

FIG. 1

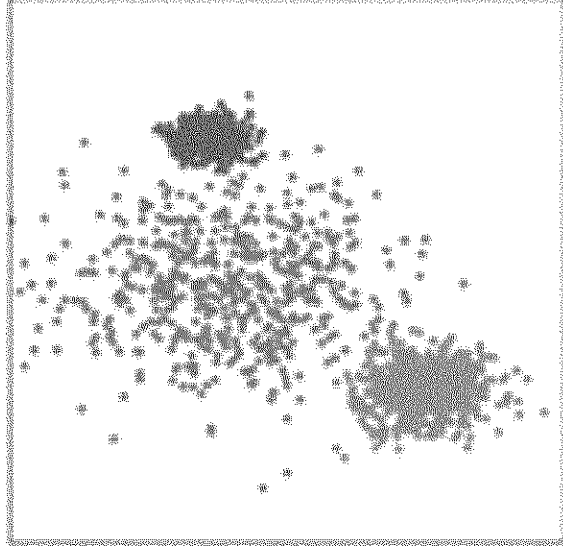
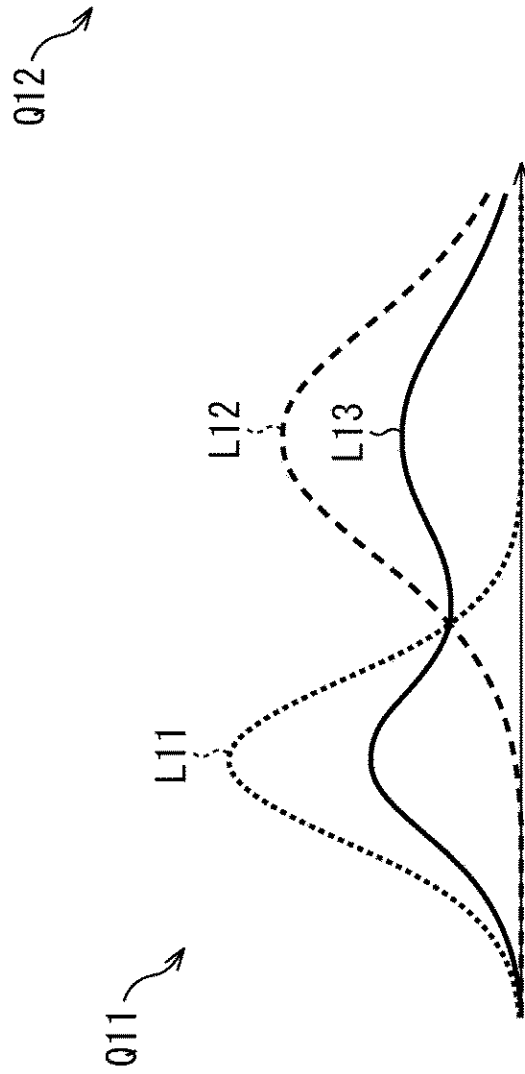


FIG. 2

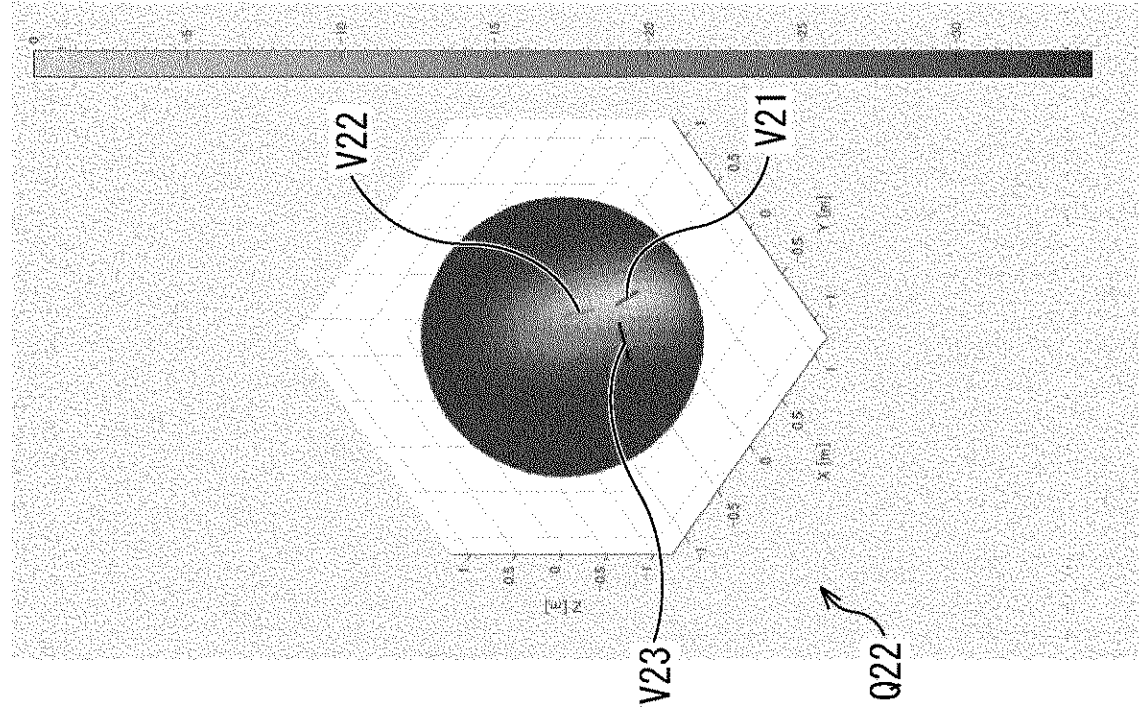
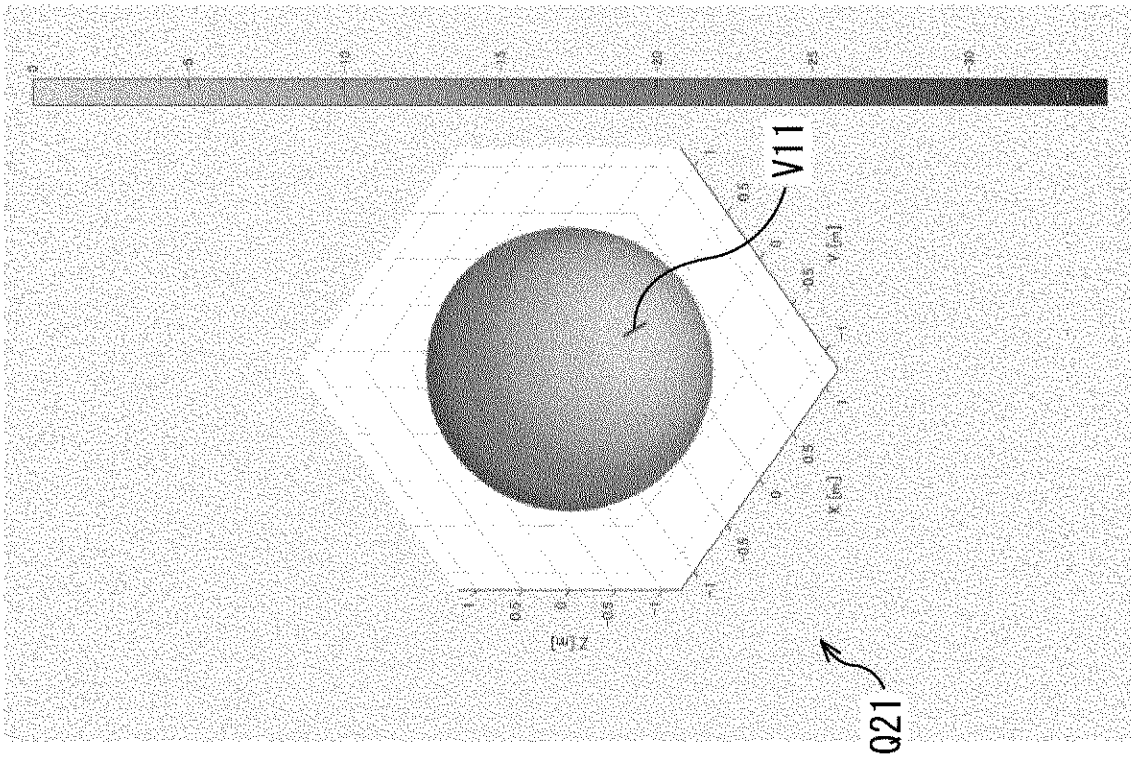


FIG. 3

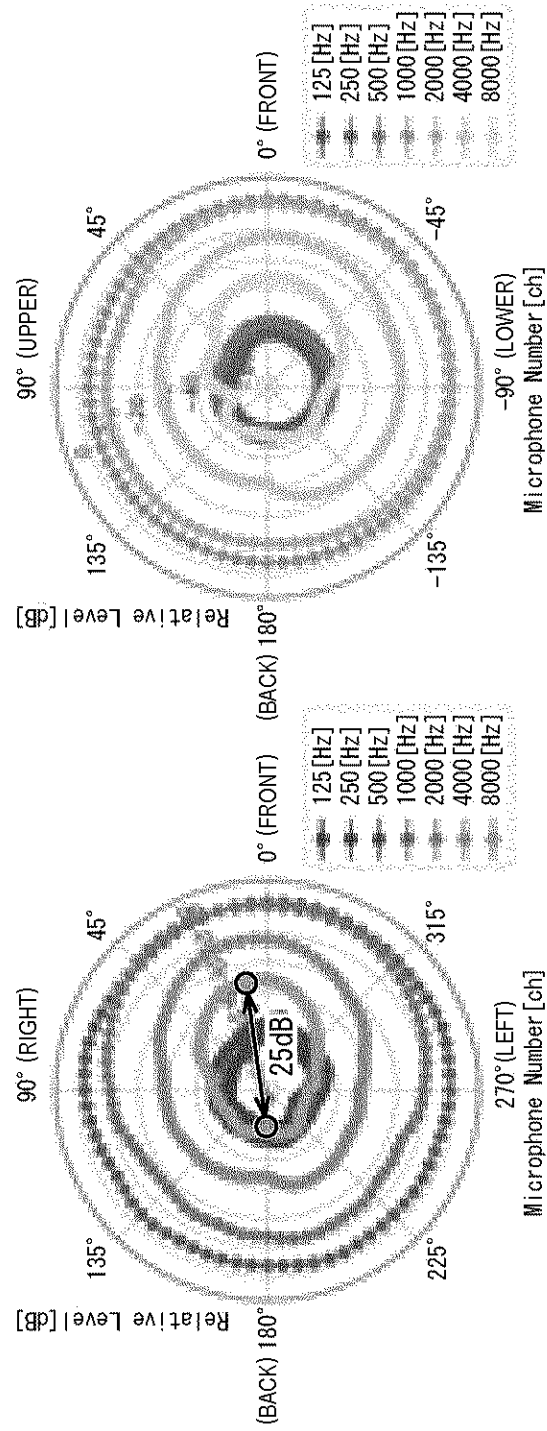


FIG. 4

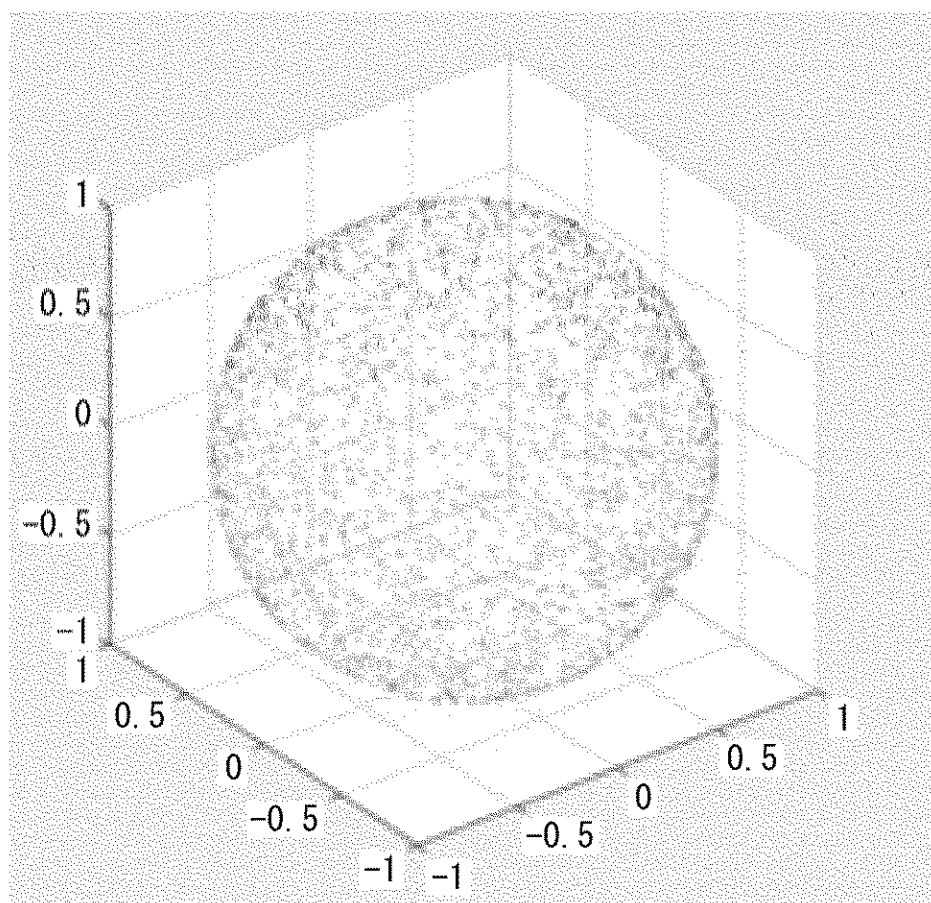


FIG. 5

Syntax	No. of bits	Data format
directivityConfig(num_sound_types_id)		
{		
fs	32	unsigned int
num_point_indices: // NUMBER OF DATA POINTS	32	unsigned int
for (i=0; i<num_point_indices; i++) {		
azimuth_table[i] // AZIMUTH ANGLE	32	float
elevation_table[i] // ELEVATION ANGLE	32	float
distance[i] (radius) // DISTANCE [meter]	32	float
}		
bin_count // NUMBER OF FREQUENCY POINTS	16	unsigned short
for (i_bin=0; i_bin<bin_count; i_bin++){		
freq[i_bin] // FREQUENCY [Hz]	32	unsigned int
}		
// vMF/kent distribution parameter		
band_count // NUMBER OF bands TO BE MODELED	16	unsigned short
for (i_band = 0; i_band<band_count; i_band++){		
mix_count[i_band] // NUMBER OF MIXTURES IN EACH band	16	unsigned short
bin_range_per_band[i_band] // bin INFORMATION ABOUT ORIGINAL DATA INCLUDED IN EACH band	16	unsigned short
for (i_mix=0; i_mix<mix_count[i_band]; i_mix++){		
weight[i_band][i_mix]	32	float
keppa[i_band][i_mix]	32	float
gamma1[i_band][i_mix][x]	32	float
gamma1[i_band][i_mix][y]	32	float
dist_flag // Kent DISTRIBUTION/v MF DISTRIBUTION SELECTION flag	8	unsigned char
if (dist_flag==1) {		
beta[i_band][i_mix]	32	float
gamma2[i_band][i_mix][x]	32	float
gamma2[i_band][i_mix][y]	32	float
gamma3[i_band][i_mix][x]	32	float
gamma3[i_band][i_mix][y]	32	float
} //flag		
} //mix		
} //band		
for (i_bin=0; i_bin < bin_count; i_bin++){		
scale_factor[i_bin] // DYNAMIC RANGE OF DIRECTIVITY DATA IN EACH bin	32	float
offset[i_bin] // DIRECTIVITY DATA MINIMUM VALUE IN EACH bin	32	float
} //bin		
for (i_point=0; i_point<num_point) {		
diff_data[i_point] // DIFFERENCE INFORMATION	TBD	
}		
}		

FIG. 6

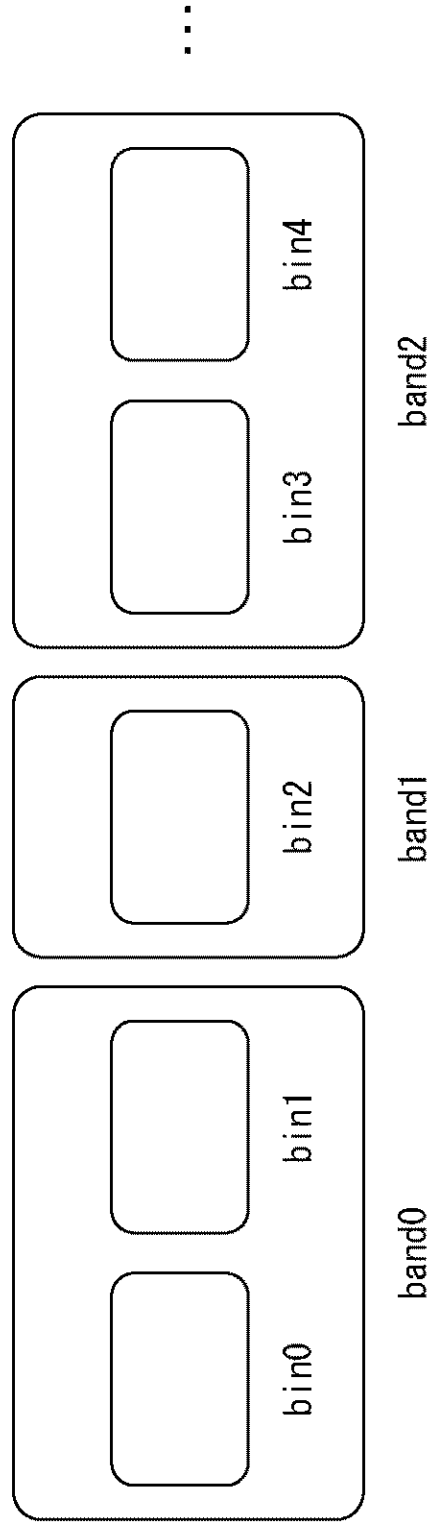


FIG. 7

mixture	1	12	19
	band1	band2	band3
weight	32	384	608
kappa	32	384	608
mean	96	1152	1824
offset	320	384	224
scale factor	320	384	224
Total (KB)	0.097656	0.328125	0.42578125
			0.8515625

Original data	Modeled by mixture of vMFdistribution
306KB	0.85KB

FIG. 8

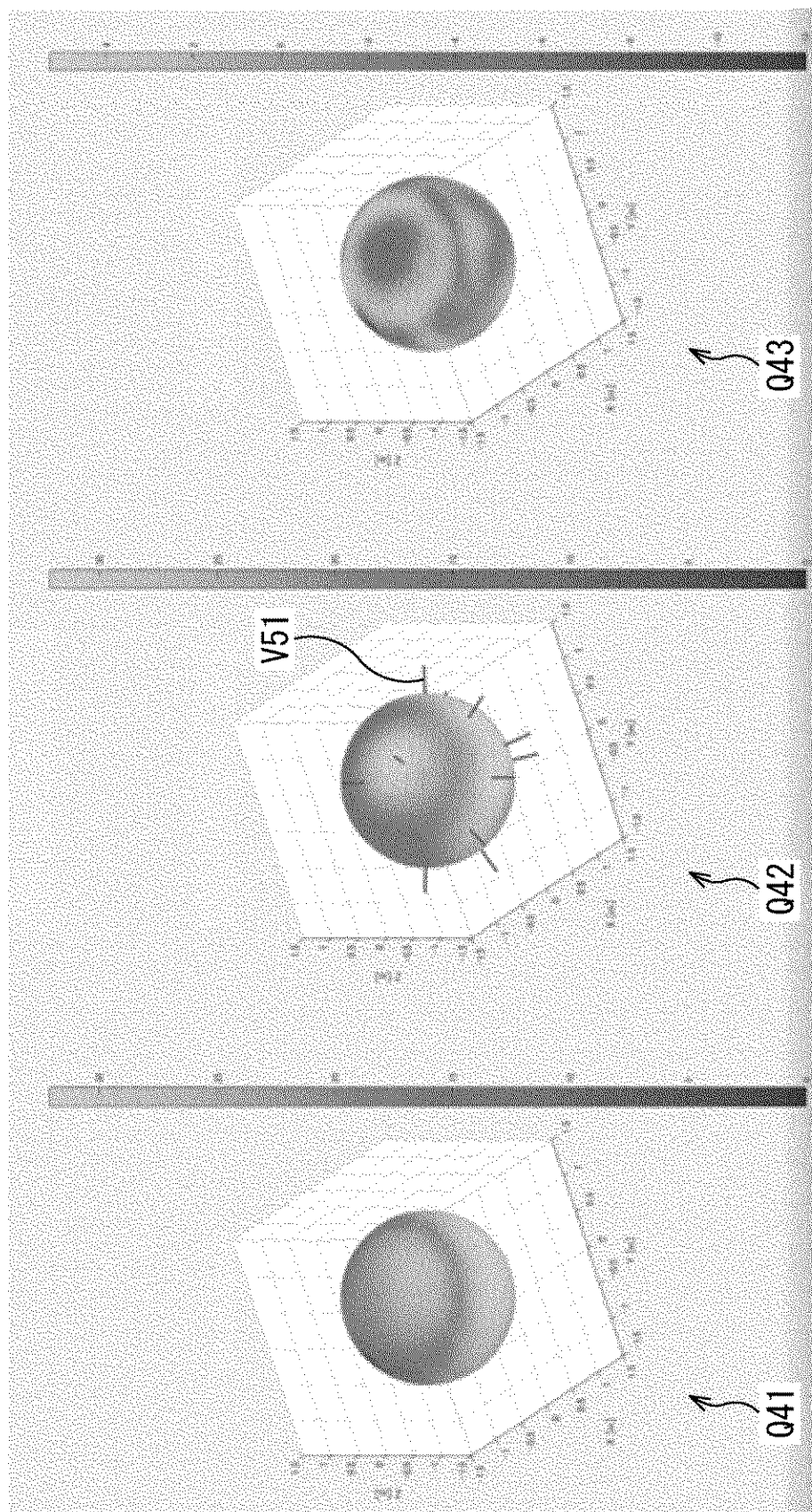


FIG. 9

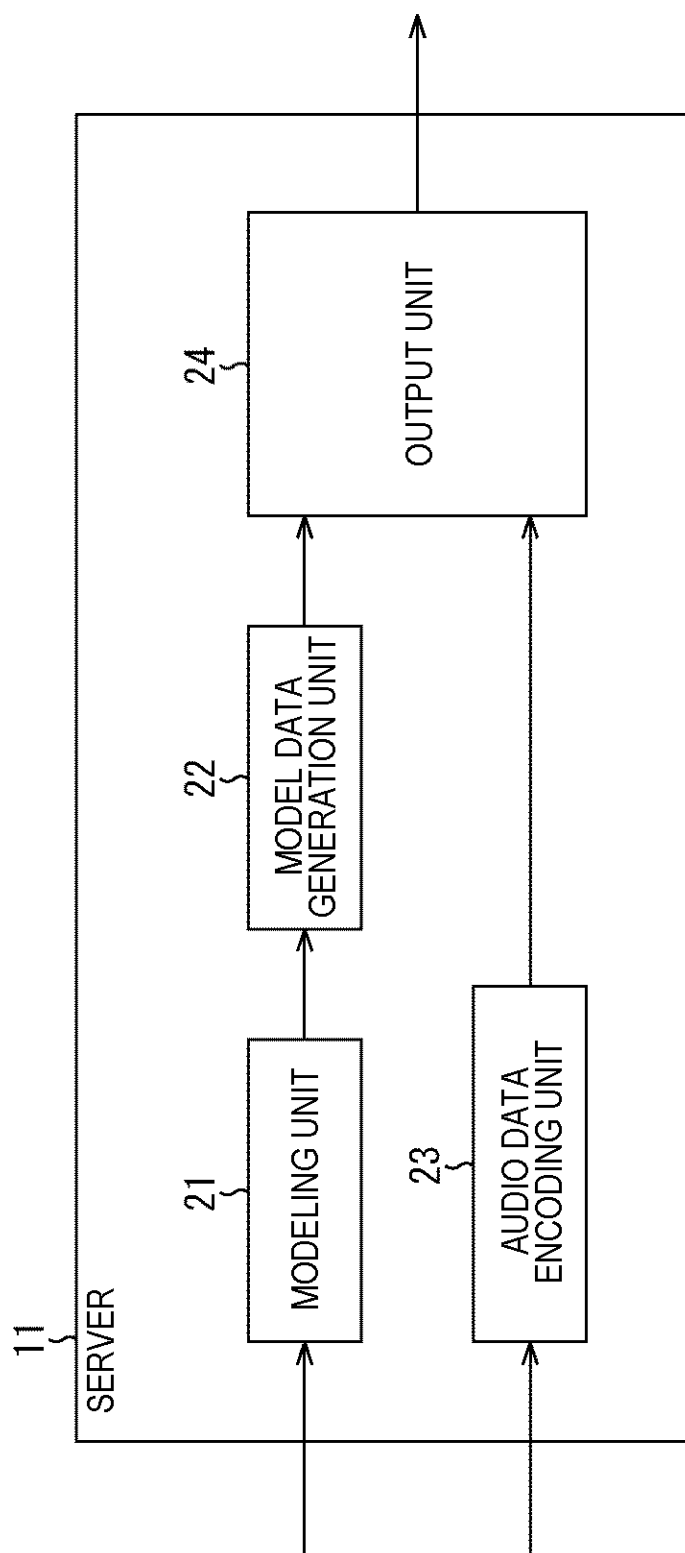


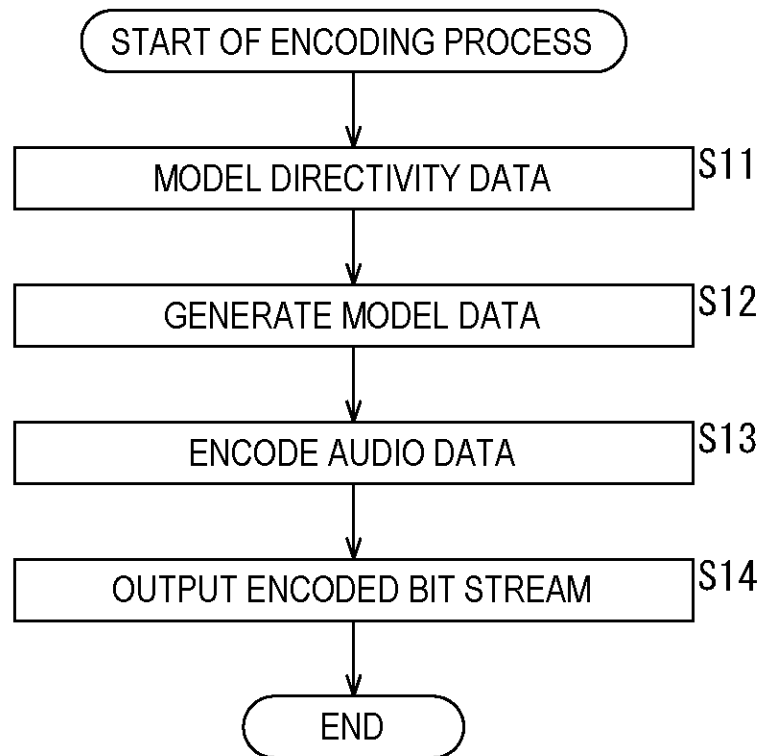
FIG. 10

FIG. 11

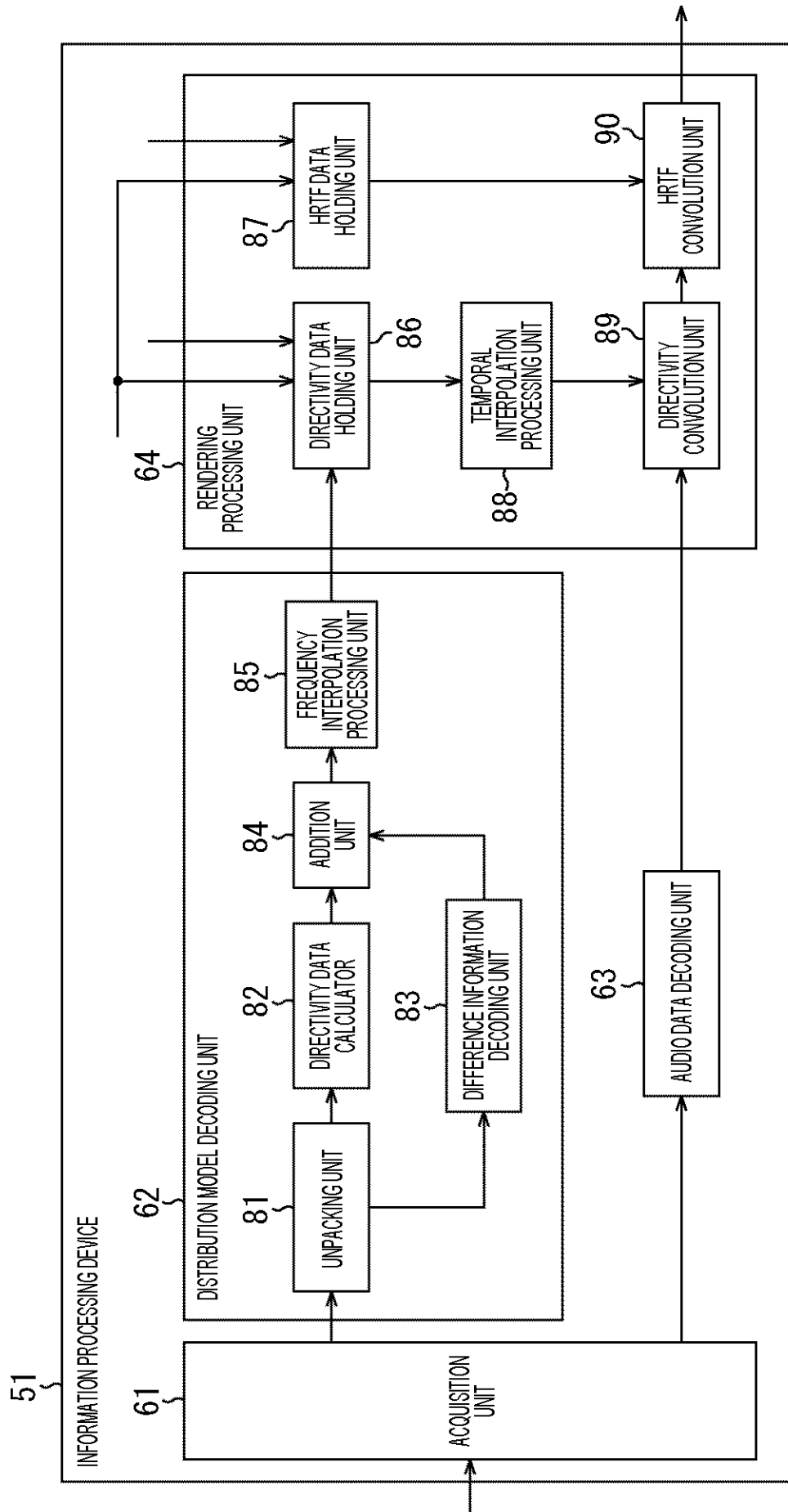


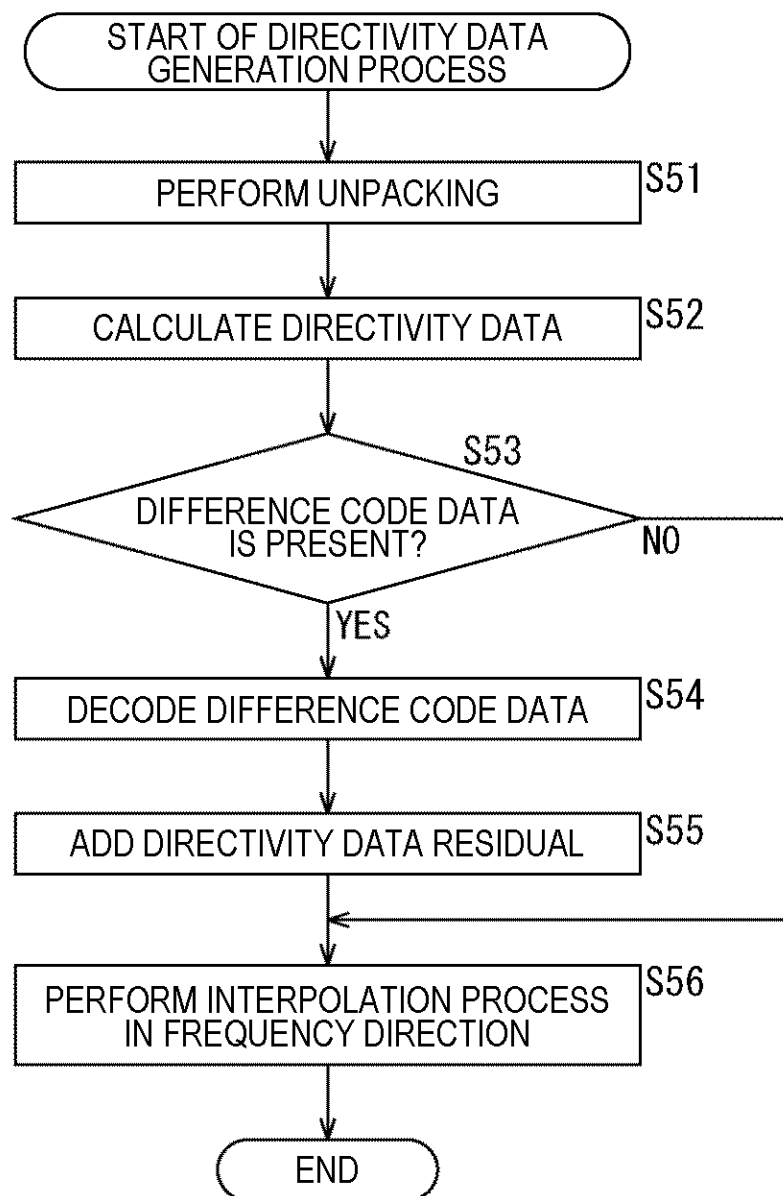
FIG. 12

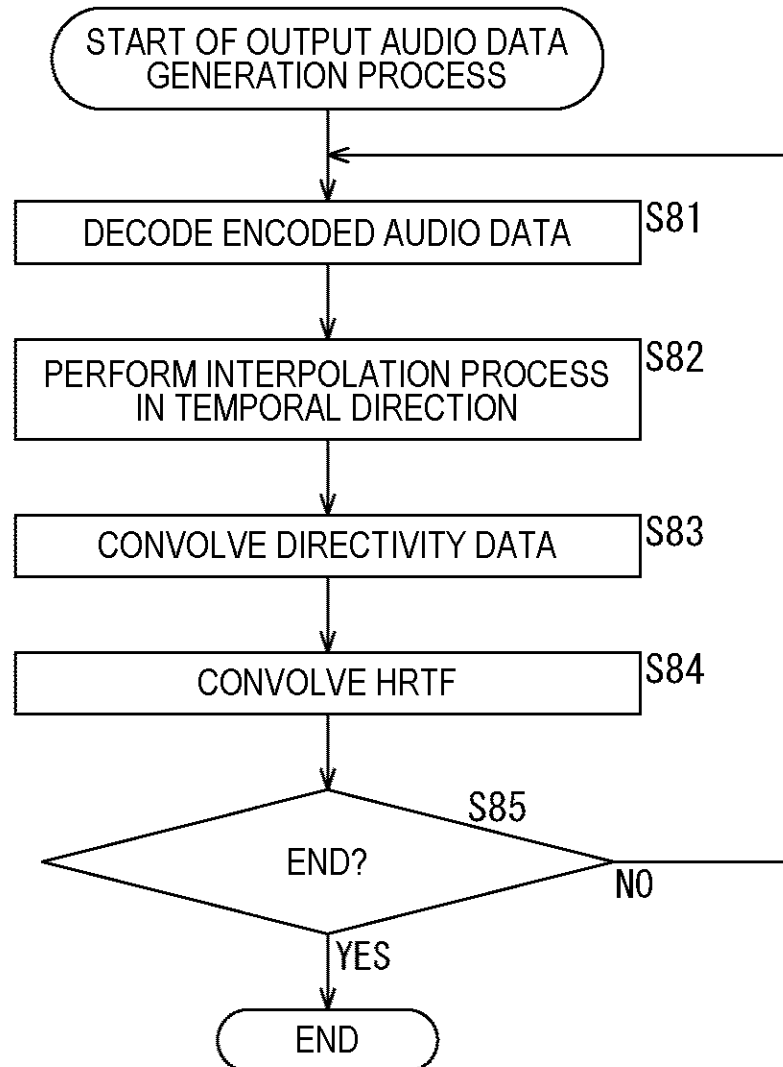
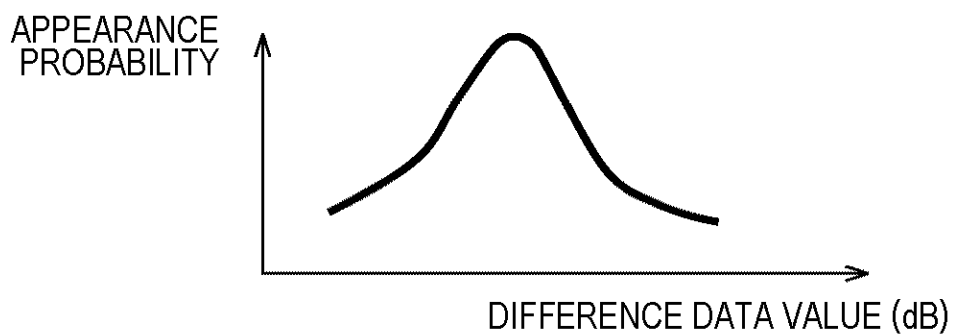
FIG. 13**FIG. 14**

FIG. 15

Syntax	No. of bits	Data format
<pre> directivityConfig(num_sound_types_id) { fs num_point_indices; // NUMBER OF DATA POINTS for (i = 0; i < num_point_indices; i++) { azimuth_table[i] // ELEVATION ANGLE elevation_table[i] // AZIMUTH ANGLE distance[i] (radius) // DISTANCE [meter] } bin_count // NUMBER OF FREQUENCY POINTS for (i_bin=0; i_bin<bin_count; i_bin++){ freq[i_bin] // FREQUENCY [Hz] } // vMF/kent distribution parameter band_count // NUMBER OF bands TO BE MODELED for (i_band = 0; i_band < band_count; i_band++){ mix_count[i_band] // NUMBER OF MIXTURES IN EACH band bin_range_per_band[i_band] // bin INFORMATION ABOUT ORIGINAL DATA INCLUDED IN EACH band for (i_mix = 0; i_mix < mix_count[i_band]; i_mix++){ weight[i_band][i_mix] 32 float kappa[i_band][i_mix] 32 float gamma1[i_band][i_mix][x] 32 float gamma1[i_band][i_mix][y] 32 float dist_flag // Kent DISTRIBUTION/vMF DISTRIBUTION SELECTION flag 1 bool if(dist_flag == 1){ beta[i_band][i_mix] 32 float gamma2[i_band][i_mix][x] 32 float gamma2[i_band][i_mix][y] 32 float gamma3[i_band][i_mix][x] 32 float gamma3[i_band][i_mix][y] 32 float } } } } </pre>		

CONTINUED ON NEXT PAGE

FIG. 16

CONTINUED FROM PREVIOUS PAGE

<pre> for (i_bin=0; i_bin < bin_count; i_bin++){ scale_factor[i_bin] offset[i_bin] } //bin mode table_index int db_resolution for (i_bin = 0; i_bin < bin_count; i_bin++){ for (i_point=0; i_point < num_point){ diff_data[i_bin][i_point] } } </pre>			
	// DYNAMIC RANGE OF DIRECTIVITY DATA IN EACH bin	32	float
	// DIRECTIVITY DATA MINIMUM VALUE IN EACH bin	32	float
	// HYBRID MODE ID USED FOR ENCODING	8	Unsigned char
	// table index USED FOR ENCODING	8	Unsigned char
	// DESIGNATE QUANTIZATION step SIZE SUCH AS QUANTIZATION FOR EACH 1.0dB		
	※EXAMPLE 0: NO QUANTIZATION, 1: 0.01dB, 2:0.2dB, 3:0.4dB, ... 2^8=256:25.6dB	8	Unsigned char
	// huffman CODE WITH DIFFERENCE INFORMATION ENCODED	1 TO nbits_res_data	

FIG. 17

diff_mode_count	// TOTAL NUMBER OF DIFFERENCE ENCODING METHODS (NO DIFFERENCE ENCODING, SPATIAL DIFFERENCE, DIFFERENCE BETWEEN bins, etc.)	8	Unsigned char
for (i_mode = 0; i_mode < diff_mode_count; i_mode++)			
int nbits_res_data	// MAXIMUM bit NUMBER OF huffman code, 7bits FOR 1.0dB, 0-128dB Huffman CODE MAXIMUM WORD LENGTH	32	Int
element_count //NUMBER OF ELEMENTS OF Huffman table		32	int
for (i_element = 0; i_element < element_count; i_element++)			
Huff_dec_table[i_element]	Huffman code -> DATA	32	int
}			
}			

17/40
FIG. 18

```

const unsigned char g_a_res_model_dectbl[NCODES_RES_MODEL_DECTBL] = {
100, 100, 100, 100, 100, 100, 100, 100, 2, 2, 4, 4, 12, 12, 12, 12,
36, 36, 36, 36, 36, 36, 36, 36, 16, 16, 16, 16, 108, 108, 108, 108,
50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81,
21, 21, 21, 21, 22, 22, 22, 22, 20, 20, 20, 20, 110, 110, 110, 110,
99, 99, 99, 99, 99, 99, 99, 99, 15, 15, 15, 15, 112, 112, 112, 112,
13, 13, 13, 13, 111, 111, 111, 111, 37, 37, 37, 37, 37, 37, 37, 37,
78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78,
52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
38, 38, 38, 38, 38, 38, 38, 38, 109, 109, 109, 109, 6, 6, 8, 8,
80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79, 79,
58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58,
77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77, 77,
56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56,
98, 98, 98, 98, 98, 98, 98, 98, 25, 25, 25, 25, 107, 107, 107, 107,
54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54,
61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61,
97, 97, 97, 97, 97, 97, 97, 97, 39, 39, 39, 39, 39, 39, 39, 39,
59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59,
57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57,
60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55,
74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74,
62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63,
64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76, 76,
40, 40, 40, 40, 40, 40, 40, 40, 125, 125, 5, 5, 7, 7, 127, 127,
65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65,
75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75,
27, 27, 27, 27, 24, 24, 24, 24, 28, 28, 28, 28, 106, 106, 106, 106,
73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 73, 73,
66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66, 66,
69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69,
70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70,
68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68,
95, 95, 95, 95, 95, 95, 95, 95, 96, 96, 96, 96, 96, 96, 96, 96,
67, 67, 67, 67, 67, 67, 67, 67, 67, 67, 67, 67, 67, 67, 67, 67,
72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72,
71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71, 71,
26, 26, 26, 26, 29, 29, 29, 29, 41, 41, 41, 41, 41, 41, 41, 41,
93, 93, 93, 93, 93, 93, 93, 93, 124, 124, 126, 126, 123, 123, 122, 122,
42, 42, 42, 42, 42, 42, 42, 42, 92, 92, 92, 92, 92, 92, 92, 92,
91, 91, 91, 91, 91, 91, 91, 91, 43, 43, 43, 43, 43, 43, 43, 43,
105, 105, 105, 105, 9, 9, 121, 121, 30, 30, 30, 30, 1, 3, 120, 120,
94, 94, 94, 94, 94, 94, 94, 94, 32, 32, 32, 32, 104, 104, 104, 104,
45, 45, 45, 45, 45, 45, 45, 45, 31, 31, 31, 31, 10, 10, 11, 11,
90, 90, 90, 90, 90, 90, 90, 90, 102, 102, 102, 102, 116, 116, 119, 119,
44, 44, 44, 44, 44, 44, 44, 44, 89, 89, 89, 89, 89, 89, 89, 89,
46, 46, 46, 46, 46, 46, 46, 46, 34, 34, 34, 34, 103, 103, 103, 103,
47, 47, 47, 47, 47, 47, 47, 47, 33, 33, 33, 33, 118, 118, 18, 18,
88, 88, 88, 88, 88, 88, 88, 88, 114, 114, 115, 115, 35, 35, 35, 35,
85, 85, 85, 85, 85, 85, 85, 85, 86, 86, 86, 86, 86, 86, 86, 86,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
48, 48, 48, 48, 48, 48, 48, 48, 49, 49, 49, 49, 49, 49, 49, 49,
101, 101, 101, 101, 23, 23, 113, 113, 84, 84, 84, 84, 84, 84, 84, 84,
87, 87, 87, 87, 87, 87, 87, 87, 117, 117, 14, 14, 17, 17, 19, 19,
83, 83, 83, 83, 83, 83, 83, 83, 82, 82, 82, 82, 82, 82, 82, 82
};

```

FIG. 19

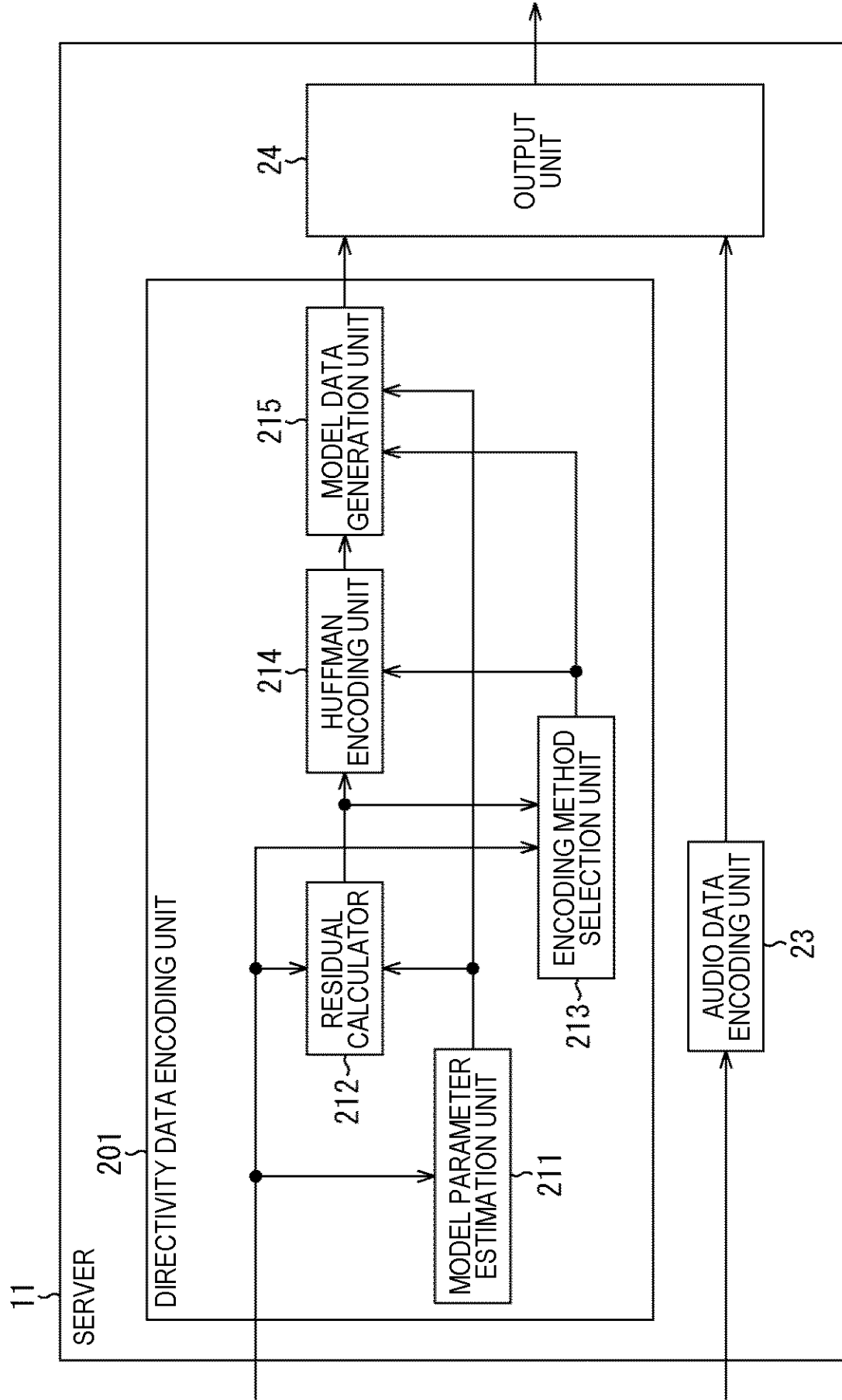


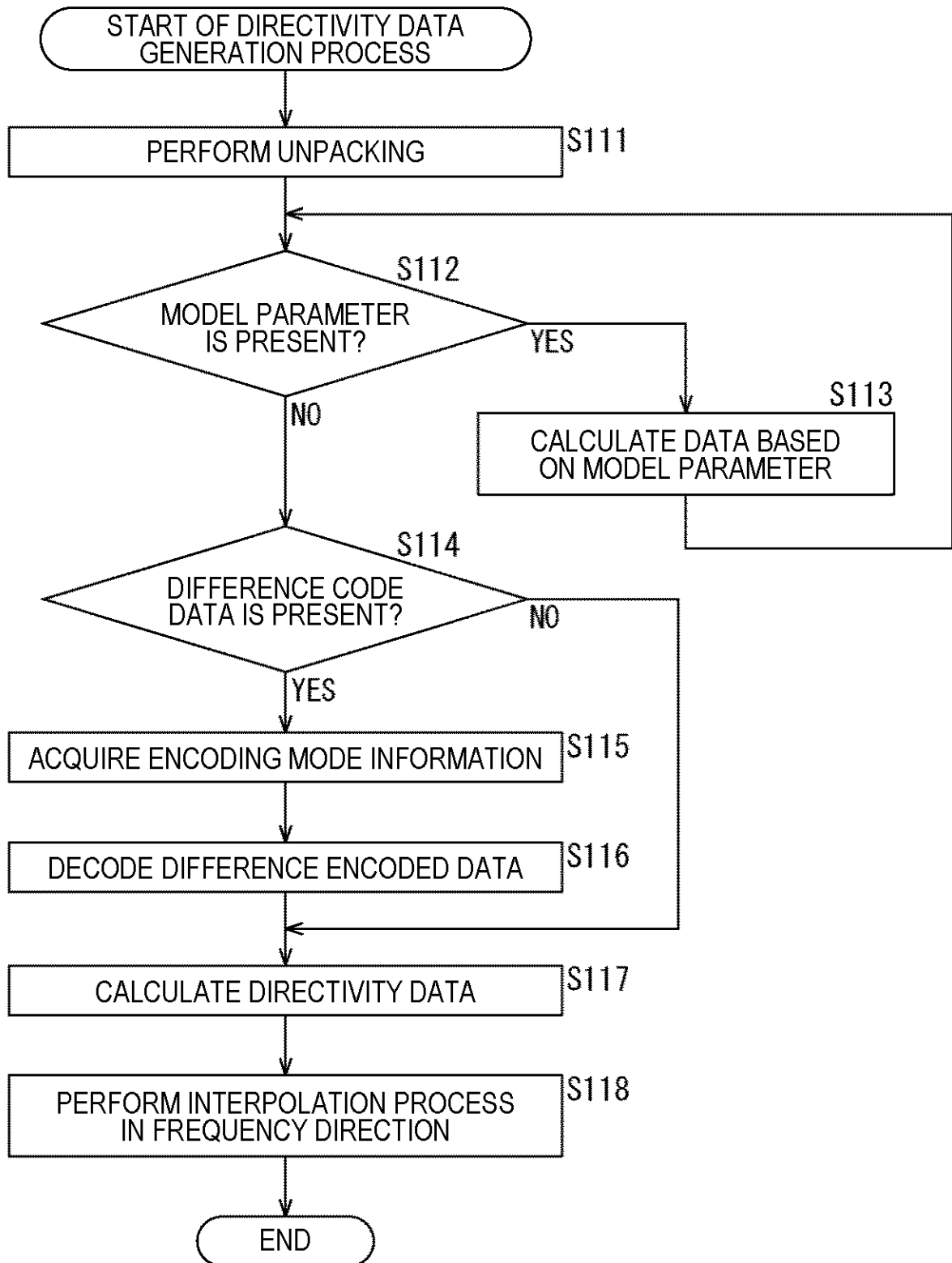
FIG. 20

FIG. 21

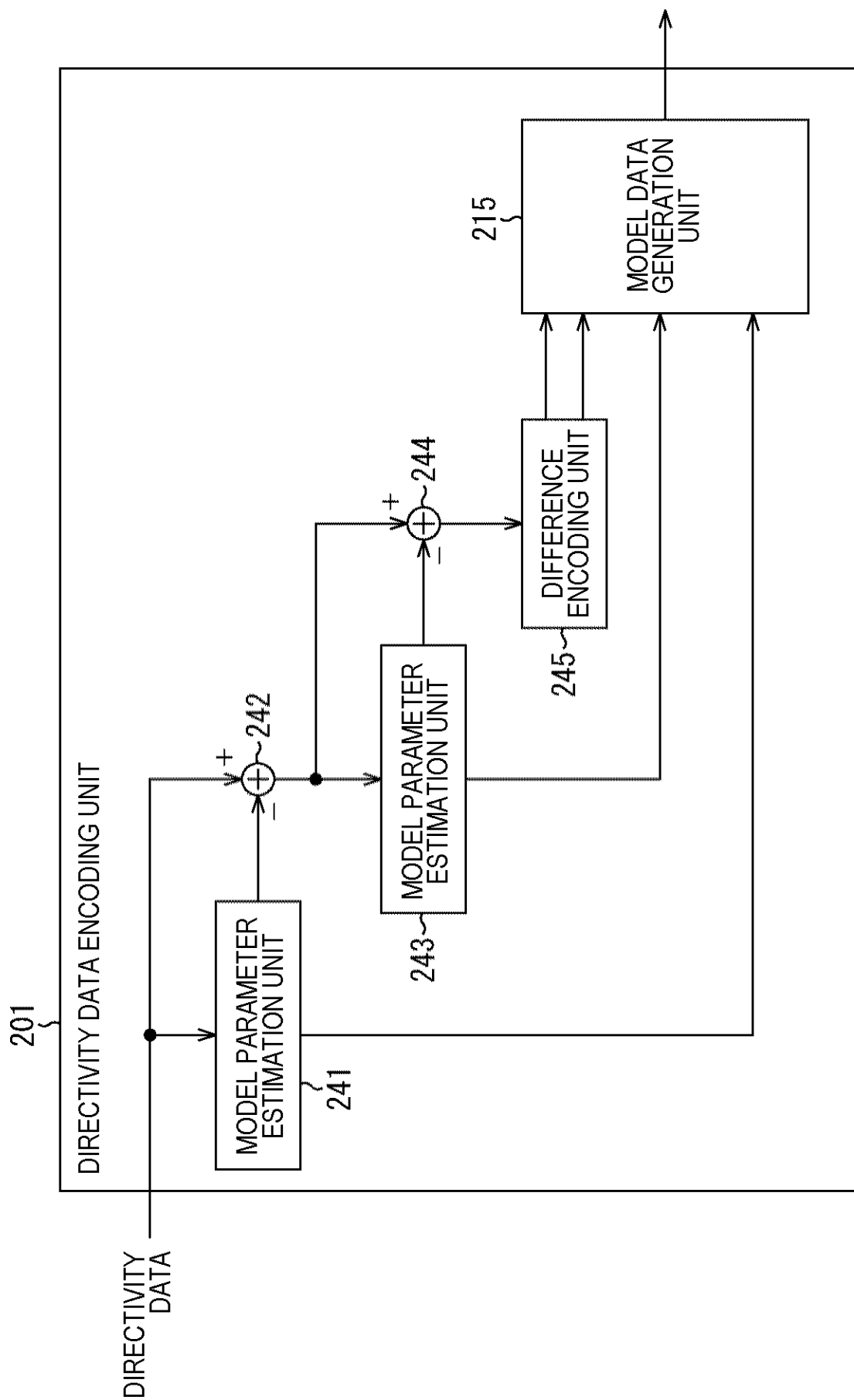


FIG. 22

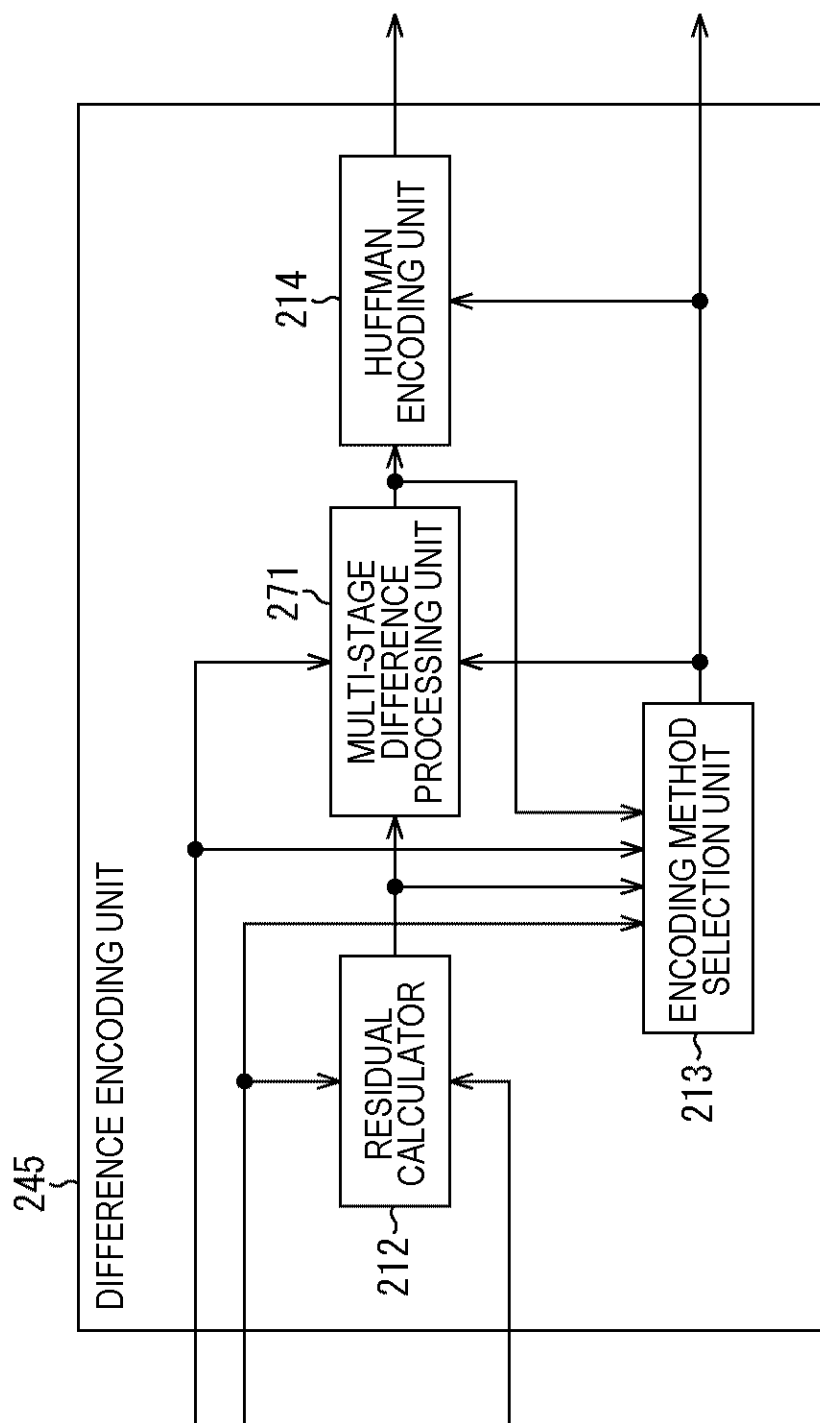


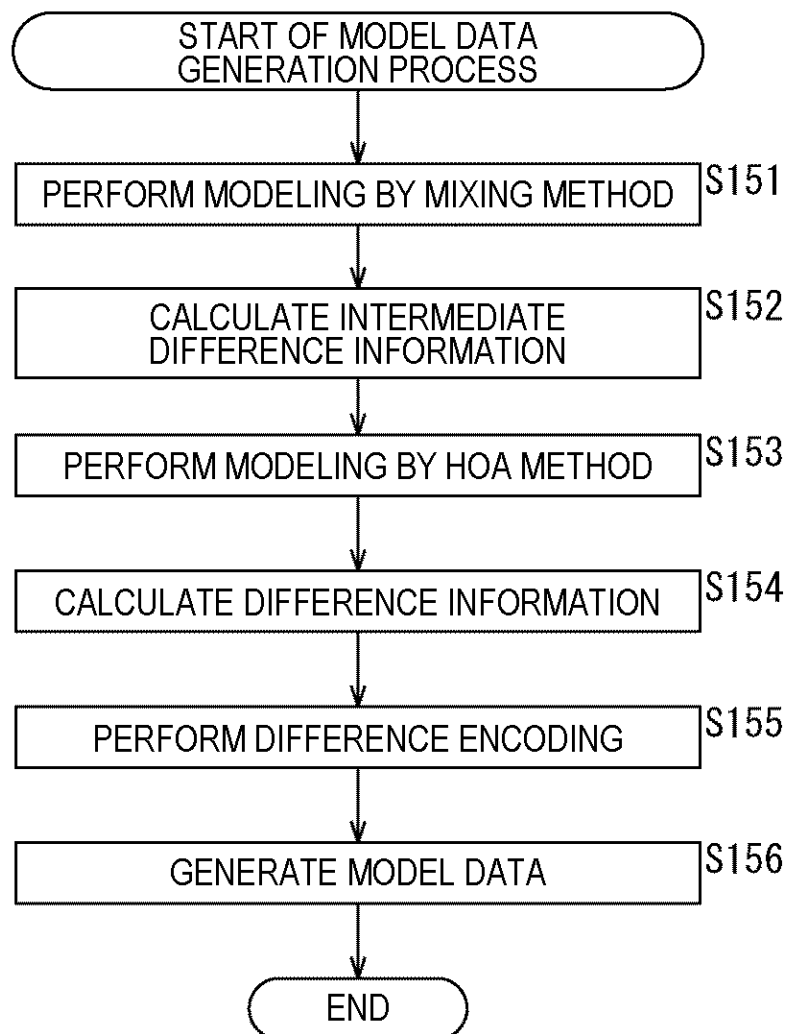
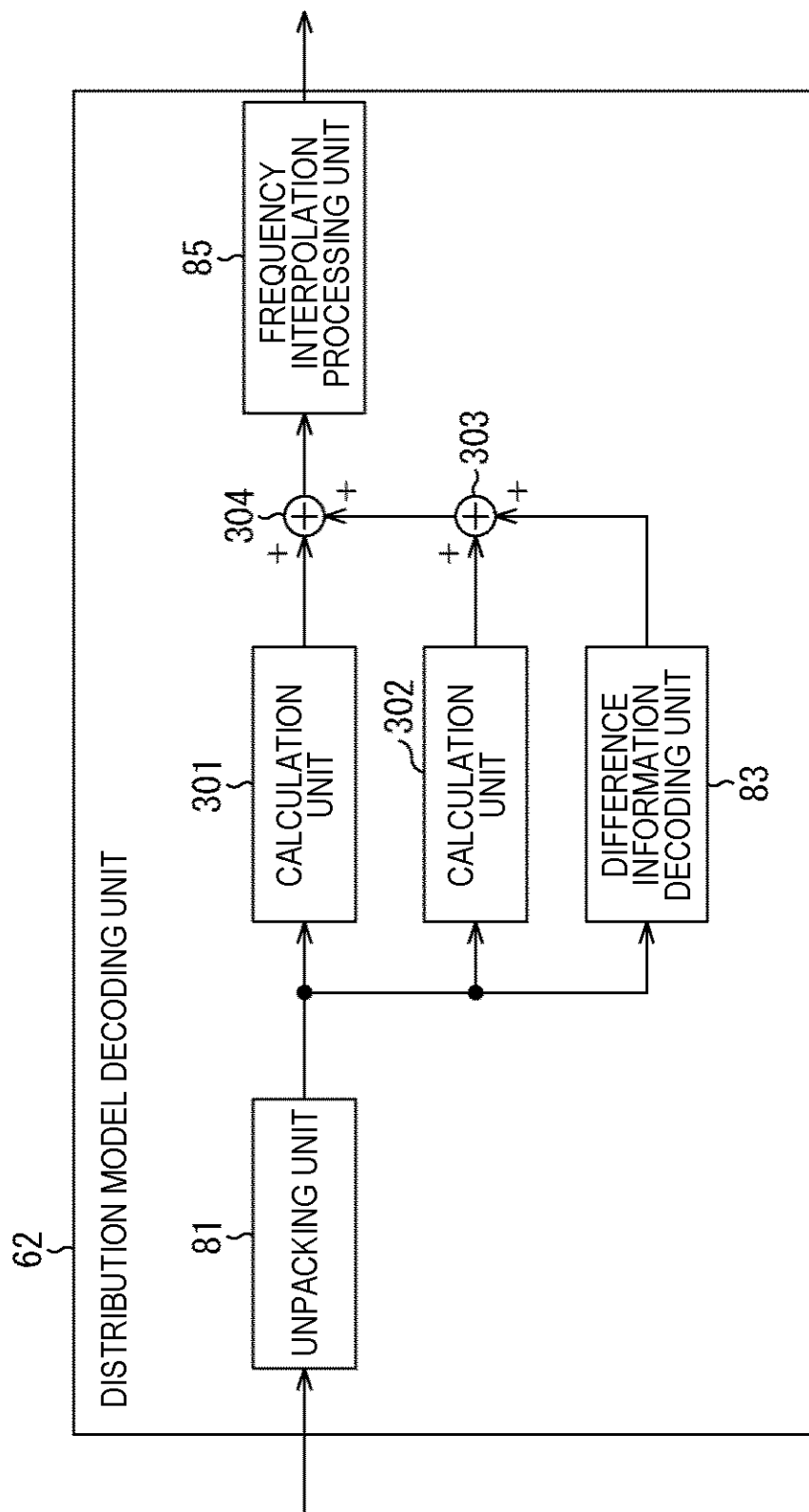
FIG. 23

FIG. 24



24/40
FIG. 25

Syntax	Mnemonic	Modified No. of bits
DirectivityReproduction() {		
bin_count;	uimsbf	12
for (i=0; i<bin_count; i++) {		
bin_freq[i];	bslbf	18
}		
band_count;	uimsbf	12
for (j=0; j<band_count; j++) {		
mix_count[j];	uimsbf	8
bin_range_per_band[j];	uimsbf	12
for (k=0; k<mix_count[j]; k++) {		
kappa[j][k];	uimsbf	11
weight[j][k];	uimsbf	10
gamma_x[j][k];	uimsbf	9
gamma_y[j][k];	uimsbf	9
gamma_z[j][k];	uimsbf	9
dist_flag	bool	1
if (dist_flag == 1) {		
beta[j][k];	uimsbf	11
gamma2_x[j][k];	uimsbf	9
gamma2_y[j][k];	uimsbf	9
gamma2_z[j][k];	uimsbf	9
gamma3_x[j][k];	uimsbf	9
gamma3_y[j][k];	uimsbf	9
gamma3_z[j][k];	uimsbf	9
} //flag		
} //mix_count		
} //band_count		
for (i=0; i<bin_count; i++) {		
scale_factor[i];	uimsbf	7
offset[i];	uimsbf	7
}		
}		

FIG. 26

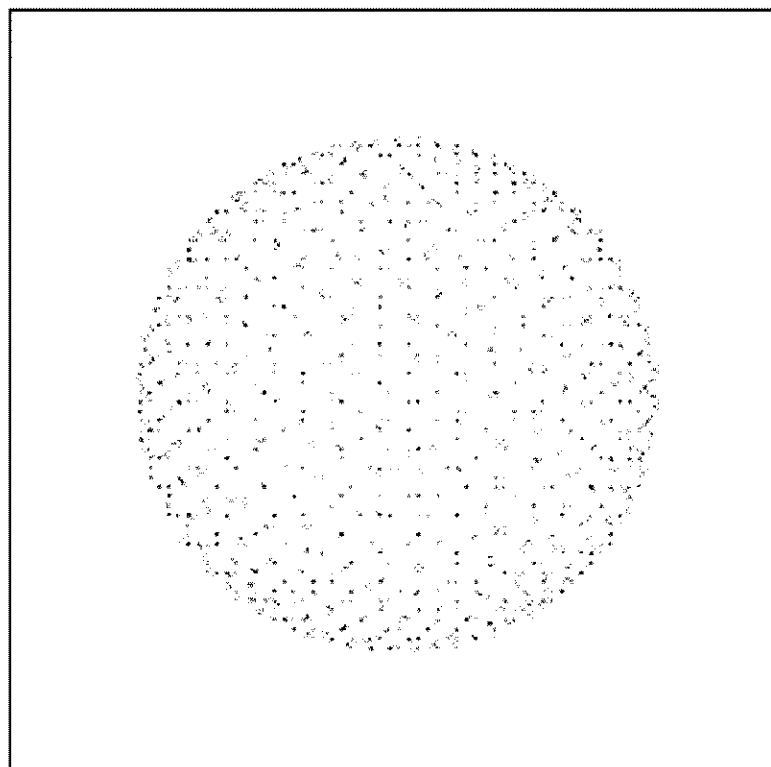


FIG. 27

Syntax	Mnemonic	No. of bits	notes
DirectivityReproductionPosInfo()			
position_type	bslbf	3	COORDINATE RECORDING METHOD
priority_index	bslbf	18	PRIORITY OF DIRECTIVITY DATA
Switch (position_type) {			
Case 0x000			
azimuth_interval	bslbf	10	INTERVAL BETWEEN AZIMUTH ANGLES OF ADJACENT DATA
elevation_interval	bslbf	9	INTERVAL BETWEEN ELEVATION ANGLES OF ADJACENT DATA
Case 0x001			
uniform_dist_point_count	bslbf	17	NUMBER OF DATA POINTS IN UNIFORM DISTRIBUTION
Case 0x010			
num_mandatory_point	bslbf	17	
for (i = 0; i < num_mandatory_point; i++) {			
azimuth_table[i]	bslbf	16	AZIMUTH ANGLE DATA
elevation_table[i]	bslbf	16	ELEVATION ANGLE DATA
}			
gain_resolution	bslbf	8	DATA POINT ARRANGEMENT RESOLUTION (dB)
}			
}			

FIG. 28

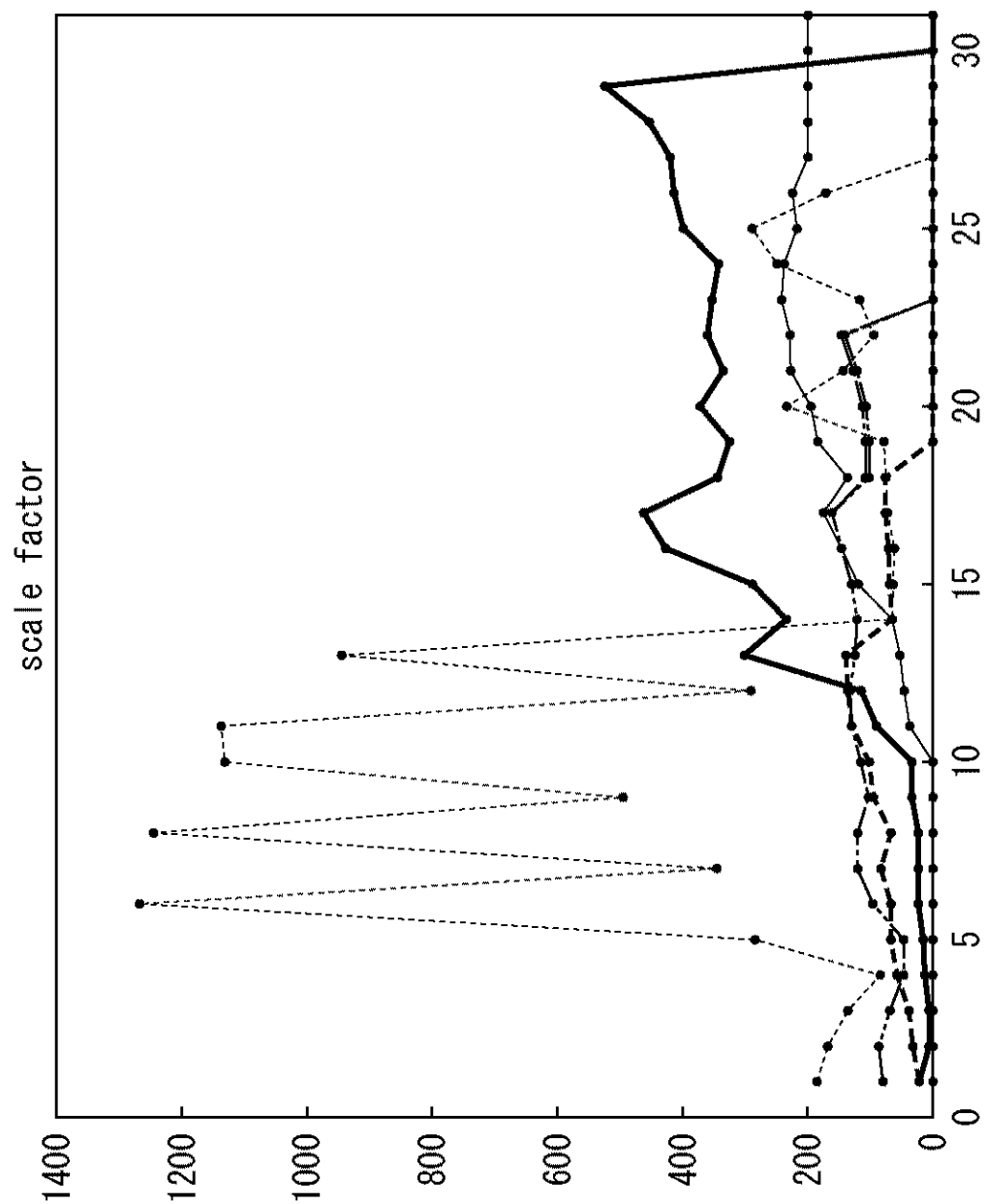


FIG. 29

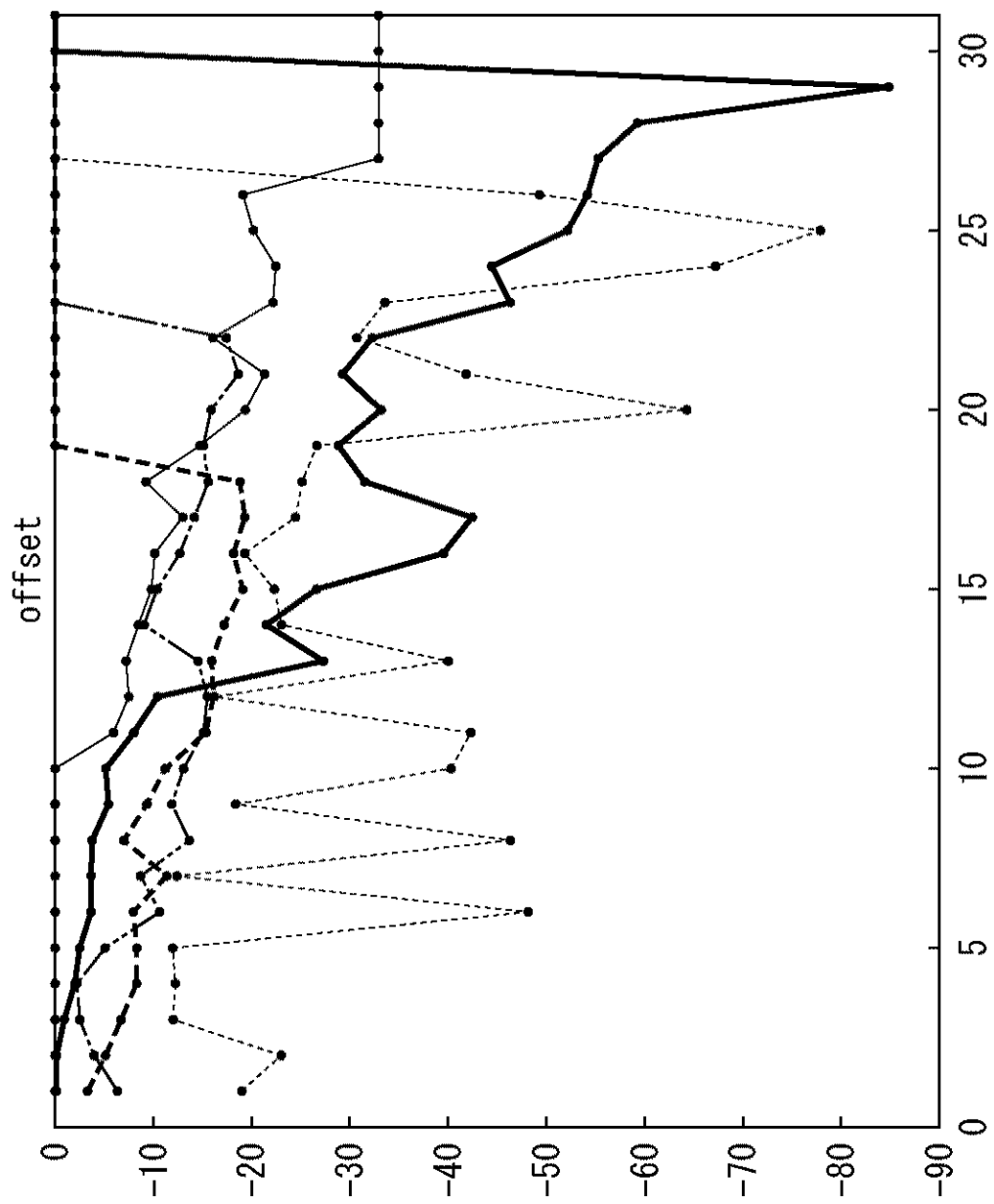


FIG. 30

Syntax	No. of bits	Mnemonic
<pre> DirectivityReproduction() { bin_count; for (i=0; i < bin_count; i++) { bin_freq[i]; } band_count; for (j=0; j < band_count; j++) { use_symmetry switch(use_symmetry) case 4: LeftRightVerticalLineSymmetricDir () // UP-AND-DOWN AND LEFT-AND-RIGHT SYMMETRY case 3: LeftRightLineSymmetricDir () // LEFT-AND-RIGHT SYMMETRY case 2: VerticalLineSymmetricDir () // UP-AND-DOWN SYMMETRY case 1: SymmetricDir () // ANY SYMMETRY, ROTATION case 0: NonSymmetricDir () // WITHOUT SYMMETRY, ROTATION (DESCRIBE ENTIRE vmf / Kent DISTRIBUTION) } } </pre>	12 32 8 3	uimsbf bslbf uimsbf uimsbf

30/40
FIG. 31

Syntax	No. of bits	Mnemonic
SymmetricDir() {		
mix_count[j];	12	uimsbf
bin_range_per_band[j];	12	uimsbf
for(k=0; k<mix_count[k]; k++) {		
kappa[j][k];	11	uimsbf
weight[j][k];	10	uimsbf
gamma_x[j][k];	9	uimsbf
gamma_y[j][k];	9	uimsbf
gamma_z[j][k];	9	uimsbf
dist_flag[j][k];	1	uimsbf
if(dist_flag == 1) {		
beta[j][k];	11	uimsbf
gamma2_x[j][k];	9	uimsbf
gamma2_y[j][k];	9	uimsbf
gamma2_z[j][k];	9	uimsbf
gamma3_x[j][k];	9	uimsbf
gamma3_y[j][k];	9	uimsbf
gamma3_z[j][k];	9	uimsbf
} //flag		
sym_operation_count;	2	uimsbf
for(j=0; j<sym_operation_count; j++) {		
sym_operation_flag;	1	uimsbf
Switch(sym_operation_flag)		
Case 1://rotation		
sym_az;	9	uimsbf
sym_elev;	9	uimsbf
sym_rotation;	9	uimsbf
Default://line symmetry		
sym_yaw;	9	uimsbf
sym_pitch;	9	uimsbf
sym_roll;	9	uimsbf
}		
}		
for(i=0; i<bin_count; i++) {		
scale_factor[i];	7	uimsbf
offset[i];	7	uimsbf
}		
}		

FIG. 32

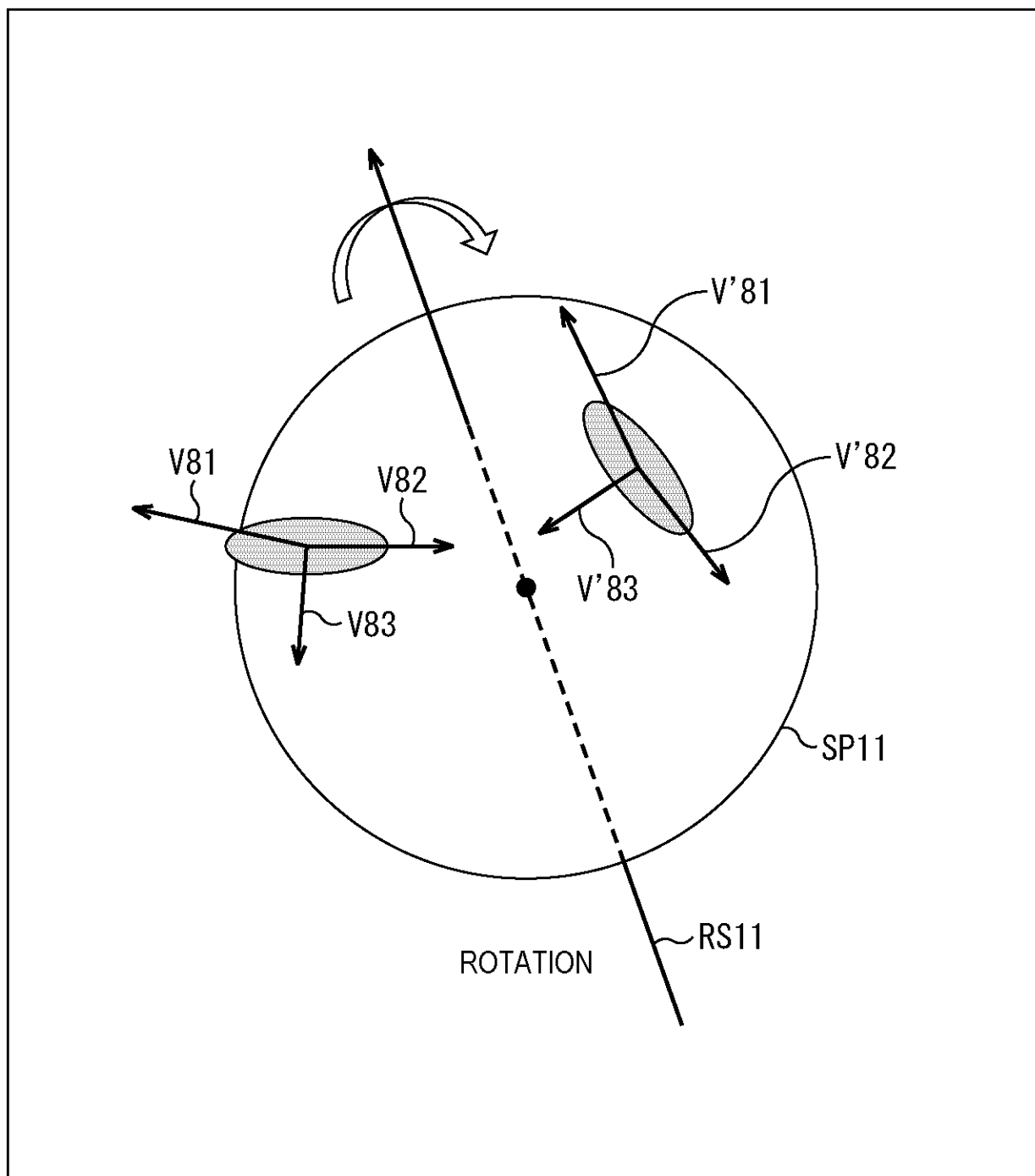


FIG. 33

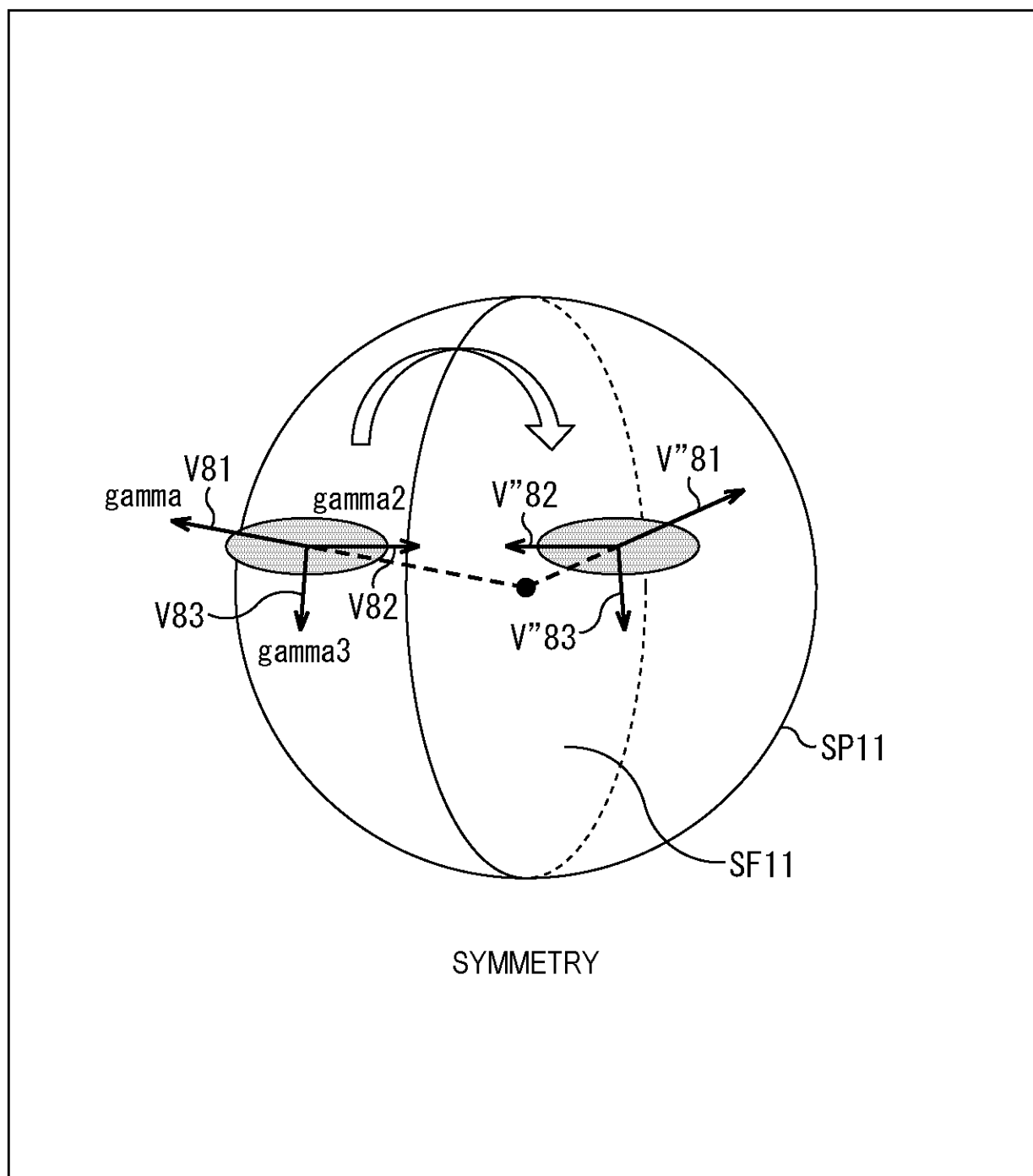


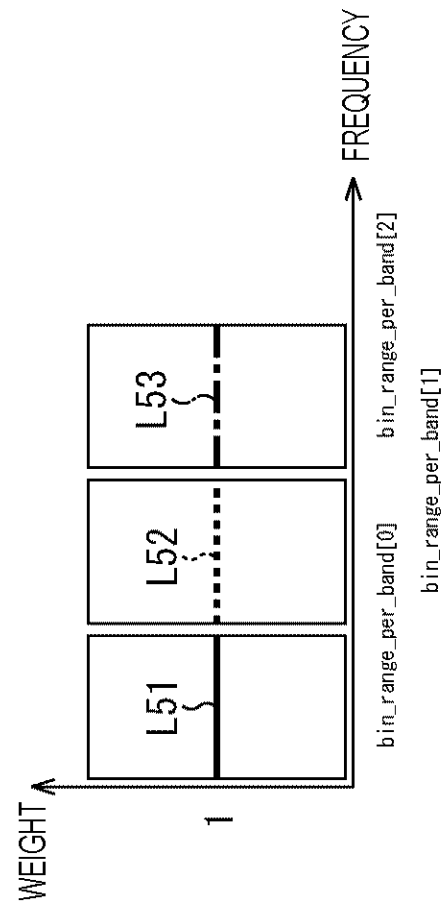
FIG. 34

Syntax	No. of bits	Mnemonic
NonSymmetricDir () {		
mix_count[j];	12	uimsbf
bin_range_per_band[j];	12	uimsbf
for (k=0; k<mix_count[k]; k++) {		
kappa[j][k];	11	uimsbf
weight[j][k];	10	uimsbf
gamma_x[j][k];	9	uimsbf
gamma_y[j][k];	9	uimsbf
gamma_z[j][k];	9	uimsbf
dist_flag[j][k];	1	uimsbf
if(dist_flag == 1) {		
beta[j][k];	11	uimsbf
gamma2_x[j][k];	9	uimsbf
gamma2_y[j][k];	9	uimsbf
gamma2_z[j][k];	9	uimsbf
gamma3_x[j][k];	9	uimsbf
gamma3_y[j][k];	9	uimsbf
gamma3_z[j][k];	9	uimsbf
} //flag		
}		
for (i=0; i<bin_count; i++) {		
scale_factor[i];	7	uimsbf
offset[i];	7	uimsbf
}		
}		

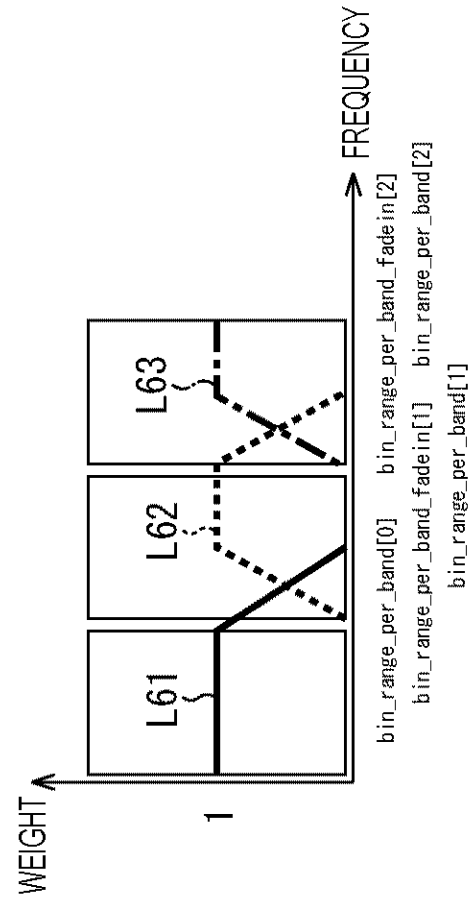
34/40
FIG. 35

Syntax	Mnemonic	Modified No. of bits
DirectivityReproduction() {		
bin_count;	uimsbf	12
for (i=0; i<bin_count; i++) {		
bin_freq[i];	bslbf	18
}		
band_count;	uimsbf	12
for (j=0; j<band_count; j++) {		
mix_count[j];	uimsbf	8
bin_range_per_band[j];	uimsbf	12
fade_flag;	uimsbf	1
if (fade_flag & j>0) {		
bin_range_per_band_fadein[j]	uimsbf	12
}		
for (k=0; k<mix_count[j]; k++) {		
kappa[j][k];	uimsbf	11
weight[j][k];	uimsbf	10
gamma_x[j][k];	uimsbf	9
gamma_y[j][k];	uimsbf	9
gamma_z[j][k];	uimsbf	9
dist_flag	bool	1
if (dist_flag == 1) {		
beta[j][k];	uimsbf	9
gamma2_x[j][k];	uimsbf	9
gamma2_y[j][k];	uimsbf	9
gamma2_z[j][k];	uimsbf	9
gamma3_x[j][k];	uimsbf	9
gamma3_y[j][k];	uimsbf	9
gamma3_z[j][k];	uimsbf	9
} //flag		
} //mix_count		
} //band_count		
for (i=0; i<bin_count; i++) {		
scale_factor[i];	uimsbf	7
offset[i];	uimsbf	7
}		
}		

FIG. 36



CASE WHERE CROSSFADE IS NOT PERFORMED (3bands)



CASE WHERE CROSSFADE IS PERFORMED (3bands)

Syntax	No. of bits	Mnemonic
DirectivityReproduction() {		
bin_count;	12	uimbsf
for (i=0; i<bin_count; i++) {		
bin_freq[i];	32	bslbf
}		
band_count;	12	uimbsf
for (j=0; j<band_count; j++) {		
use_symmetry[j]	3	uimbsf
mix_count[j];	8	uimbsf
bin_range_per_band[j];	12	uimbsf
if (j>0) {		
fade_flag;	1	uimbsf
if (fade_flag) {		
bin_range_per_band_fadein[j]	12	uimbsf
}		
} else {		
start_bin	4	uimbsf
}		
switch (use_symmetry[j])		
case 7:		
FrontBackVerticalSymmetricDir() // UP-AND-DOWN AND FRONT-AND-BACK SYMMETRY		
case 6:		
FrontBackLeftRightSymmetricDir() // FRONT-AND-BACK AND LEFT-AND-RIGHT SYMMETRY		
case 5:		
FrontBackSymmetricDir() // FRONT-AND-BACK SYMMETRY		
case 4:		
LeftRightVerticalLineSymmetricDir() // UP-AND-DOWN AND LEFT-AND-RIGHT SYMMETRY		
case 3:		
LeftRightLineSymmetricDir() // LEFT-AND-RIGHT SYMMETRY		
case 2:		
VerticalLineSymmetricDir() // UP-AND-DOWN SYMMETRY		
case 1:		
SymmetricDir() // ANY SYMMETRY, ROTATION		
case 0:		
NonSymmetricDir() // WITHOUT SYMMETRY, ROTATION (DESCRIBE ENTIRE vmf/Kent DISTRIBUTION)		
} //use_symmetry[j]		
} //j:band_count		
DynamicRangeForDir()		
}		

FIG. 40

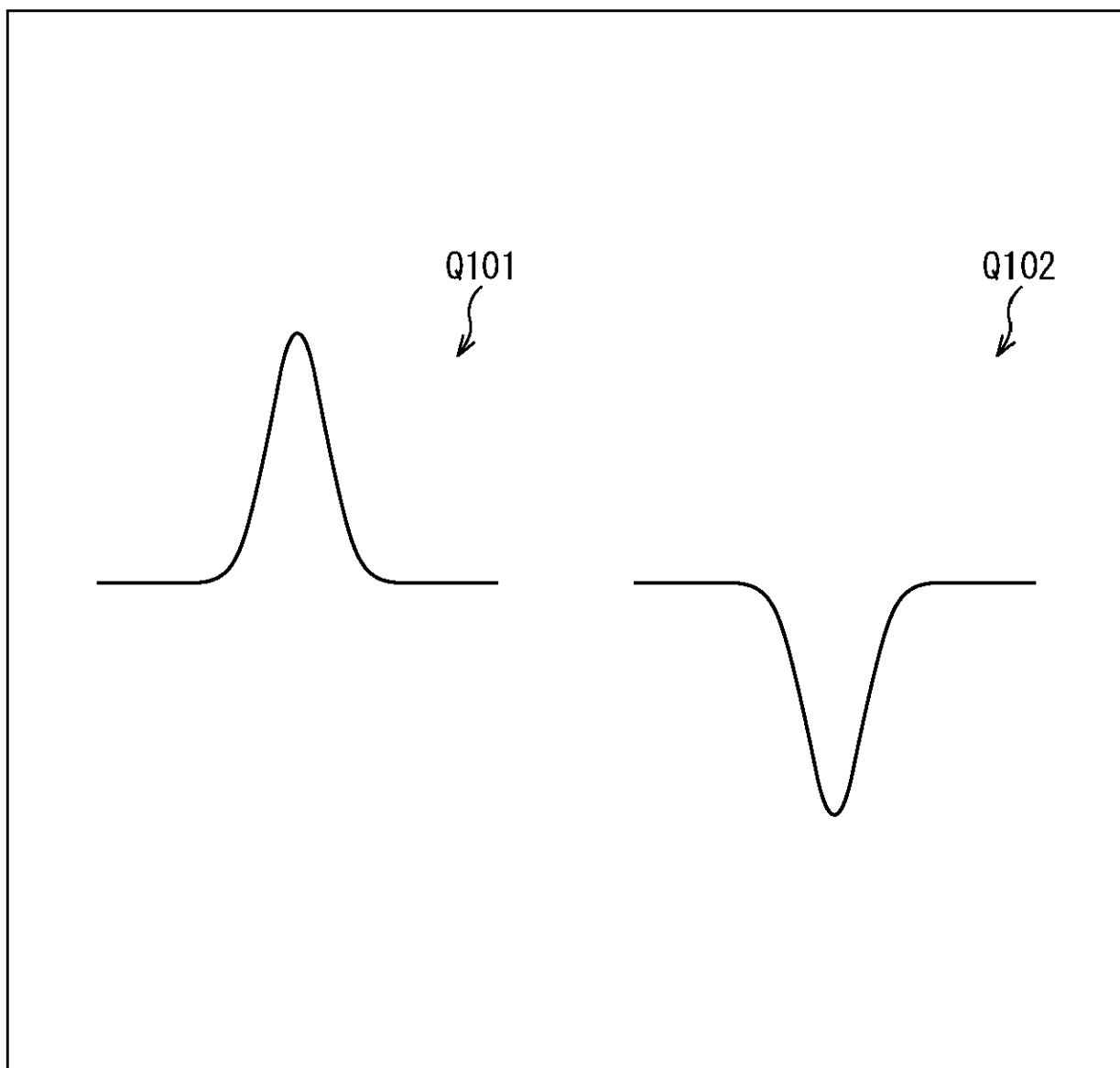


FIG. 41

