FIG. 1

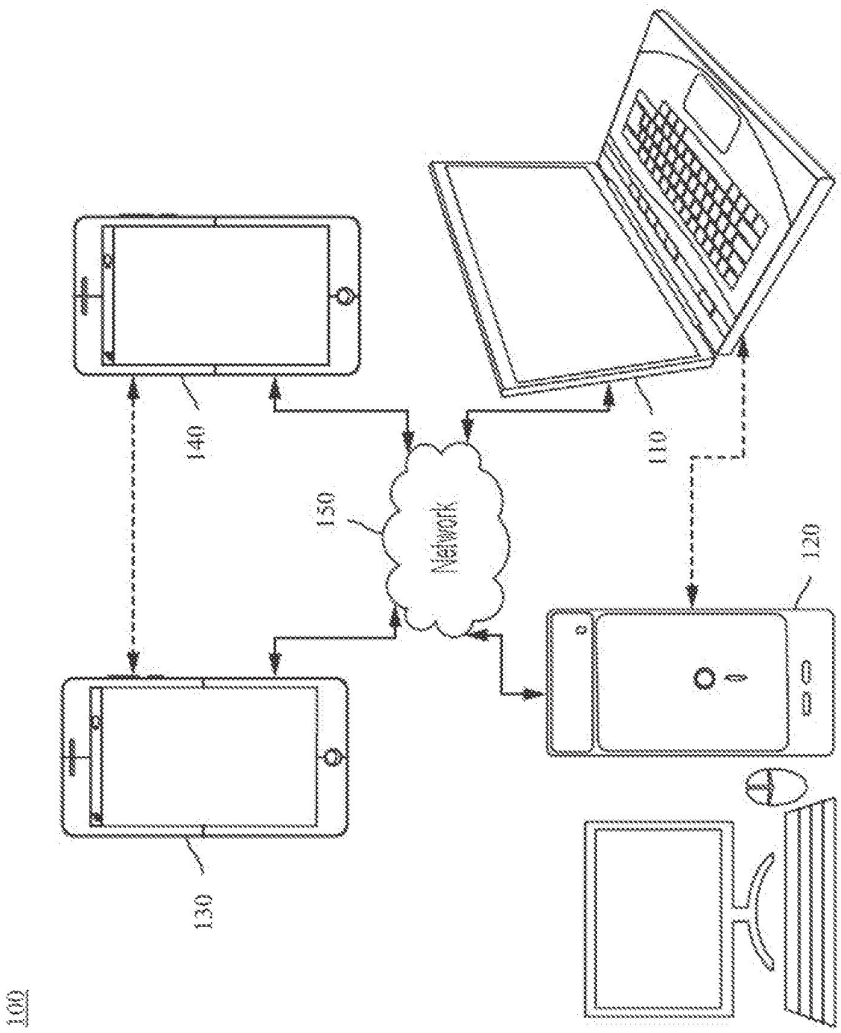


FIG. 2

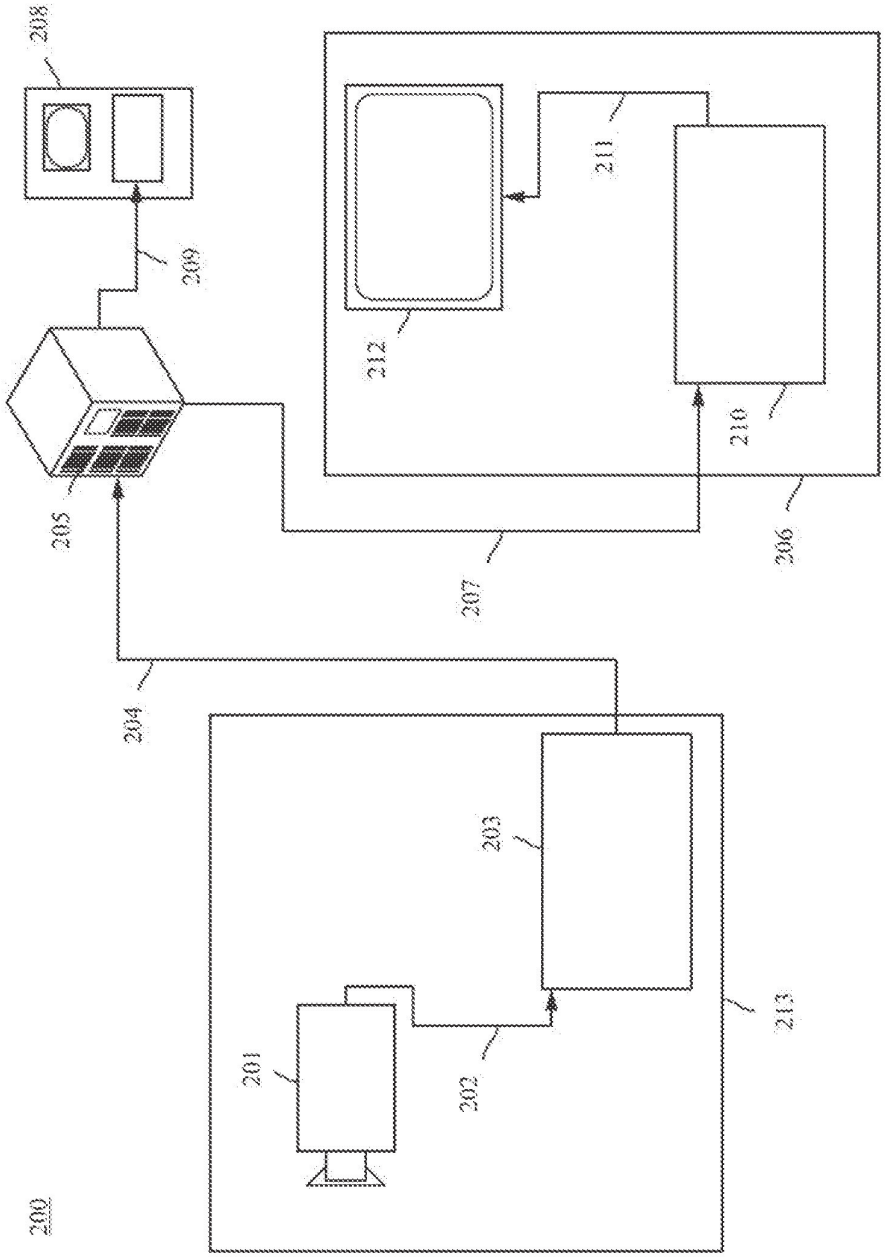


FIG. 3

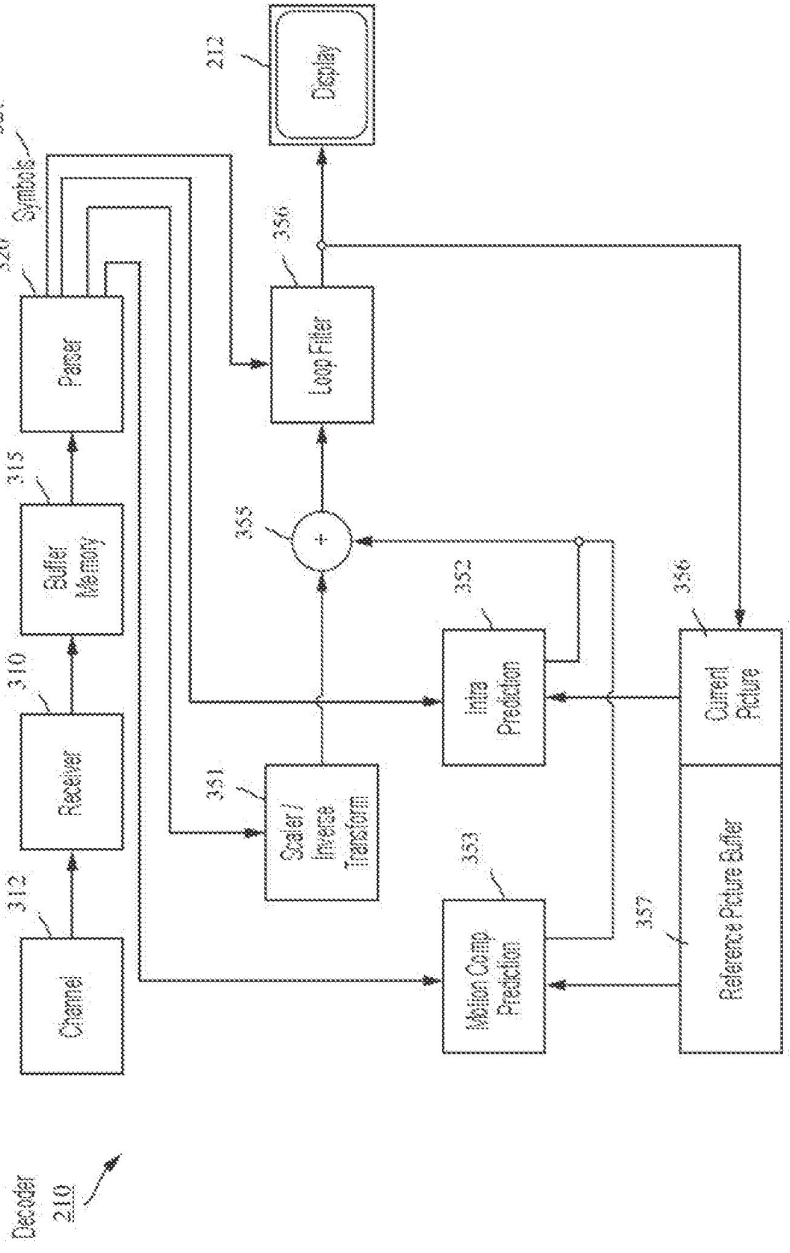

Decoder
210
Channel
312
Receiver
310
Buffer Memory
315
Parser
320
Symbols
321
Scaler / Inverse Transform
351
Intra Prediction
352
Motion Comp Prediction
353
Loop Filter
356
Display
212
Current Picture
356
Reference Picture Buffer
357
+
355

FIG. 4

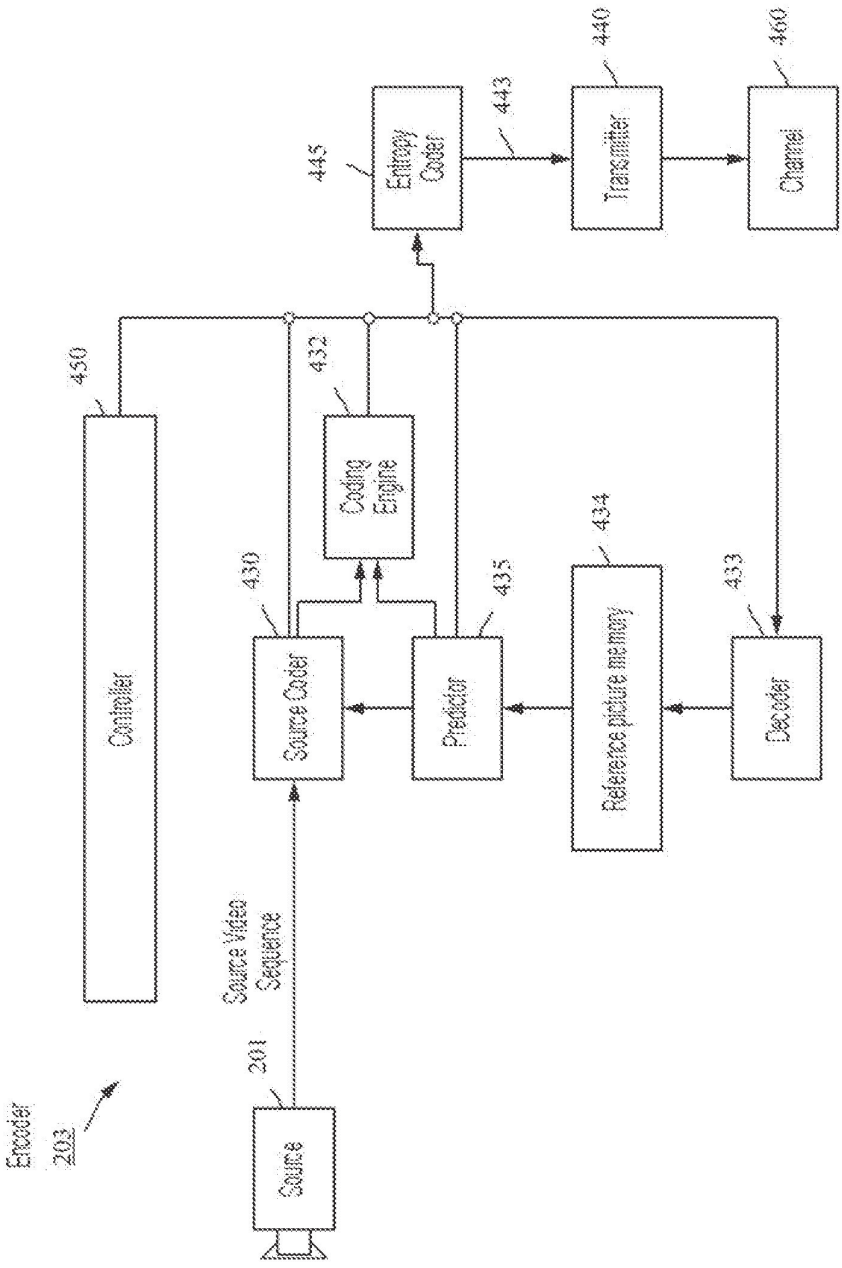

Encoder
203
Source
201
Source Video Sequence
Controller
450
Source Coder
430
Coding Engine
432
Predictor
435
Reference picture memory
434
Decoder
433
Entropy Coder
445
443
Transmitter
440
Channel
460

FIG. 5A
(related art)

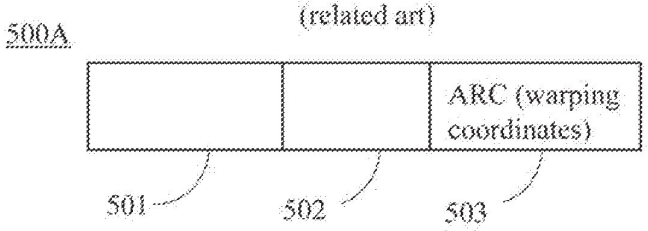


FIG. 5B
(related art)

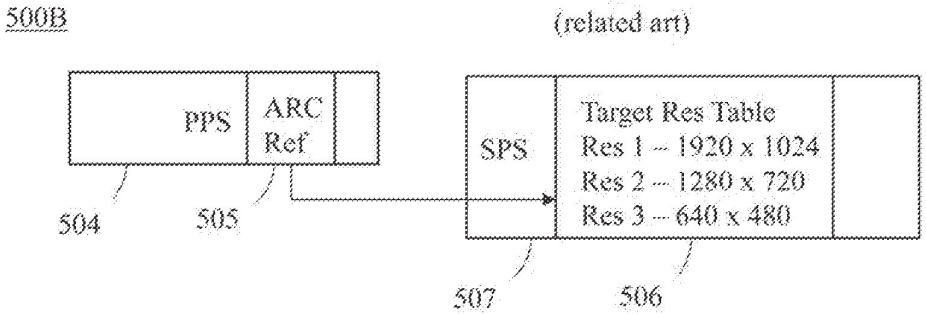


FIG. 5C

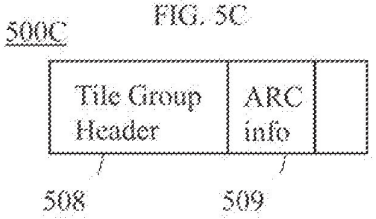


FIG. 5D

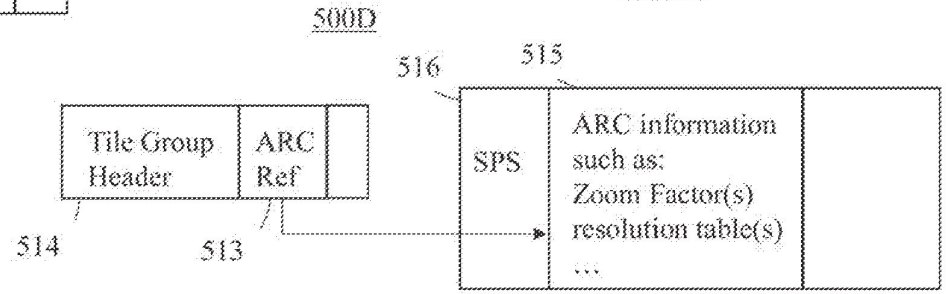
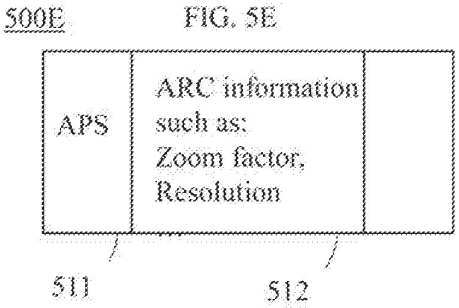


FIG. 5E



600

FIG. 6

601	tile_group_header( ){	
	...	
603	if(adaptive_pic_resolution_change_flag){	
602	**dec_pic_size_idx**	u(1)
	}	
	...	
	}	

610	seq_parameter_set_rbsp( ) {	**Descriptor**
	...	
611	**adaptive_pic_resolution_change_flag**	u(1)
612	if(adaptive_pic_resolution_change_flag) {	
613	**output_pic_width_in_luma_samples**	ue(v)
	output_pic_height_in_luma_samples	ue(v)
614	**reference_pic_size_present_flag**	u(1)
	if(reference_pic_size_present_flag)	
	{	
615	**reference_pic_width_in_luma_samples**	ue(v)
	reference_pic_height_in_luma_samples	ue(v)
	}	
616	**num_dec_pic_size_in_luma_samples_minus1**	ue(v)
	for(i = 0; i <= num_dec_pic_size_in_luma_samples_minus1; i++){	
617	**dec_pic_width_in_luma_samples[i]**	ue(v)
	dec_pic_height_in_luma_samples[i]	ue(v)
	}	
	}	
	...	
	}	

FIG. 7

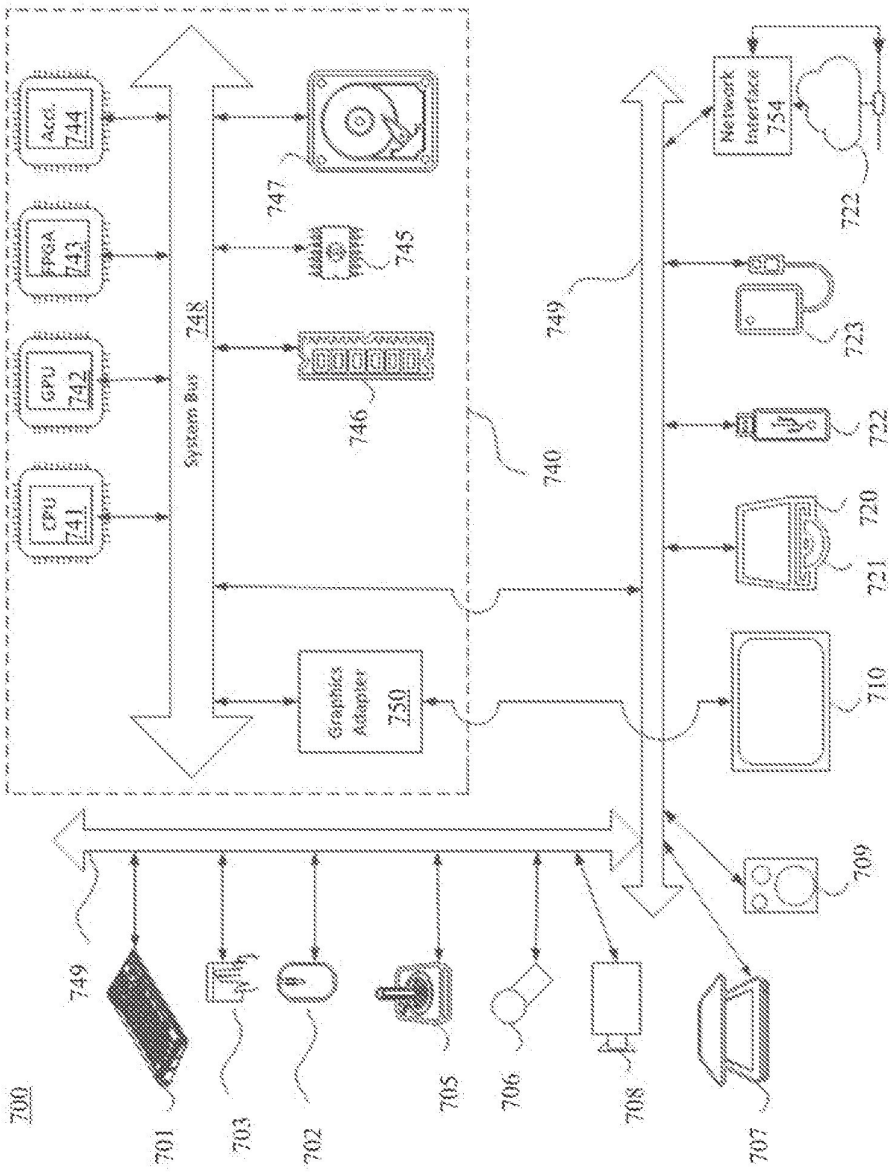

700
701
702
703
705
706
707
708
709
710
720
721
722
723
740
745
746
747
748
749
CPU 741
GPU 742
FPGA 743
Acc. 744
System Bus
Graphics Adapter 750
Network Interface 754

FIG. 8

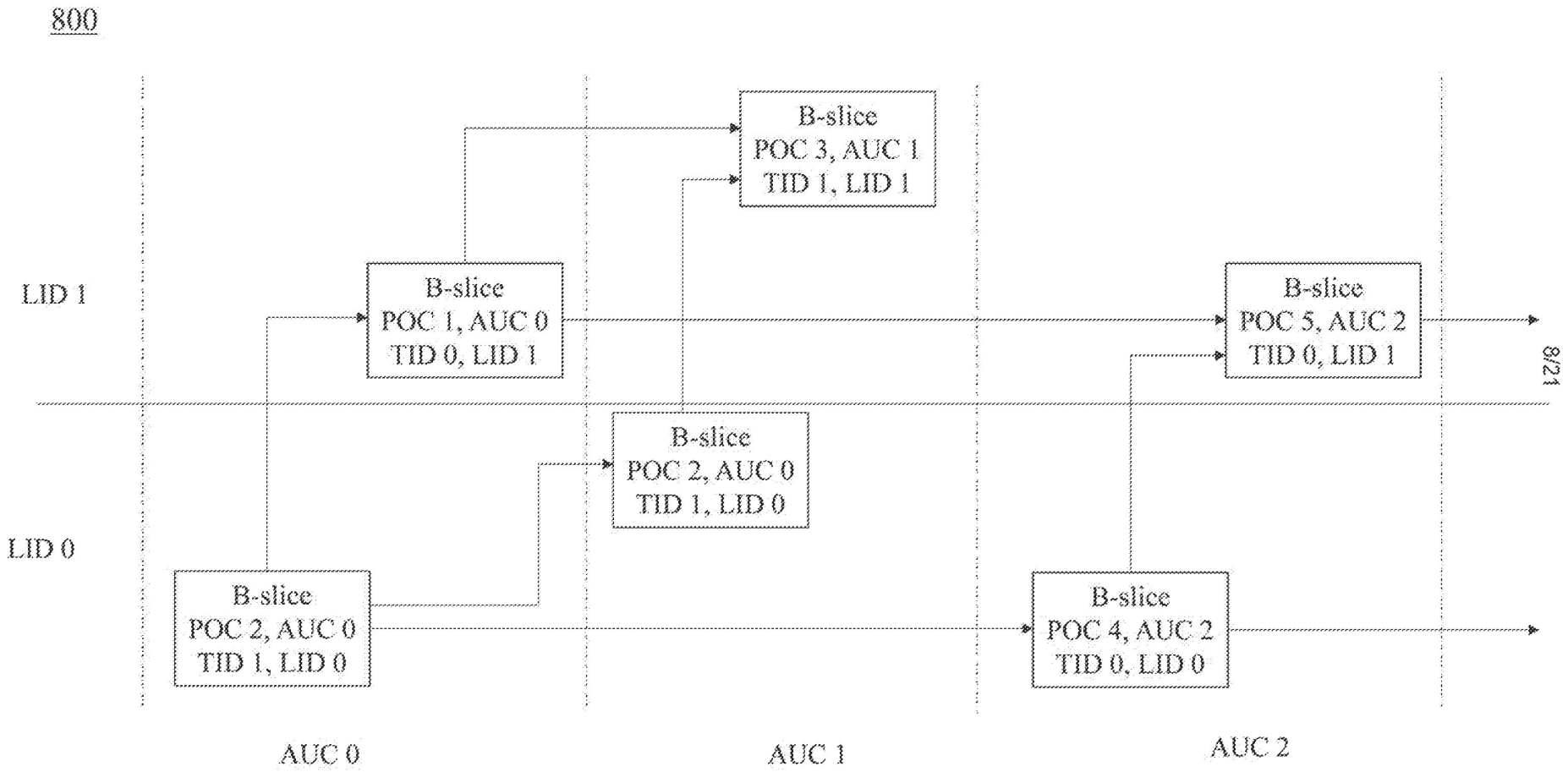

800
LID 1
LID 0
B-slice
POC 3, AUC 1
TID 1, LID 1
B-slice
POC 1, AUC 0
TID 0, LID 1
B-slice
POC 5, AUC 2
TID 0, LID 1
B-slice
POC 2, AUC 0
TID 1, LID 0
B-slice
POC 2, AUC 0
TID 1, LID 0
B-slice
POC 4, AUC 2
TID 0, LID 0
AUC 0
AUC 1
AUC 2
8/21

Fig. 9

900

video_parameter_set rbsp( ) {	Descriptor
vps_video_parameter_set_id	u(4)
vps_max_layers_minus1	u(8)
for(i = 0; i <= vps_max_layers_minus1; i++) {	
vps_included_layer_id [i]	u(7)
vps_reserved_zero_bit	u(1)
}	
vps_constraint_info_present_flag	u(1)
vps_constant_poc_cycle_per_au	u(1)
if(vps_constant_poc_cycle_per_au)	
vps_poc_cycle_au	u(8)
…	
}	
slice_header( ) {	**Descriptor**
slice_pic_parameter_set_id	ue(v)
if(rect_slice_flag || NumBricksInPic > 1)	
slice_address	u(v)
if (!rect_slice_flag && !single_brick_per_slice_flag)	
num_bricks_in_slice_minus1	ue(v)
slice_type	ue(v)
if(NalUnitType == GRA_NUT)	
recovery_poc_cnt	se(v)
slice_pic_order_cnt_lsb	u(v)
…	
if(!vps_constant_poc_cycle_per_au)	
slice_poc_cycle_au	u(8)
}	

Fig. 10

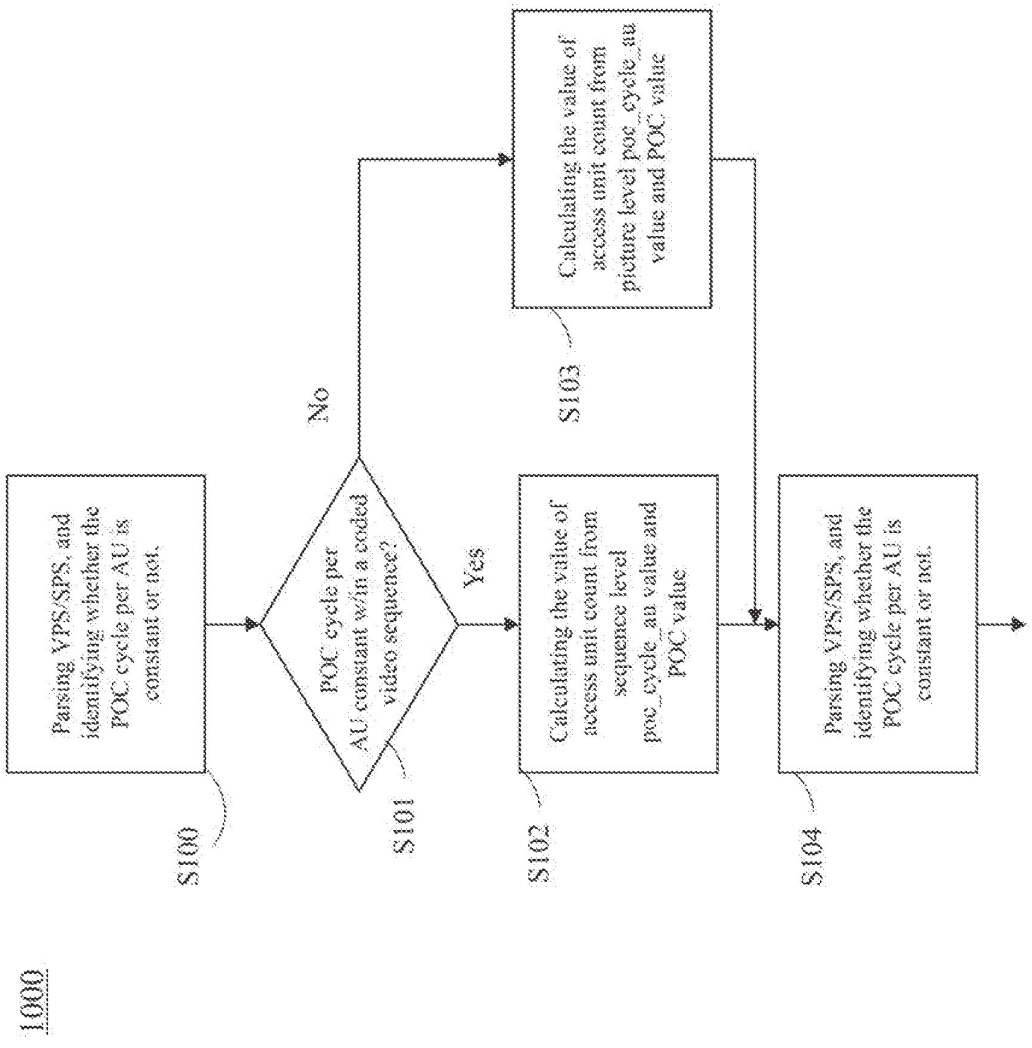

1000
Parsing VPS/SPS, and identifying whether the POC cycle per AU is constant or not.
S100
POC cycle per AU constant w/in a coded video sequence?
S101
No
Yes
Calculating the value of access unit count from picture level poc_cycle_au value and POC value
S103
Calculating the value of access unit count from sequence level poc_cycle_au value and POC value
S102
Parsing VPS/SPS, and identifying whether the POC cycle per AU is constant or not.
S104

Fig.11

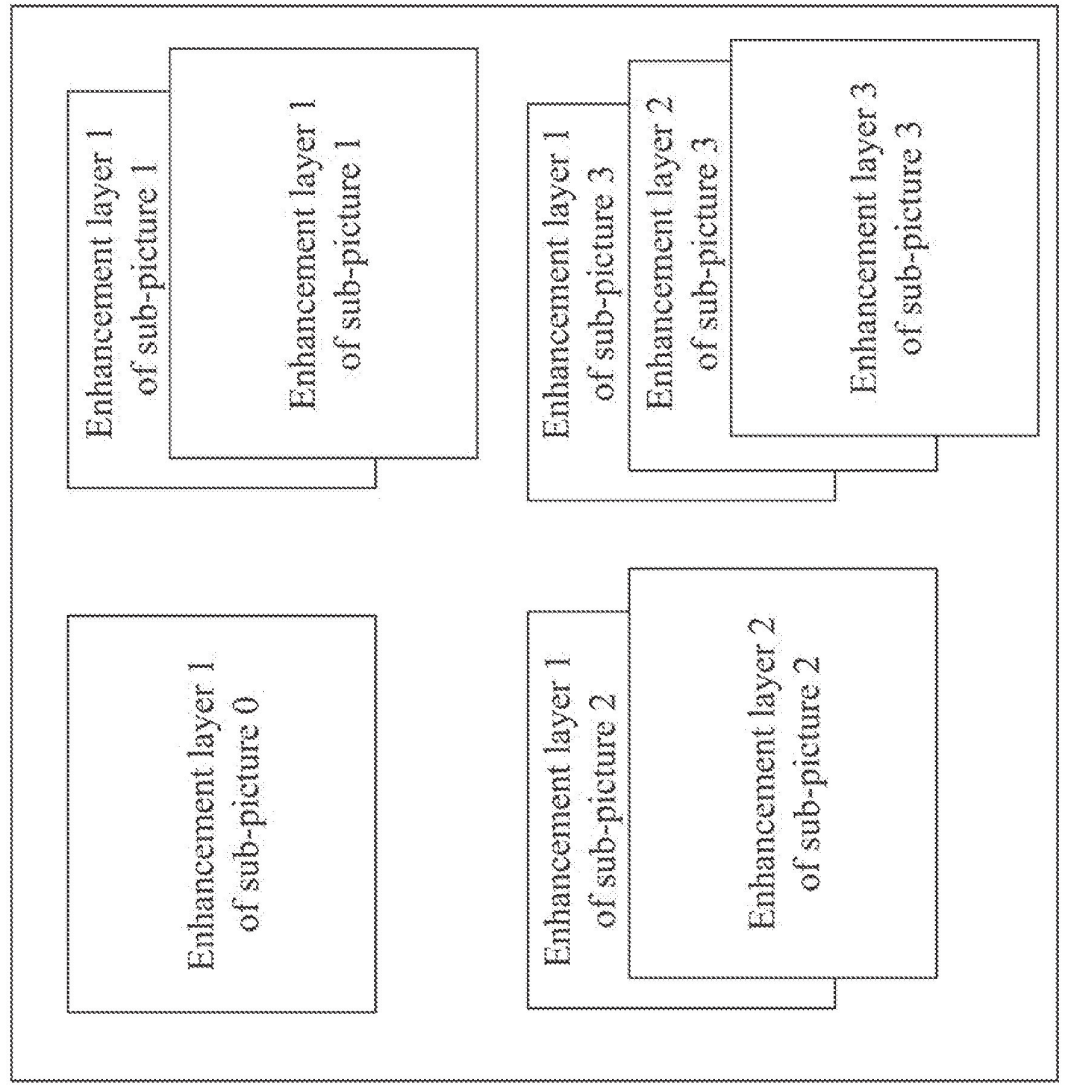

1100
Enhancement layer 1 of sub-picture 0
Enhancement layer 1 of sub-picture 1
Enhancement layer 1 of sub-picture 1
Enhancement layer 1 of sub-picture 2
Enhancement layer 2 of sub-picture 2
Enhancement layer 1 of sub-picture 3
Enhancement layer 2 of sub-picture 3
Enhancement layer 3 of sub-picture 3

FIG. 12

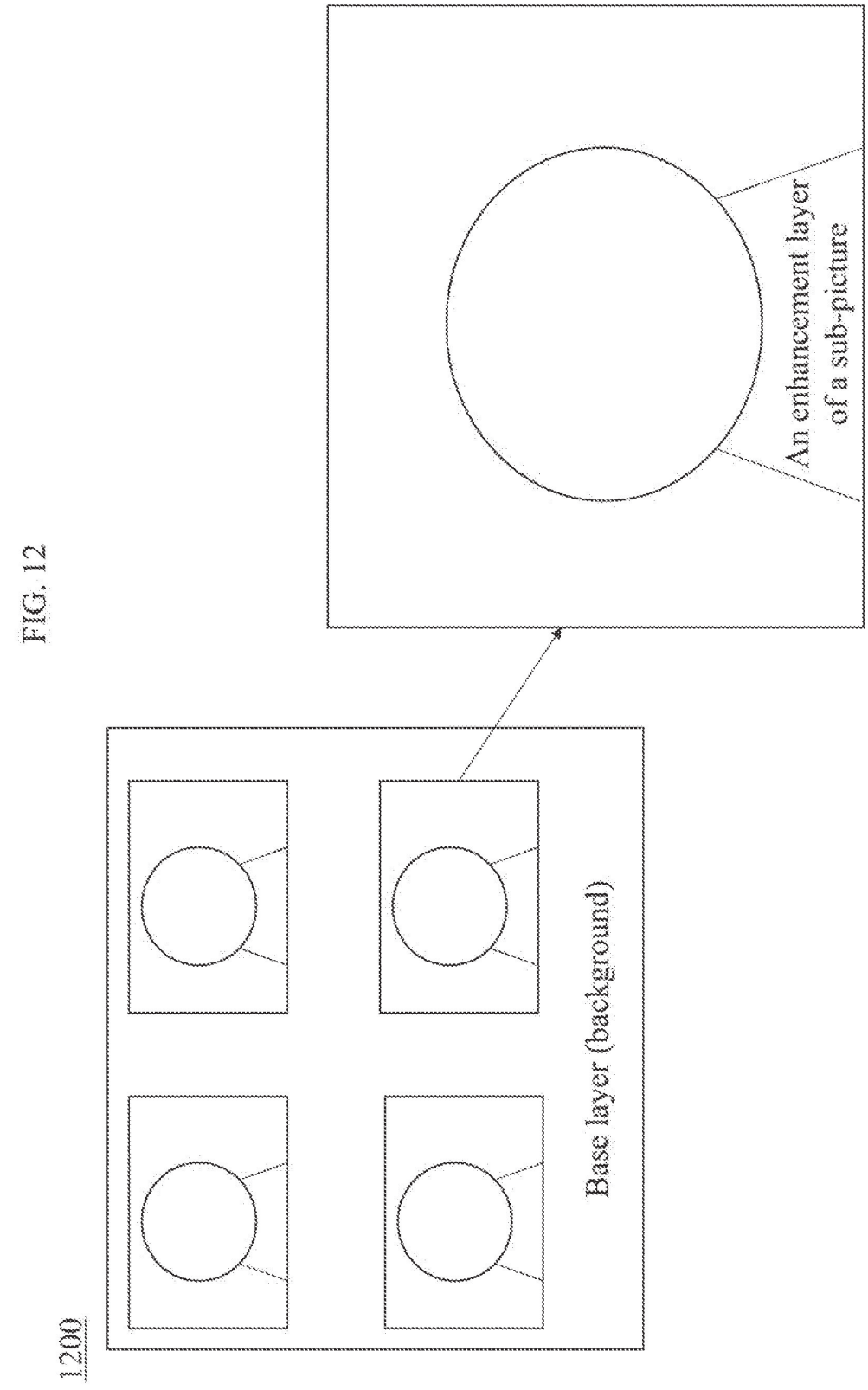

1200
Base layer (background)
An enhancement layer of a sub-picture

FIG. 13

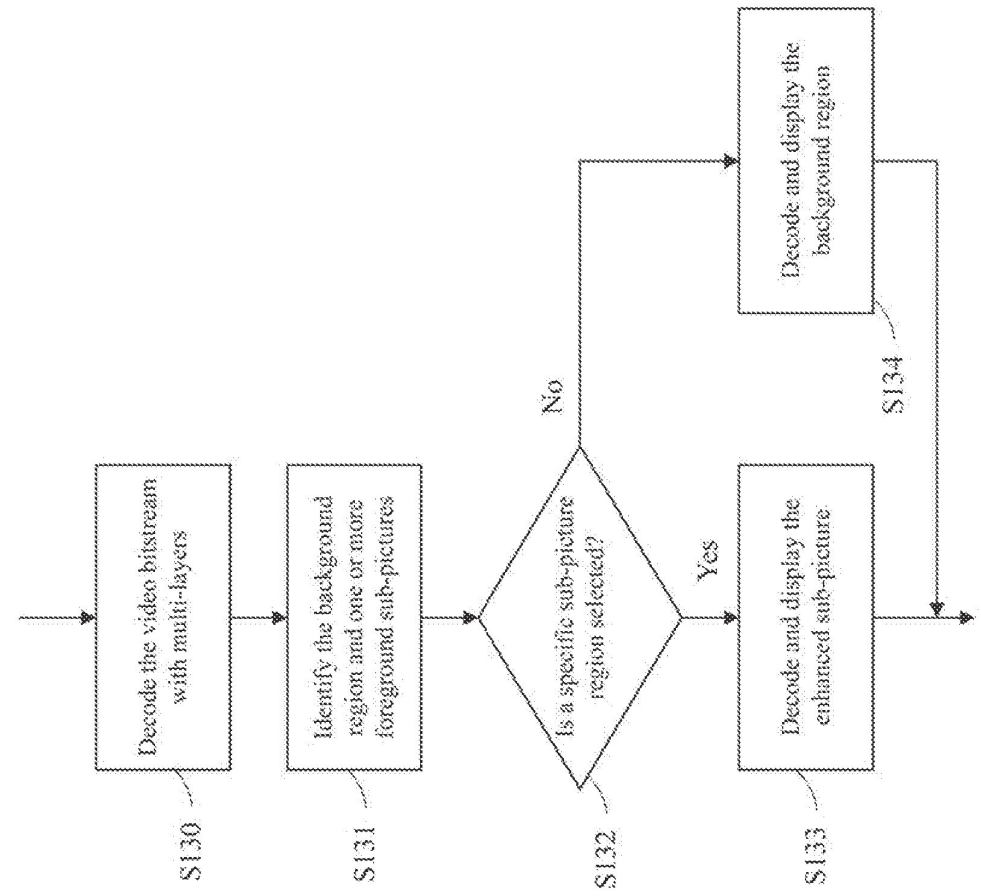

1300
Decode the video bitstream with multi-layers
S130
Identify the background region and one or more foreground sub-pictures
S131
Is a specific sub-picture region selected?
S132
No
Yes
Decode and display the enhanced sub-picture
S133
Decode and display the background region
S134

FIG. 14

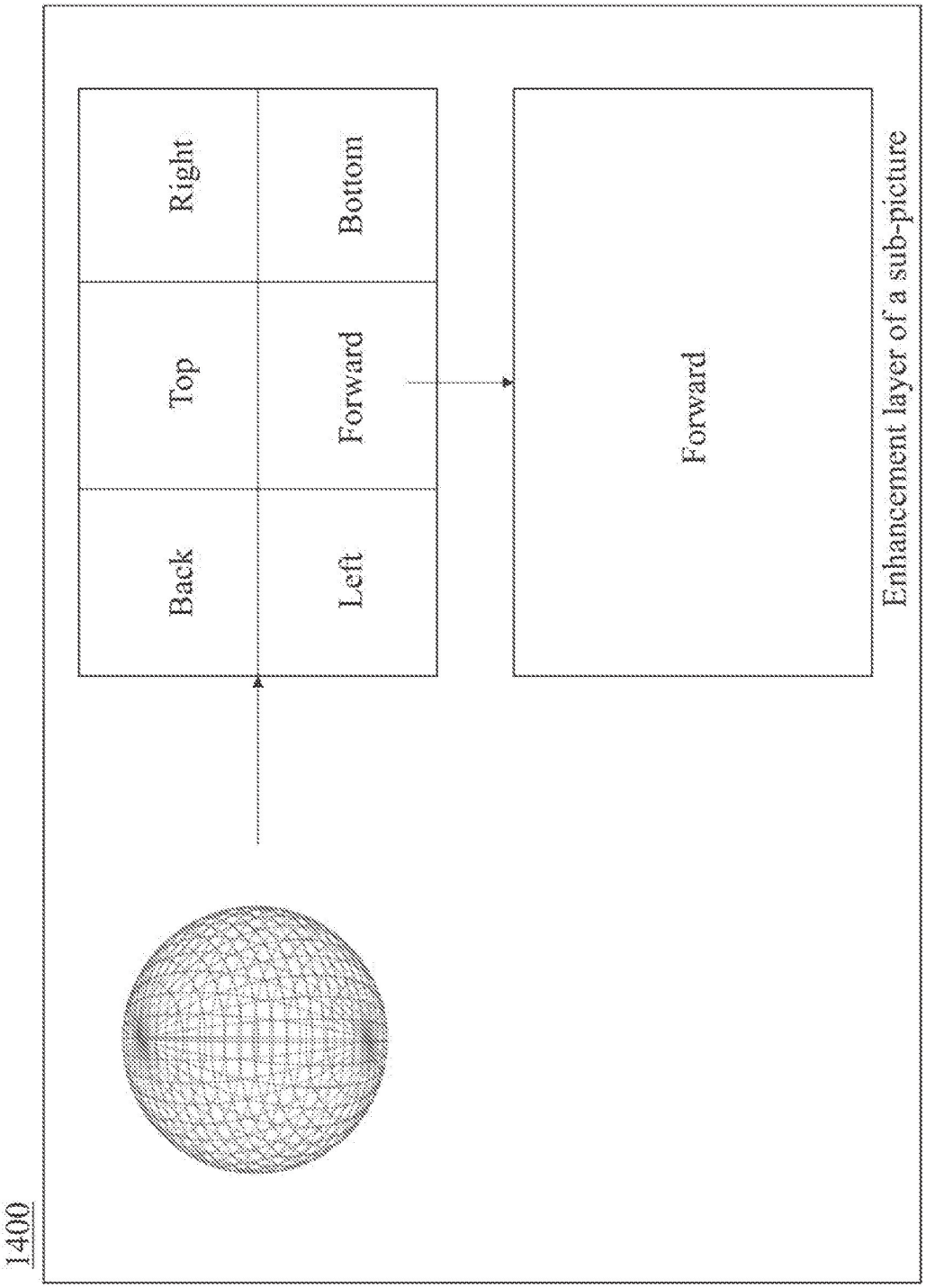

1400
Back
Top
Right
Left
Forward
Bottom
Forward
Enhancement layer of a sub-picture

Fig. 15

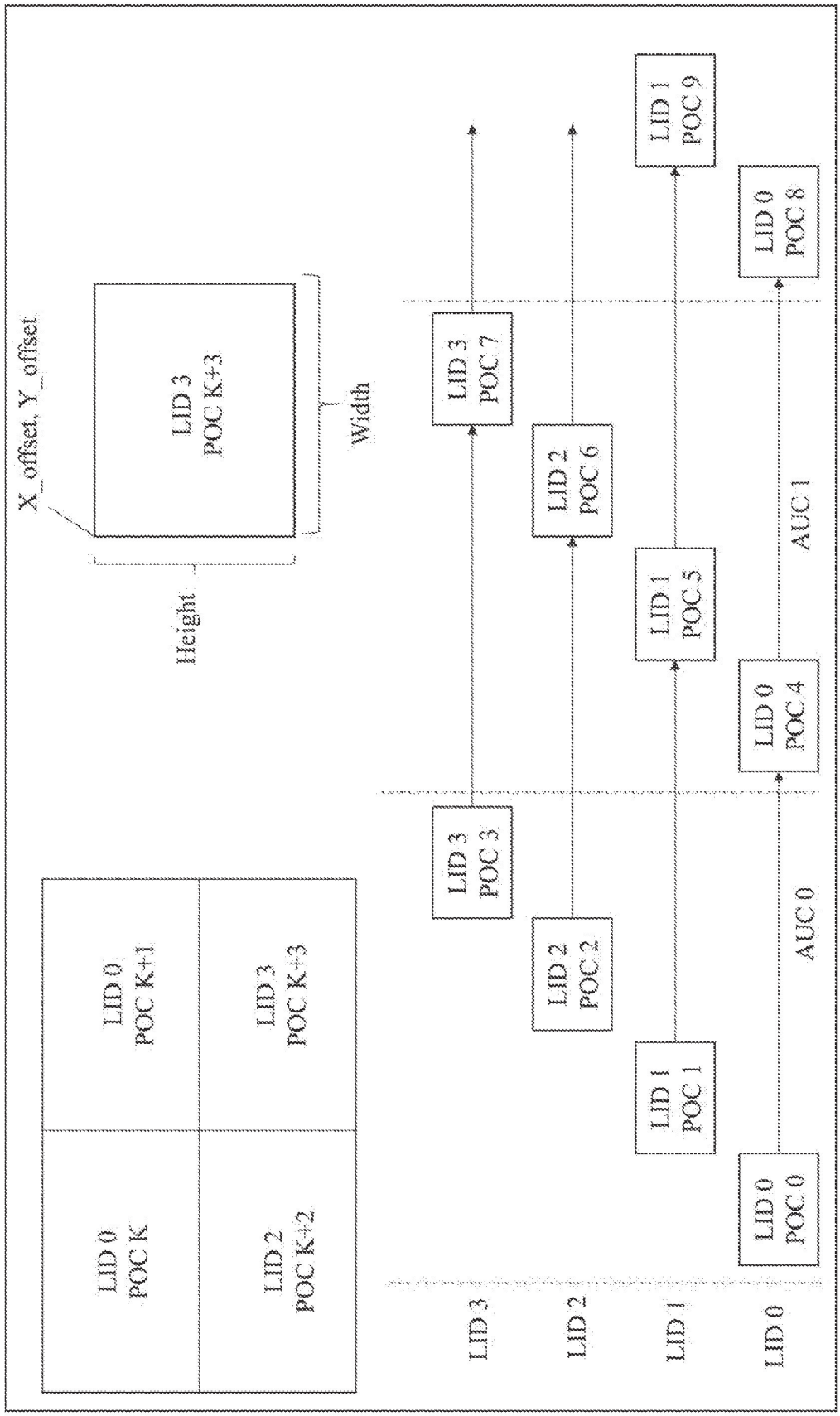

1500
LID 0
POC K
LID 0
POC K+1
LID 2
POC K+2
LID 3
POC K+3
X_offset, Y_offset
Height
Width
LID 3
POC K+3
LID 3
LID 2
LID 1
LID 0
LID 0
POC 0
LID 1
POC 1
LID 2
POC 2
LID 3
POC 3
LID 0
POC 4
LID 1
POC 5
LID 2
POC 6
LID 3
POC 7
LID 0
POC 8
LID 1
POC 9
AUC 0
AUC 1

Fig. 16

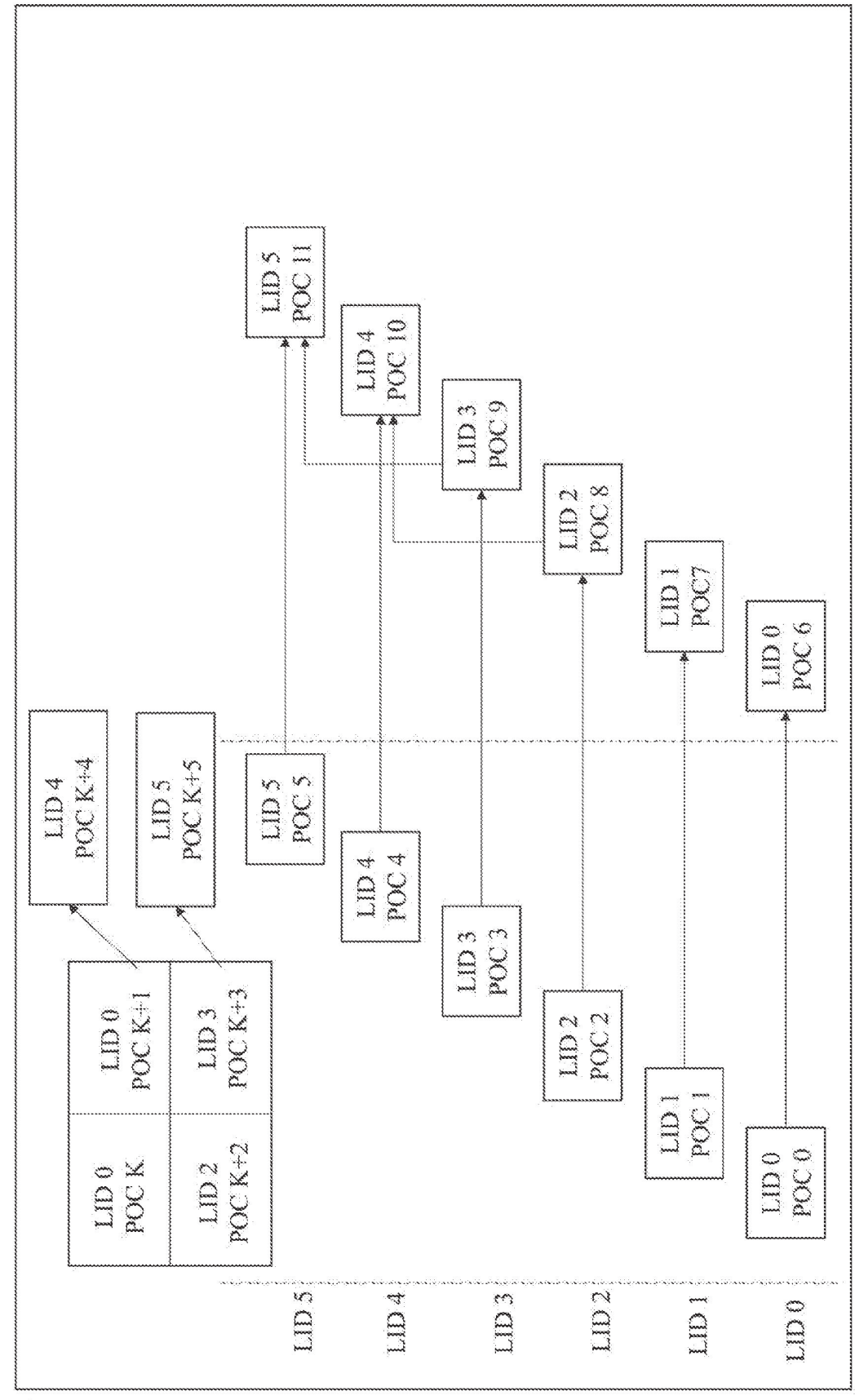

1600
LID 5
LID 4
LID 3
LID 2
LID 1
LID 0
LID 0 POC K
LID 0 POC K+1
LID 2 POC K+2
LID 3 POC K+3
LID 4 POC K+4
LID 5 POC K+5
LID 5 POC 5
LID 4 POC 4
LID 3 POC 3
LID 2 POC 2
LID 1 POC 1
LID 0 POC 0
LID 5 POC 11
LID 4 POC 10
LID 3 POC 9
LID 2 POC 8
LID 1 POC7
LID 0 POC 6

Fig. 17

1700

video_parameter_set rbsp( ) {	Descriptor
vps_video_parameter_set_id	u(4)
vps_max_layers_minus1	u(8)
for(i = 0; i <= vps_max_layers_minus1; i++) {	
vps_included_layer_id [i]	u(7)
vps_reserved_zero_bit	u(1)
}	
...	
vps_sub_picture_dividing_flag	u(1)
if(vps_sub_picture_dividing_flag) {	
vps_full_pic_width_in_luma_samples	ue(v)
vps_full_pic_height_in_luma_samples	ue(v)
}	
...	
}	

seq_parameter_set_rbsp( ) {	Descriptor
sps_decoding_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sub_layers_minus1	u(3)
...	
pic_width_in_luma_samples	ue(v)
pic_height_in_luma_samples	ue(v)
if(vps_sub_picture_dividing_flag) {	
pic_offset_x	ue(v)
pic_offset_y	ue(v)
}	
...	
}	

FIG. 18

1800

sub_region_partitioning_info(payloadSize) {	**Descriptor**
num_sub_region	ue(v)
num_layers	ue(v)
for (i = 0; i <= num_layers; i++)	
layer_id [i]	ue(v)
for (i = 1; i <= num_layers; i++)	
for (j = 0; j < i; j++)	
direct_dependency_flag [i] [j]	u(1)
for(i = 0; i < num_sub_region; i++) {	
num_layers_for_region[I]	ue(v)
for(i = 0; j < num_layers_for_region [i]; j++)	
sub_region_layer_id [i][j]	ue(v)
sub_region_offset_x[i]	ue(v)
sub_region_offset_y[i]	ue(v)
sub_region_width[i]	ue(v)
sub_region_height[i]	ue(v)
}	
...	
}	

FIG. 19

1900

video_parameter_set_rbsp() {	**Descriptor**
...	
vps_max_layers_minus1	u(6)
num_output_layer_sets	ue(v)
num_profile_tier_level	ue(v)
for(i = 0; i < num_profile_tier_level; i++)	
profile tier level (vps_max_sub_layers_minus1)	
for(i = 0; i < num_output_layer_sets; i++)	
for(j = 0; j < NumLayersInIdList [i]; j++) {	
output_layer_flag[i][j]	u(1)
profile_tier_level_idx[i][j]	u(v)
}	
...	
}	

FIG. 20

2000

video_parameter_set_rbsp() {	**Descriptor**
…	
vps_max_layers_minus1	u(6)
if(vps_max_layers_minus1 >0) {	
num_output_layer_sets	ue(v)
num_profile_tier_level	ue(v)
}	
for(i = 0; i < num_profile_tier_level; i++)	
profile tier level (vps_max_sub_layers_minus1)	
for(i = 0; i < num_output_layer_sets; i++)	
vps_output_layers_mode[i]	u(2)
vps_ptl_signal_flag[i]	u(1)
for(j = 0; j < NumLayersInIdList [i]; j++) {	
if(vps_output_layers_mode[i] = = 2)	
output_layer_flag[i][j]	u(1)
if(vps_pts_signal_flag[i])	
profile_tier_level_idx[i][j]	u(v)
}	
}	
…	
}	

Fig. 21

2100

video_parameter_set_rbsp() {	Descriptor
...	
vps_max_layers_minus1	u(6)
if(vps_max_layers_minus1 >0) {	
num_output_layer_sets	ue(v)
num_profile_tier_level	ue(v)
}	
max_subpics_minus1	u(8)
for(i = 0; i < **max_subpics_minus1**; i++){	
sub_pic_id[i]	u(8)
}	
for(i = 0; i < num_profile_tier_level; i++)	
profile tier level (vps_max_sub_layers_minus1)	
for(i = 0; i < num_output_layer_sets; i++)	
vps_output_layers_mode[i]	u(2)
vps_ptl_signal_flag[I]	u(1)
for(j = 0; j < NumLayersInIdList [I]; j++) {	
num_output_subpics_layer[i][j]	ue(v)
for(k = 0; k < num_output_subpic_layers[i][j]; k++)	
sub_pic_id_layer[i][j] [k]	u(8)
if(vps_output_layers_mode[i] = = 2)	
output_layer_flag[I][j]	u(1)
if(vps_pts_signal_flag[i])	
profile_tier_level_idx[i][j]	u(v)
}	
}	
...	
}	