x3076E006	0x31F0E606	0xC076B806	0xF000B806	0xF076A006	0xF1F05E06
0x01B6E607	0x3076E007	0x31F0E607	0xC076B807	0xF000B807	0xF076A007	0xF1F05E07
0x01B6F800	0x3076E600	0x31F0F800	0xC076BE00	0xF000BE00	0xF076A600	0xF1F0A006
0x01B6F801	0x3076E601	0x31F0F801	0xC076BE01	0xF000BE01	0xF076A601	0xF1F0A007
0x01B6F806	0x3076E606	0x31F0F806	0xC076BE06	0xF000BE06	0xF076A606	0xF1F0A606
0x01B6F807	0x3076E607	0x31F0F807	0xC076BE07	0xF000BE07	0xF076A607	0xF1F0A607
0x01B6FE00	0x3076F800	0x31F0FE00	0xC076E000	0xF000E006	0xF076B800	0xF1F0B800
0x01B6FE01	0x3076F801	0x31F0FE01	0xC076E001	0xF000E007	0xF076B801	0xF1F0B801
0x01B6FE06	0x3076F806	0x31F0FE06	0xC076E006	0xF000E606	0xF076B806	0xF1F0B806
0x01B6FE07	0x3076F807	0x31F0FE07	0xC076E007	0xF000E607	0xF076B807	0xF1F0B807
0x01C04006	0x3076FE00	0x31F60000	0xC076E600	0xF000F800	0xF076BE00	0xF1F0BE00
0x01C04007	0x3076FE01	0x31F60001	0xC076E601	0xF000F801	0xF076BE01	0xF1F0BE01
0x01C04606	0x3076FE06	0x31F60006	0xC076E606	0xF000F806	0xF076BE06	0xF1F0BE06
0x01C04607	0x3076FE07	0x31F60007	0xC076E607	0xF000F807	0xF076BE07	0xF1F0BE07
0x01C05800	0x31800006	0x31F60600	0xC076F800	0xF000FE00	0xF076E000	0xF1F0E006
0x01C05801	0x31800007	0x31F60601	0xC076F801	0xF000FE01	0xF076E001	0xF1F0E007
0x01C05806	0x31800606	0x31F60606	0xC076F806	0xF000FE06	0xF076E006	0xF1F0E606
0x01C05807	0x31800607	0x31F60607	0xC076F807	0xF000FE07	0xF076E007	0xF1F0E607
0x01C05E00	0x31801800	0x31F61800	0xC076FE00	0xF0060000	0xF076E600	0xF1F0F800
0x01C05E01	0x31801801	0x31F61801	0xC076FE01	0xF0060001	0xF076E601	0xF1F0F801
0x01C05E06	0x31801806	0x31F61806	0xC076FE06	0xF0060006	0xF076E606	0xF1F0F806
0x01C05E07	0x31801807	0x31F61807	0xC076FE07	0xF0060007	0xF076E607	0xF1F0F807
0x01C0A006	0x31801E00	0x31F61E00	0xC1804006	0xF0060600	0xF076F800	0xF1F0FE00
0x01C0A007	0x31801E01	0x31F61E01	0xC1804007	0xF0060601	0xF076F801	0xF1F0FE01
0x01C0A606	0x31801E06	0x31F61E06	0xC1804606	0xF0060606	0xF076F806	0xF1F0FE06
0x01C0A607	0x31801E07	0x31F61E07	0xC1804607	0xF0060607	0xF076F807	0xF1F0FE07
0x01C0B800	0x31804006	0x31F64000	0xC1805800	0xF0061800	0xF076FE00	0xF1F60000
0x01C0B801	0x31804007	0x31F64001	0xC1805801	0xF0061801	0xF076FE01	0xF1F60001
0x01C0B806	0x31804606	0x31F64006	0xC1805806	0xF0061806	0xF076FE06	0xF1F60006
0x01C0B807	0x31804607	0x31F64007	0xC1805807	0xF0061807	0xF076FE07	0xF1F60007
0x01C0BE00	0x31805800	0x31F64600	0xC1805E00	0xF0061E00	0xF1800006	0xF1F60600
0x01C0BE01	0x31805801	0x31F64601	0xC1805E01	0xF0061E01	0xF1800007	0xF1F60601
0x01C0BE06	0x31805806	0x31F64606	0xC1805E06	0xF0061E06	0xF1800606	0xF1F60606
0x01C0BE07	0x31805807	0x31F64607	0xC1805E07	0xF0061E07	0xF1800607	0xF1F60607
0x01C0E006	0x31805E00	0x31F65800	0xC180A006	0xF0064000	0xF1801800	0xF1F61800
0x01C0E007	0x31805E01	0x31F65801	0xC180A007	0xF0064001	0xF1801801	0xF1F61801
0x01C0E606	0x31805E06	0x31F65806	0xC180A606	0xF0064006	0xF1801806	0xF1F61806
0x01C0E607	0x31805E07	0x31F65807	0xC180A607	0xF0064007	0xF1801807	0xF1F61807
0x01C0F800	0x3180A006	0x31F65E00	0xC180B800	0xF0064600	0xF1801E00	0xF1F61E00
0x01C0F801	0x3180A007	0x31F65E01	0xC180B801	0xF0064601	0xF1801E01	0xF1F61E01
0x01C0F806	0x3180A606	0x31F65E06	0xC180B806	0xF0064606	0xF1801E06	0xF1F61E06
0x01C0F807	0x3180A607	0x31F65E07	0xC180B807	0xF0064607	0xF1801E07	0xF1F61E07
0x01C0FE00	0x3180B800	0x31F6A000	0xC180BE00	0xF0065800	0xF1804006	0xF1F64000
0x01C0FE01	0x3180B801	0x31F6A001	0xC180BE01	0xF0065801	0xF1804007	0xF1F64001
0x01C0FE06	0x3180B806	0x31F6A006	0xC180BE06	0xF0065806	0xF1804606	0xF1F64006
0x01C0FE07	0x3180B807	0x31F6A007	0xC180BE07	0xF0065807	0xF1804607	0xF1F64007
0x01C64000	0x3180BE00	0x31F6A600	0xC180E006	0xF0065E00	0xF1805800	0xF1F64600
0x01C64001	0x3180BE01	0x31F6A601	0xC180E007	0xF0065E01	0xF1805801	0xF1F64601
0x01C64006	0x3180BE06	0x31F6A606	0xC180E606	0xF0065E06	0xF1805806	0xF1F64606
0x01C64007	0x3180BE07	0x31F6A607	0xC180E607	0xF0065E07	0xF1805807	0xF1F64607
0x01C64600	0x3180E006	0x31F6B800	0xC180F800	0xF006A000	0xF1805E00	0xF1F65800
0x01C64601	0x3180E007	0x31F6B801	0xC180F801	0xF006A001	0xF1805E01	0xF1F65801
0x01C64606	0x3180E606	0x31F6B806	0xC180F806	0xF006A006	0xF1805E06	0xF1F65806
0x01C64607	0x3180E607	0x31F6B807	0xC180F807	0xF006A007	0xF1805E07	0xF1F65807
0x01C65800	0x3180F800	0x31F6BE00	0xC180FE00	0xF006A600	0xF180A006	0xF1F65E00
0x01C65801	0x3180F801	0x31F6BE01	0xC180FE01	0xF006A601	0xF180A007	0xF1F65E01
0x01C65806	0x3180F806	0x31F6BE06	0xC180FE06	0xF006A606	0xF180A606	0xF1F65E06

0x01C65807	0x3180F807	0x31F6BE07	0xC180FE07	0xF006A607	0xF180A607	0xF1F65E07
0x01C65E00	0x3180FE00	0x31F6E000	0xC1864000	0xF006B800	0xF180B800	0xF1F6A000
0x01C65E01	0x3180FE01	0x31F6E001	0xC1864001	0xF006B801	0xF180B801	0xF1F6A001
0x01C65E06	0x3180FE06	0x31F6E006	0xC1864006	0xF006B806	0xF180B806	0xF1F6A006
0x01C65E07	0x3180FE07	0x31F6E007	0xC1864007	0xF006B807	0xF180B807	0xF1F6A007
0x01C6A000	0x31860000	0x31F6E600	0xC1864600	0xF006BE00	0xF180BE00	0xF1F6A600
0x01C6A001	0x31860001	0x31F6E601	0xC1864601	0xF006BE01	0xF180BE01	0xF1F6A601
0x01C6A006	0x31860006	0x31F6E606	0xC1864606	0xF006BE06	0xF180BE06	0xF1F6A606
0x01C6A007	0x31860007	0x31F6E607	0xC1864607	0xF006BE07	0xF180BE07	0xF1F6A607
0x01C6A600	0x31860600	0x31F6F800	0xC1865800	0xF006E000	0xF180E006	0xF1F6B800
0x01C6A601	0x31860601	0x31F6F801	0xC1865801	0xF006E001	0xF180E007	0xF1F6B801
0x01C6A606	0x31860606	0x31F6F806	0xC1865806	0xF006E006	0xF180E606	0xF1F6B806
0x01C6A607	0x31860607	0x31F6F807	0xC1865807	0xF006E007	0xF180E607	0xF1F6B807
0x01C6B800	0x31861800	0x31F6FE00	0xC1865E00	0xF006E600	0xF180F800	0xF1F6BE00
0x01C6B801	0x31861801	0x31F6FE01	0xC1865E01	0xF006E601	0xF180F801	0xF1F6BE01
0x01C6B806	0x31861806	0x31F6FE06	0xC1865E06	0xF006E606	0xF180F806	0xF1F6BE06
0x01C6B807	0x31861807	0x31F6FE07	0xC1865E07	0xF006E607	0xF180F807	0xF1F6BE07
0x01C6BE00	0x31861E00	0xC0004006	0xC186A000	0xF006F800	0xF180FE00	0xF1F6E000
0x01C6BE01	0x31861E01	0xC0004007	0xC186A001	0xF006F801	0xF180FE01	0xF1F6E001
0x01C6BE06	0x31861E06	0xC0004606	0xC186A006	0xF006F806	0xF180FE06	0xF1F6E006
0x01C6BE07	0x31861E07	0xC0004607	0xC186A007	0xF006F807	0xF180FE07	0xF1F6E007
0x01C6E000	0x31864000	0xC0005800	0xC186A600	0xF006FE00	0xF1860000	0xF1F6E600
0x01C6E001	0x31864001	0xC0005801	0xC186A601	0xF006FE01	0xF1860001	0xF1F6E601
0x01C6E006	0x31864006	0xC0005806	0xC186A606	0xF006FE06	0xF1860006	0xF1F6E606
0x01C6E007	0x31864007	0xC0005807	0xC186A607	0xF006FE07	0xF1860007	0xF1F6E607
0x01C6E600	0x31864600	0xC0005E00	0xC186B800	0xF0300006	0xF1860600	0xF1F6F800
0x01C6E601	0x31864601	0xC0005E01	0xC186B801	0xF0300007	0xF1860601	0xF1F6F801
0x01C6E606	0x31864606	0xC0005E06	0xC186B806	0xF0300606	0xF1860606	0xF1F6F806
0x01C6E607	0x31864607	0xC0005E07	0xC186B807	0xF0300607	0xF1860607	0xF1F6F807
0x01C6F800	0x31865800	0xC000A006	0xC186BE00	0xF0301800	0xF1861800	0xF1F6FE00
0x01C6F801	0x31865801	0xC000A007	0xC186BE01	0xF0301801	0xF1861801	0xF1F6FE01
0x01C6F806	0x31865806	0xC000A606	0xC186BE06	0xF0301806	0xF1861806	0xF1F6FE06
0x01C6F807	0x31865807	0xC000A607	0xC186BE07	0xF0301807	0xF1861807	0xF1F6FE07

7.3.11.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper10

Fold-down coefficients for virtual speakers in the equatorial plane are derived from Table 32.

Table 32: GroupEquatorOrUpper10 Fold-down Coefficients for Virtual Speakers in Equatorial Plane

Physical Speakers `eqtr_functions`	PrefOrder (Lhr, Split (Lss,Chr,-3 dB), ClosestElv (0, ClosestAzUH(-90)))	PrefOrder (Rhr, Split (Rss,Chr,-3 dB), ClosestElv (0, ClosestAzUH(90)))
Virtual Speakers `eqtr_virt_srcs`	eqtr_folddown[][]	
(-135,0,1)	0 dB	
(135,0,1)		0 dB

Since there are no virtual speakers in the upper hemisphere, set the corresponding arrays to an empty list.

```
uphemi_functions = [];
uphemi_virt_srcs = [];
```

7.3.12 GroupEquatorOrUpper11

7.3.12.1 Channel Layouts in GroupEquatorOrUpper11

Channel layout bitmasks that use `GroupEquatorOrUpper11` fold-down coefficients are listed in Table 33.

Table 33: GroupEquatorOrUpper11 list_of_channel_masks[]

0x0000001E	0x0000181E	0x0006001E	0x0006181E
0x0000001F	0x0000181F	0x0006001F	0x0006181F

7.3.12.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper11

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 34.

Table 34: GroupEquatorOrUpper11 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	L	R	Ls	Rs
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]			
(-45, 45,1)	0,0 dB			
(45, 45,1)		0,0 dB		
(-135, 45,1)			-0,4576 dB	-10 dB
(135, 45,1)			-10 dB	-0,4576 dB

Fold-down coefficients for virtual speakers in lower hemisphere are defined in clause 7.4.5.

7.3.13 GroupEquatorOrUpper12

7.3.13.1 Channel Layouts in GroupEquatorOrUpper12

Channel layout bitmasks that use `GroupEquatorOrUpper12` fold-down coefficients are listed in Table 35.

Table 35: GroupEquatorOrUpper12 list_of_channel_masks[]

0x0000005E	0x0000185E	0x0006005E	0x0006185E
0x0000005F	0x0000185F	0x0006005F	0x0006185F

7.3.13.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper12

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 36.

Table 36: GroupEquatorOrUpper12 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	L	R	Ls	Rs	Cs
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]				
(-45, 45,1)	0,0 dB				
(45, 45,1)		0,0 dB			
(-135, 45,1)			-3,0103 dB		-3,0103 dB
(135, 45,1)				-3,0103 dB	-3,0103 dB

Fold-down coefficients for virtual speakers in lower hemisphere are defined in clause 7.4.6.

7.3.14 GroupEquatorOrUpper13

7.3.14.1 Channel Layouts in GroupEquatorOrUpper13

Channel layout bitmasks that use `GroupEquatorOrUpper13` fold-down coefficients are listed in Table 37.

Table 37: GroupEquatorOrUpper13 list_of_channel_masks[]

0x0000019E	0x0000199E	0x0006019E	0x0006199E
0x0000019F	0x0000199F	0x0006019F	0x0006199F
0x000001DE	0x000019DE	0x000601DE	0x000619DE
0x000001DF	0x000019DF	0x000601DF	0x000619DF
0x00000786	0x00001F86	0x00060786	0x00061F86
0x00000787	0x00001F87	0x00060787	0x00061F87
0x000007C6	0x00001FC6	0x000607C6	0x00061FC6
0x000007C7	0x00001FC7	0x000607C7	0x00061FC7

7.3.14.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper13

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 38.

Table 38: GroupEquatorOrUpper13 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	L	R	Ls	Rs
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]			
(-45, 45,1)	0,0 dB			
(45, 45,1)		0,0 dB		
(-135, 45,1)			0,0 dB	
(135, 45,1)				0,0 dB

Fold-down coefficients for virtual speakers in lower hemisphere are defined in clause 7.4.7.

7.3.15 GroupEquatorOrUpper14

7.3.15.1 Channel Layouts in GroupEquatorOrUpper14

Channel layout bitmasks that use `GroupEquatorOrUpper14` fold-down coefficients are listed in Table 39.

Table 39: GroupEquatorOrUpper14 list_of_channel_masks[]

0x0000A01E	0x0000A01F	0x3000001E	0x3000001F

7.3.15.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper14

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 40.

Table 40: GroupEquatorOrUpper14 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	Ls	Rs
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]	
(-135, 45,1)	-0,4576 dB	-10 dB
(135, 45,1)	-10 dB	-0,4576 dB

7.3.16 GroupEquatorOrUpper15

7.3.16.1 Channel Layouts in GroupEquatorOrUpper15

Channel layout bitmasks that use fold-down coefficients in `GroupEquatorOrUpper15` are defined in Table 41.

Table 41: GroupEquatorOrUpper15 list_of_channel_masks[]

0x0030001E	0x0030001F

7.3.16.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper15

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 42.

Table 42: GroupEquatorOrUpper15 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	L	R	Ls	Rs	ExclusiveOr (Lhs,Ltm)	ExclusiveOr (Rhs,Rtm)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]					
(-45, 45,1)	-3,0103 dB				-3,0103 dB	
(45, 45,1)		-3,0103 dB				-3,0103 dB
(-135, 45,1)			-4,6976 dB	-13,7828 dB	-2,0823 dB	
(135, 45,1)			-13,7828 dB	-4,6976 dB		-2,0823 dB

7.3.17 GroupEquatorOrUpper16

7.3.17.1 Channel Layouts in GroupEquatorOrUpper16

Channel layout bitmasks that use fold-down coefficients in `GroupEquatorOrUpper16` are listed in Table 43.

Table 43: GroupEquatorOrUpper16 list_of_channel_masks[]

0x0040401E	0x0040401F

7.3.17.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper16

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 44.

Table 44: GroupEquatorOrUpper16 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	L	R	Ls	Rs	Ch	Chr
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]					
(-45, 45,1)	-3,0103 dB				-3,0103 dB	
(45, 45,1)		-3,0103 dB			-3,0103 dB	
(-135, 45,1)			-3,0103 dB			-3,0103 dB
(135, 45,1)				-3,0103 dB		-3,0103 dB

7.3.18 GroupEquatorOrUpper17

7.3.18.1 Channel Layouts in GroupEquatorOrUpper17

Channel layout bitmasks that use `GroupEquatorOrUpper17` fold-down coefficients are listed in Table 45.

Table 45: GroupEquatorOrUpper17 list_of_channel_masks[]

0x0040A01E	0x0040A01F	0x3040001E	0x3040001F

7.3.18.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper17

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 46.

Table 46: GroupEquatorOrUpper17 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	Ls	Rs	Chr
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]		
(-135, 45,1)	-3,0103 dB		-3,0103 dB
(135, 45,1)		-3,0103 dB	-3,0103 dB

7.3.19 GroupEquatorOrUpper18

7.3.19.1 Channel Layouts in GroupEquatorOrUpper18

Channel layout bitmasks that use `GroupEquatorOrUpper18` fold-down coefficients are listed in Table 47.

Table 47: GroupEquatorOrUpper18 list_of_channel_masks[]

0x0000A19E	0x0006A19E	0x3000019E	0x3006019E
0x0000A19F	0x0006A19F	0x3000019F	0x3006019F
0x0000A786	0x0006A786	0x30000786	0x30060786
0x0000A787	0x0006A787	0x30000787	0x30060787

7.3.19.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper18

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 48.

Table 48: GroupEquatorOrUpper18 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	Lsr	Rsr
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]	
(-135, 45,1)	0,0 dB	
(135, 45,1)		0,0 dB

7.3.20 GroupEquatorOrUpper19

7.3.20.1 Channel Layouts in GroupEquatorOrUpper19

Channel layout bitmasks that use `GroupEquatorOrUpper19` fold-down coefficients are listed in Table 49.

Table 49: GroupEquatorOrUpper19 list_of_channel_masks[]

0x0030019E	0x00300786	0x0036019E	0x00360786
0x0030019F	0x00300787	0x0036019F	0x00360787

7.3.20.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper19

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 50.

Table 50: GroupEquatorOrUpper19 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	L	R	Lsr	Rsr	ExclusiveOr (Lhs,Ltm)	ExclusiveOr (Rhs,Rtm)
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]					
(-45, 45,1)	-3,0103 dB				-3,0103 dB	
(45, 45,1)		-3,0103 dB				-3,0103 dB
(-135, 45,1)			-3,0103 dB		-3,0103 dB	
(135, 45,1)				-3,0103 dB		-3,0103 dB

7.3.21 GroupEquatorOrUpper20

7.3.21.1 Channel Layouts in GroupEquatorOrUpper20

Channel layout bitmasks that use `GroupEquatorOrUpper20` fold-down coefficients are listed in Table 51.

Table 51: GroupEquatorOrUpper20 list_of_channel_masks[]

0x0040419E	0x00404786	0x0046419E	0x00464786
0x0040419F	0x00404787	0x0046419F	0x00464787

7.3.21.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper20

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 52.

Table 52: GroupEquatorOrUpper20 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers uphemi_functions	L	R	Lsr	Rsr	Ch	Chr
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]					
(-45, 45,1)	-3,0103 dB				-3,0103 dB	
(45, 45,1)		-3,0103 dB			-3,0103 dB	
(-135, 45,1)			-3,0103 dB			-3,0103 dB
(135, 45,1)				-3,0103 dB		-3,0103 dB

7.3.22 GroupEquatorOrUpper21

7.3.22.1 Channel Layouts in GroupEquatorOrUpper21

Channel layout bitmasks that use `GroupEquatorOrUpper21` fold-down coefficients are listed in Table 53.

Table 53: GroupEquatorOrUpper21 list_of_channel_masks[]

0x0040A19E	0x0046A19E	0x3040019E	0x3046019E
0x0040A19F	0x0046A19F	0x3040019F	0x3046019F
0x0040A786	0x0046A786	0x30400786	0x30460786
0x0040A787	0x0046A787	0x30400787	0x30460787

7.3.22.2 Virtual Speakers and their Fold-down Coefficients for GroupEquatorOrUpper21

Since there are no virtual speakers in the equatorial plane, set the corresponding arrays to an empty list.

```
eqtr_functions = [];
eqtr_virt_srcs = [];
```

Fold-down coefficients for virtual speakers in the upper hemisphere are derived from Table 54.

Table 54: GroupEquatorOrUpper21 Fold-down Coefficients for Virtual Speakers in Upper Hemisphere

Physical Speakers `uphemi_functions`	Lsr	Rsr	Chr
Virtual Speakers `uphemi_virt_srcs`	uphemi_folddown[][]		
(-135, 45,1)	-3,0103 dB		-3,0103 dB
(135, 451,)		-3,0103 dB	-3,0103 dB

7.4 Lower Hemisphere

7.4.1 Conditions for Lower Hemisphere Groups

The custom virtual sources for lower hemisphere were separated as there are only five tables shared with multiple groups defined in clause 7.3. For all the tables, the virtual sources in the lower hemisphere is fixed:

$$V_l = [(-45°, -45°, 1), (45°, -45°, 1), (-135°, -45°, 1), (135°, -45°, 1)]$$

The channel layout masks defined in the lower hemisphere groups require that the upper hemisphere channels be masked to 0. The function `GetChannelLayoutMaskForLowerHemisphere()` masks the LFE and upper hemisphere channels to zero.

7.4.2 Functions for Virtual Speakers in the Lower Hemisphere

7.4.2.1 GetChannelLayoutMaskForLowerHemisphere

`GetChannelLayoutMaskForLowerHemisphere()`converts physical speaker layout into a masked layout for the look up tables in clause 7.4.2.2. The channel masks created by `GetChannelLayoutMaskForLowerHemisphere()` find the virtual sources and corresponding fold-down coefficients according to the bitmask corresponding to their respective lower grouping, found in clauses 7.4.3 to 7.4.7.

```
GetChannelLayoutMaskForLowerHemisphere( physical_speaker_channel_layout_mask )
{
    // Defining masking flags that will be masked to zero for lower hemisphere.
    mask_lfe = LFE1 | LFE2;
    mask_upper_hemisphere = Lh | Rh | Ch | Lhr | Chr | Rhr | Lhs | Rhs | Oh | Ltf | Rtf | Ltr | Rtr;

    // Take negation of both lfe & upper hemisphere masks
    mask_to_zero = (~mask_lfe) | (~mask_upper_hemisphere);

    // Apply conjunction to make lfe's and upper hemisphere channel masks zero.
    channel_layout_mask_for_lower_hemisphere =
    physical_speaker_channel_layout_mask & mask_to_zero;

    // return the resultant channel mask.
    return channel_layout_mask_for_lower_hemisphere;
};
```

7.4.2.2 GroupLower

The following structure gives a concrete representation for a group in the lower hemisphere virtual speakers.

```
struct GroupLower
{
    // An array listing all the channel masks supported by the group.
    uint32 list_of_channel_masks[];

    // An array of lower hemisphere virtual speakers used by the group.
    Point lohemi_virt_srcs[];

    // An array of function pointers to determine the lower hemisphere virtual speaker index.
    FunctionP lohemi_functions[];

    // Table of fold-down coefficients for the lower hemisphere virtual speakers.
    GainCoefficient_dB lohemi_folddown[][];

    // Get linear gain vector for virtual sources in lower hemisphere
    GetLinearGainMatrixLoHemi(Point speaker_locations[]);
};

GroupEquatorOrUpper::GetLinearGainMatrixLoHemi(Point speaker_locations[])
{
    out_gain_matrix = [][];

    for ( virtual_p_idx : range(lohemi_virt_srcs.size()) )
    {
        out_gain_matrix[virtual_p_idx] =
            GetLinearGainVectorFromTable(
                lohemi_virt_srcs,
                lohemi_functions,
                virtual_p_idx,
                speaker_locations );
    }
    return out_gain_matrix;
}
```

7.4.2.3 list_of_all_groups_lower

The following list defines all the supported groups for custom virtual speakers in the upper hemisphere and the equatorial plane.

```
list_of_all_groups_lower = {
    GroupLower1,
    GroupLower2,
    GroupLower3,
    GroupLower4,
    GroupLower5
};
```

7.4.3 GroupLower1

7.4.3.1 Channel layouts in GroupLower1

Channel layout bitmasks that use `GroupLower1` fold-down coefficients are listed in Table 55.

Table 55: GroupLower1 list_of_channel_masks[]

0x00000006	0x00001800	0x00060000

7.4.3.2 Virtual Speakers and their Fold-down Coefficients for GroupLower1

Fold-down coefficients for virtual speakers in the lower hemisphere are derived from Table 56.

Table 56: GroupLower1 Fold-down Coefficients for Virtual Speakers in Lower Hemisphere

Physical Speakers `lohemi_functions[]`	ExclusiveOr (L,Lw,Lc)	ExclusiveOr (R,Rw,Rc)
Virtual Speakers `lohemi_virt_srcs[]`	lohemi_folddown[][]	
(-45, -45,1)	0,0 dB	
(45, -45,1)		0,0 dB
(-135, -45,1)	0,0 dB	
(135, -45,1)		0,0 dB

7.4.4 GroupLower2

7.4.4.1 Channel layouts in GroupLower2

Channel layout bitmasks that use `GroupLower2` fold-down coefficients are listed in Table 57.

Table 57: GroupLower2 list_of_channel_masks[]

0x00000007	0x00001807	0x00060001	0x00060606	0x00061807
0x00000606	0x00001E00	0x00060006	0x00060607	0x00061E00
0x00000607	0x00001E01	0x00060007	0x00061800	0x00061E01
0x00001801	0x00001E06	0x00060600	0x00061801	0x00061E06
0x00001806	0x00001E07	0x00060601	0x00061806	0x00061E07

7.4.4.2 Virtual Speakers and their Fold-down Coefficients for GroupLower2

Fold-down coefficients for virtual speakers in the lower hemisphere are derived from Table 58.

Table 58: GroupLower2 Fold-down Coefficients for Virtual Speakers in Lower Hemisphere

Physical Speakers `lohemi_functions[]`	ClosestAzEq (-40)	ClosestAzEq(40)	LeftLargestEqtr()	RightLargestEqtr()
Virtual Speakers `lohemi_virt_srcs[]`	lohemi_folddown[][]			
(-45, -45,1)	0,0 dB			
(45, -45,1)		0,0 dB		
(-135, -45,1)			0,0 dB	
(135, -45,1)				0,0 dB

7.4.5 GroupLower3

7.4.5.1 Channel layouts in GroupLower3

Channel layout bitmasks that use `GroupLower3` fold-down coefficients are listed in Table 59.

Table 59: GroupLower3 list_of_channel_masks[]

0x0000001E	0x0000181E	0x0006001E	0x0006181E
0x0000001F	0x0000181F	0x0006001F	0x0006181F

7.4.5.2 Virtual Speakers and their Fold-down Coefficients for GroupLower3

Fold-down coefficients for virtual speakers in the lower hemisphere are derived from Table 60.

Table 60: GroupLower3 Fold-down Coefficients for Virtual Speakers in Lower Hemisphere

Physical Speakers `lohemi_functions[]`	L	R	Ls	Rs
Virtual Speakers `lohemi_virt_srcs[]`	lohemi_folddown[][]			
(-45, -45,1)	0 dB			
(45, -45,1)		0 dB		
(-135, -45,1)			-0,4576 dB	-10 dB
(135, -45,1)			-10,0 dB	-0,4576 dB

7.4.6 GroupLower4

7.4.6.1 Channel layouts in GroupLower4

Channel layout bitmasks that use `GroupLower4` fold-down coefficients are listed in Table 61.

Table 61: GroupLower4 list_of_channel_masks[]

0x0000005E	0x0000185E	0x0006005E	0x0006185E
0x0000005F	0x0000185F	0x0006005F	0x0006185F

7.4.6.2 Virtual Speakers and their Fold-down Coefficients for GroupLower4

Fold-down coefficients for virtual speakers in the lower hemisphere are derived from Table 62.

Table 62: GroupLower4 Fold-down Coefficients for Virtual Speakers in Lower Hemisphere

Physical Speakers `lohemi_functions[]`	L	R	Ls	Rs	Cs
Virtual Speakers `lohemi_virt_srcs[]`	lohemi_folddown[][]				
(-45, -45,1)	0 dB				
(45, -45,1)		0 dB			
(-135, -45,1)			-3,0103 dB		-3,0103 dB
(135, -45,1)				-3,0103 dB	-3,0103 dB

7.4.7 GroupLower5

7.4.7.1 Channel Layouts in GroupLower5

Channel layout bitmasks that use `GroupLower5` fold-down coefficients are listed in Table 63.

Table 63: GroupLower5 list_of_channel_masks[]

0x0000019E	0x0000199E	0x0006019E	0x0006199E
0x0000019F	0x0000199F	0x0006019F	0x0006199F
0x000001DE	0x000019DE	0x000601DE	0x000619DE
0x000001DF	0x000019DF	0x000601DF	0x000619DF
0x00000786	0x00001F86	0x00060786	0x00061F86
0x00000787	0x00001F87	0x00060787	0x00061F87
0x000007C6	0x00001FC6	0x000607C6	0x00061FC6
0x000007C7	0x00001FC7	0x000607C7	0x00061FC7

7.4.7.2 Virtual Speakers and their Fold-down Coefficients for GroupLower5

Fold-down coefficients for virtual speakers in the lower hemisphere are derived from Table 64.

Table 64: GroupLower5 Fold-down Coefficients for Virtual Speakers in Lower Hemisphere

Physical Speakers `lohemi_functions[]`	L	R	Lsr	Rsr
Virtual Speakers `lohemi_virt_srcs[]`	lohemi_folddown[][]			
(-45, -45,1)	0 dB			
(45, -45,1)		0 dB		
(-135, -45,1)			0 dB	
(135, -45,1)				0 dB

History

Document history		
V1.1.1	January 2018	Publication