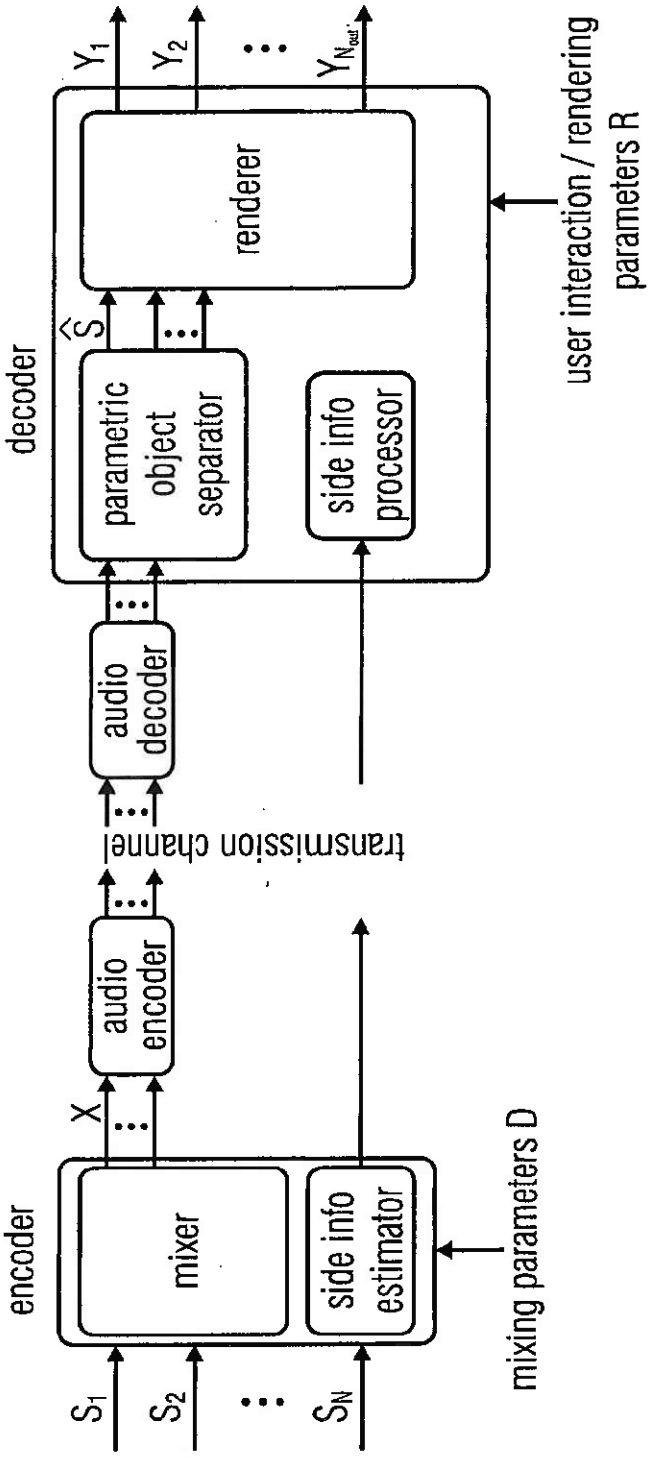
encoder
mixer
side info estimator
mixing parameters D
X
audio encoder
transmission channel
audio decoder
decoder
parametric object separator
side info processor
renderer
user interaction / rendering parameters R

Fig. 1

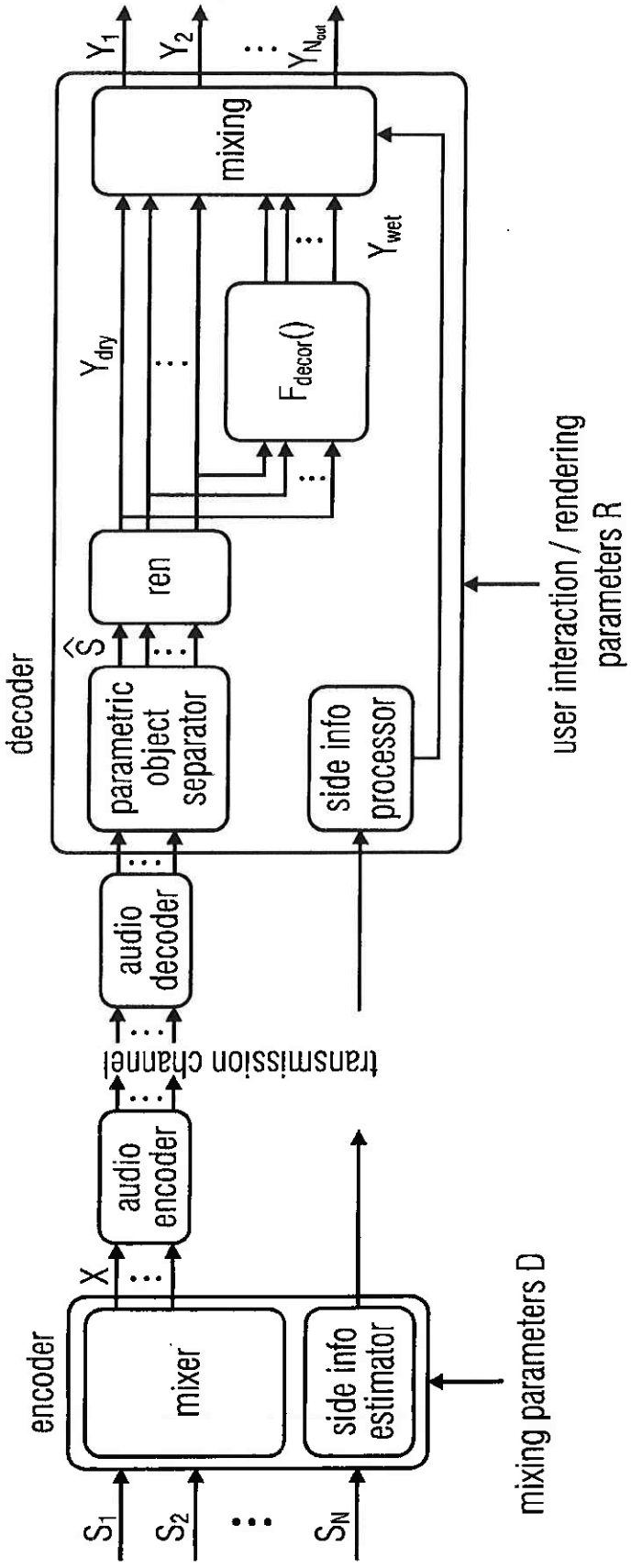
encoder
mixer
side info estimator
audio encoder
X
transmission channel
audio decoder
decoder
parametric object separator
side info processor
ren
$\hat{S}$
Y_{dry}
$F_{decor}()$
Y_{wet}
mixing
Y_1
Y_2
$Y_{N_{out}}$
S_1
S_2
S_N
mixing parameters D
user interaction / rendering parameters R

Fig. 2

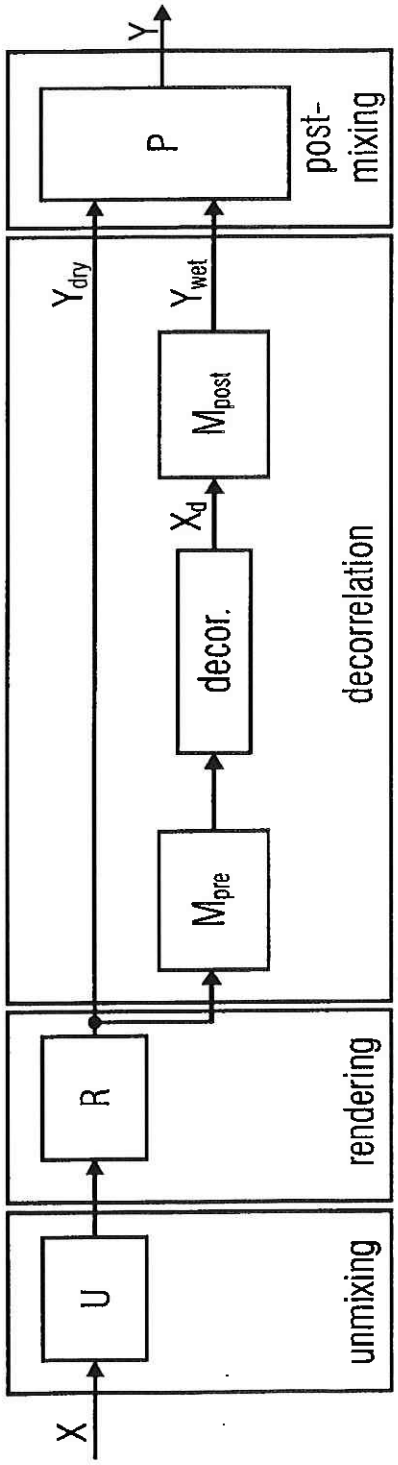
X
U
unmixing
R
rendering
Y_{dry}
Y_{wet}
M_{pre}
decor.
X_d
M_{post}
decorrelation
P
post-mixing
Y

Fig. 3

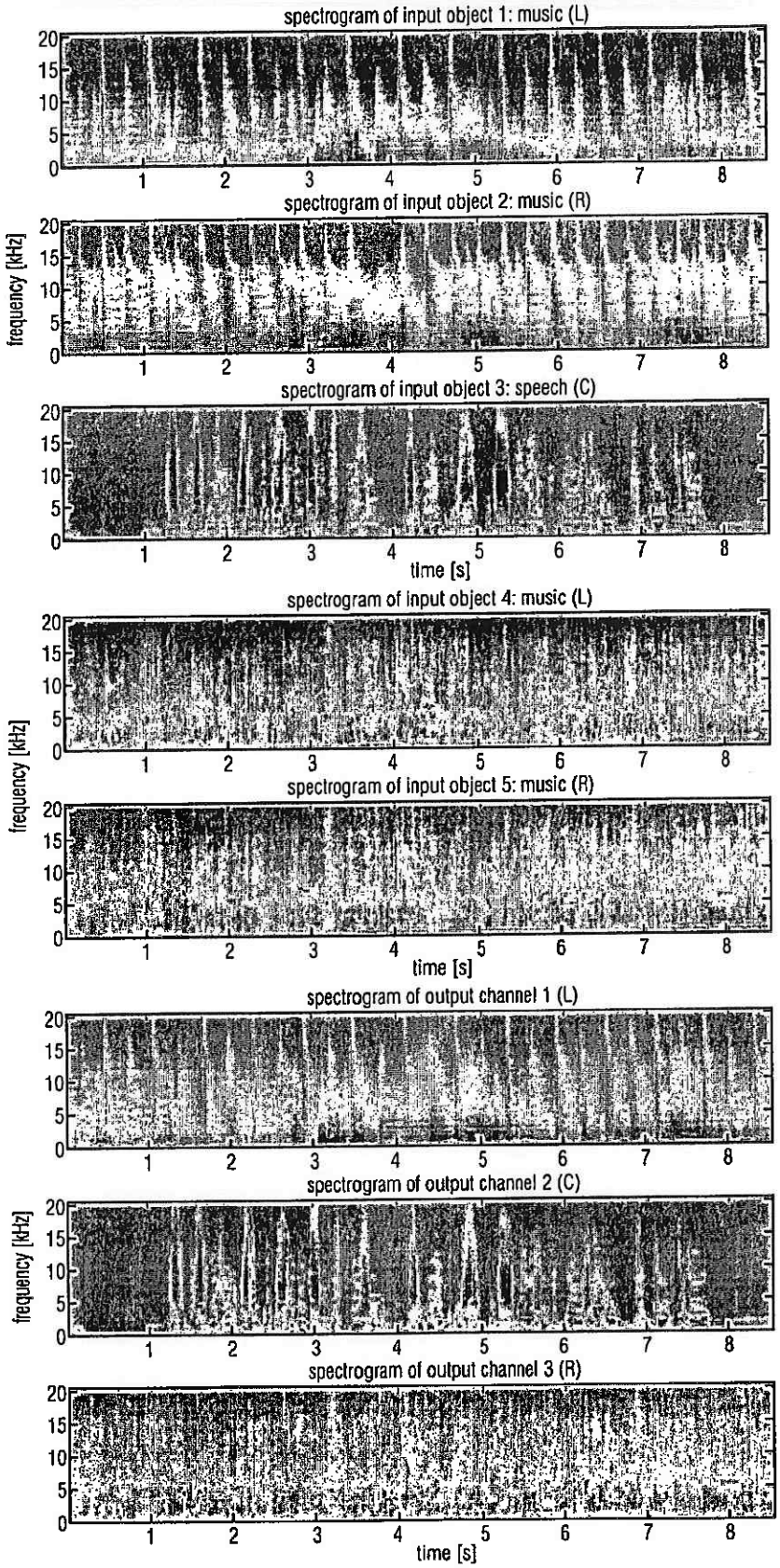
spectrogram of input object 1: music (L)
spectrogram of input object 2: music (R)
spectrogram of input object 3: speech (C)
spectrogram of input object 4: music (L)
spectrogram of input object 5: music (R)
spectrogram of output channel 1 (L)
spectrogram of output channel 2 (C)
spectrogram of output channel 3 (R)
frequency [kHz]
time [s]

Fig. 4

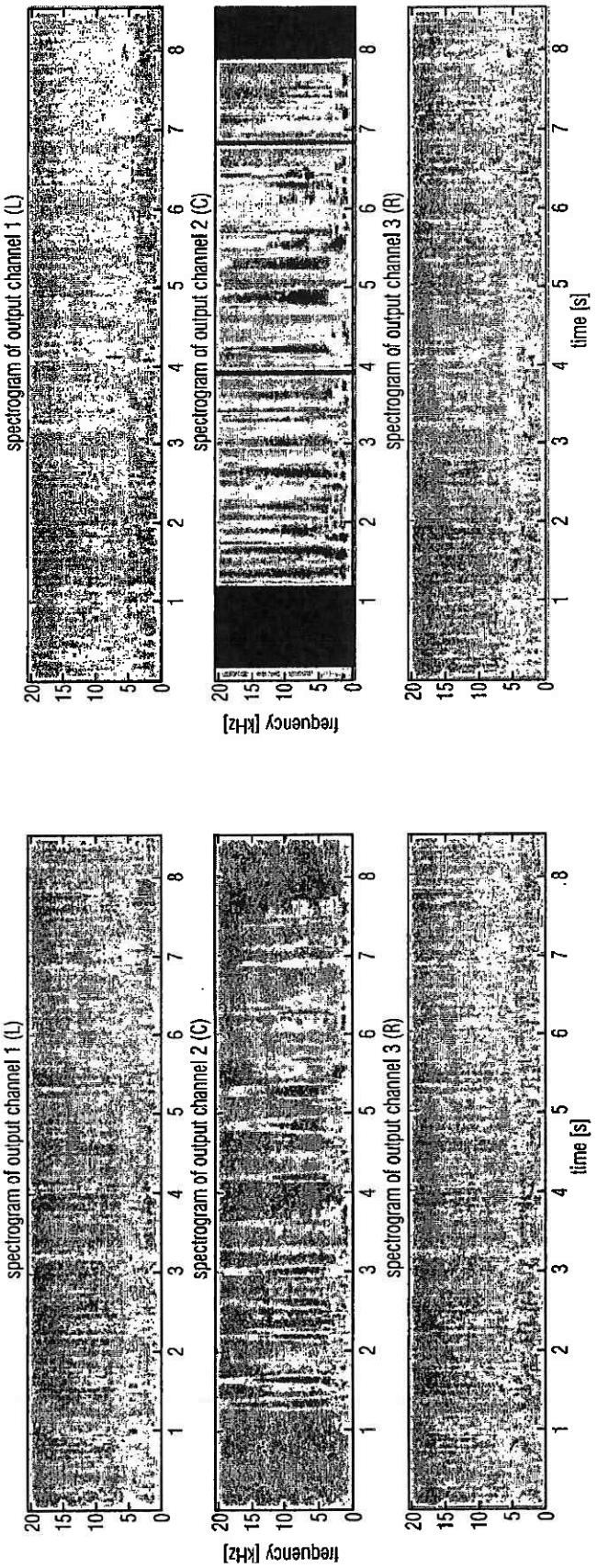
spectrogram of output channel 1 (L)
spectrogram of output channel 2 (C)
spectrogram of output channel 3 (R)
frequency [kHz]
time [s]

Fig. 5

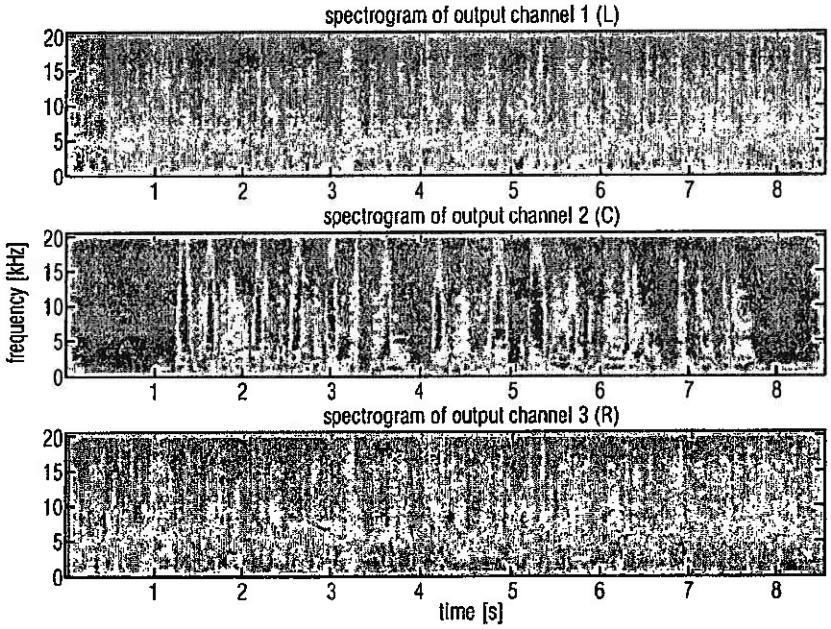
spectrogram of output channel 1 (L)
spectrogram of output channel 2 (C)
spectrogram of output channel 3 (R)
frequency [kHz]
time [s]
0
5
10
15
20
1
2
3
4
5
6
7
8

Fig. 6

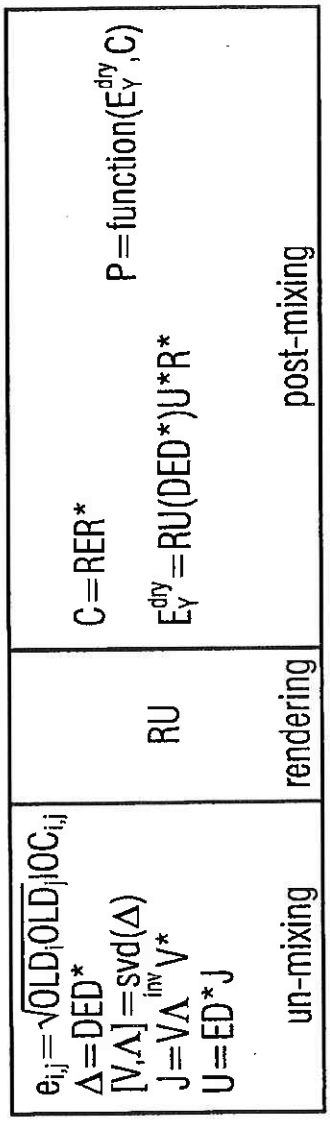
$e_{i,j}=\sqrt{OLD_i OLD_j} OC_{i,j}$
$\Delta=DED^*$
$[V,\Lambda]=svd(\Delta)$
$J=V\Lambda^{inv}V^*$
$U=ED^*J$
un-mixing
RU
rendering
$C=RER^*$
$E_Y^{dry}=RU(DED^*)U^*R^*$
$P=function(E_Y^{dry},C)$
post-mixing

Fig. 7

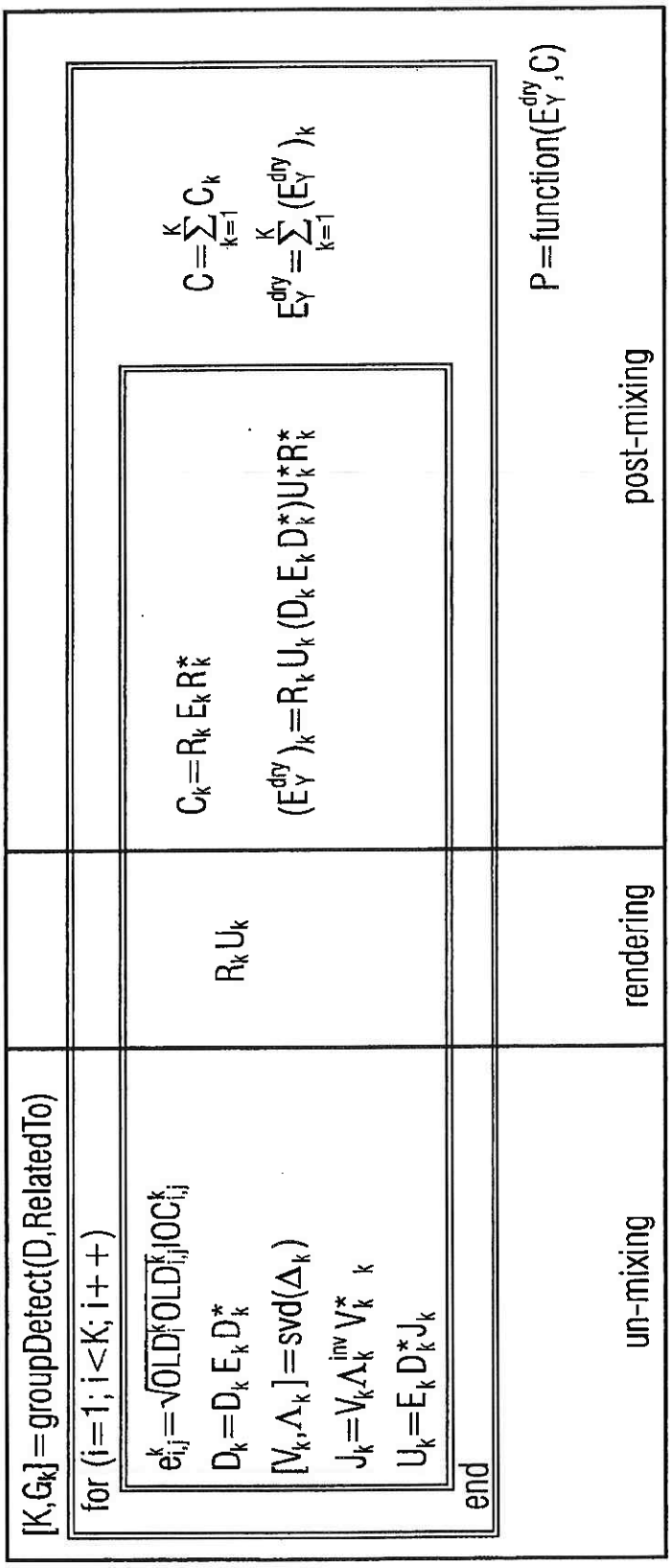
[K,G_k]=groupDetect(D,RelatedTo)
for (i=1; i<K; i++)
e^k_{i,j}=√OLD^k|OLD^k_{i,j}|OC^k_{i,j}
D_k=D_k E_k D*_k
[V_k,Λ_k]=svd(Δ_k)
J_k=V_k Λ_k^inv V*_k k
U_k=E_k D*_k J_k
end
un-mixing
R_k U_k
rendering
C_k=R_k E_k R*_k
(E_Y^dry)_k=R_k U_k (D_k E_k D*_k)U*_k R*_k
C=Σ_{k=1}^K C_k
E_Y^dry=Σ_{k=1}^K (E_Y^dry)_k
post-mixing
P=function(E_Y^dry,C)

Fig. 8

```
#define SAOC_MAX_OBJECTS                128
#define SAOC_MAX_DMX_CHANNELS           32

#define SAOC_MAX_GROUP_OBJECTS          32 /* Maximum number of objects in one group */

#define SAOC_MAX_ GROUP_DMX_CHANNELS    16 /* Maximum number of downmix channels in one group */

typedef struct saocCoreGroupList {
 int numDmx;
 int numObj;
 int dmxList[SAOC_MAX_DMX_CHANNELS];
 int objList[SAOC_MAX_OBJECTS];
} saocCoreGroupList;

typedef struct saocCoreGroup {
 int numGroups;
 saocCoreGroupList saocGroup[SAOC_MAX_DMX_CHANNELS];
} saocCoreGroup;

static int getSaocCoreGroups(int CrossCov[SAOC_MAX_DMX_CHANNELS][SAOC_MAX_OBJECTS],
                             int numObjects,
                             int numDmxChannels,
                             saocCoreGroup *groupInfo)
{
 int iGr, iObj, jObj, iDmx, jDmx, indexObj, indexChan, groupIndex, foundGroup;

 int channelsFlag[SAOC_MAX_DMX_CHANNELS];
 int objectsFlag[SAOC_MAX_OBJECTS];

 /* Init empty grouping info */
 for (iGr = 0; iGr<SAOC_MAX_DMX_CHANNELS; iGr++) {
  for (iObj = 0; iObj < numObjects; iObj++){
   groupInfo->saocGroup[iGr].objList[iObj] = -1;
  }
  for (iDmx = 0; iDmx <SAOC_MAX_DMX_CHANNELS; iDmx++) {
   groupInfo->saocGroup[iGr].dmxList[iDmx] = -1;
  }
  groupInfo->saocGroup[iGr].numDmx = 0;
  groupInfo->saocGroup[iGr].numObj = 0;
 }
 groupInfo->numGroups = 0;

 for (iObj = 0; iObj < numObjects; iObj++){
  objectsFlag[iObj] = -1;
 }
 for (jDmx = 0; jDmx <numDmxChannels; jDmx++) {
  channelsFlag[jDmx] = jDmx;
 }
```

Fig. 9 (1)

```
for (iDmx = 0; iDmx <numDmxChannels; iDmx++) {
 for (iObj = 0; iObj < numObjects; iObj++) {          /* search for not assigned objects */
  if (CrossCov[iDmx][iObj]>0) {
   if (objectsFlag[iObj] == -1) {                     /* iObj not assigned to any group */
    objectsFlag[iObj] = iDmx;                         /* assign iObj to iDmx group*/
   } else {                                           /* iObj already assigned to a group*/
foundGroup = objectsFlag[iObj];
for (jObj = 0; jObj < numObjects; jObj++) {          /* assign all objects from this existing
                                                      group to the new one */
  if (objectsFlag[jObj] == foundGroup) {
  objectsFlag[jObj] = iDmx;
 }
}
for (jDmx = 0; jDmx <iDmx; jDmx++) {                 /* assign all channels from this existing
                                                      group to the new one */
 if (channelsFlag[jDmx] == foundGroup) {
  channelsFlag[jDmx] = iDmx;
     }
    }
   }
  }
 }
}
for (iObj = 0; iObj < numObjects; iObj++) {
  iGr = groupInfo->numGroups;
  if (objectsFlag[iObj] > -1) {                      /* object not assigned to any group */
   groupIndex = objectsFlag[iObj];
   for (jObj = 0; jObj < numObjects; jObj++) {       /* assign all objects from this group
                                                      into the object list */
    if (objectsFlag[jObj] == groupIndex) {
 indexObj = groupInfo->saocGroup[iGr].numObj++;
 groupInfo->saocGroup[iGr].objList[indexObj] = jObj;
 objectsFlag[jObj] = -1;                             /* mark object as assigned to an object
                                                      list */
    }
   }
   for (iDmx = 0; iDmx <numDmxChannels; iDmx++) {    /* assign all chnls from this group into
                                                      the list */
    if (channelsFlag[iDmx] == groupIndex) {
     indexChan = groupInfo->saocGroup[iGr].numDmx++;
     groupInfo->saocGroup[iGr].dmxList[indexChan] = iDmx;
     channelsFlag[iDmx] = -1;                        /* mark channel as assigned to a channel
                                                      list */
    }
   }
   groupInfo->numGroups++;
  }
}
```

Fig. 9 (2)

```
  for (iGr = 0; iGr<groupInfo->numGroups; iGr++) {
   assert(groupInfo->saocGroup[iGr].numDmx < SAOC_MAX_GROUP_DMX_CHANNELS);
   assert(groupInfo->saocGroup[iGr].numObj < SAOC_MAX_GROUP_OBJECTS);
  }

  return 0;
}

static void getSaocDmxPossitions(float Dmx[SAOC_MAX_DMX_CHANNELS][SAOC_MAX_OBJECTS],
                         int RelatedToMatrix[SAOC_MAX_OBJECTS][SAOC_MAX_OBJECTS],
                         int CrossCov[SAOC_MAX_DMX_CHANNELS][SAOC_MAX_OBJECTS],
                         int numObjects,
                         int firstObject,
                         int numDmxChannels,
                         int firstDmxChannel)
{
  int ch, dCh, obj, obj2, objIdx, objIdx2;
  int D[SAOC_MAX_DMX_CHANNELS][SAOC_MAX_OBJECTS] = {{0}};
  float minVal = (float)pow(10.0f,-150.0f/20.0f) + SAOC_EPSILON;

  for (ch=0; ch<numDmxChannels; ch++) {
   dCh = firstDmxChannel + ch;
   for (obj=0; obj<numObjects; obj++) {
    D[ch][obj] = (minVal<Dmx[dCh][firstObject + obj] ? 1 : 0);
   }
  }

  for (ch=0; ch<numDmxChannels; ch++) {
   dCh = firstDmxChannel + ch;
   for (obj=0; obj<numObjects; obj++) {
    objIdx = firstObject + obj;
    CrossCov[ch][obj] = 0.0f;
    for (obj2=0; obj2<numObjects; obj2++) {
     objIdx2 = firstObject + obj2;
     CrossCov[ch][obj] += D[ch][obj] * RelatedToMatrix[objIdx][objIdx2];
    }
   }
  }
}
static int getSaocCoreGrouping(float Dmx[][SAOC_MAX_OBJECTS],
                               int RelatedToMatrix[SAOC_MAX_OBJECTS][SAOC_MAX_OBJECTS],
                               int numObjects,
                               int numDmxChannels,
                               saocCoreGroup *groupInfo)
{
  int CrossCov[SAOC_MAX_DMX_CHANNELS][SAOC_MAX_OBJECTS] = {{0}};

  getSaocDmxPossitions(Dmx, RelatedToMatrix, CrossCov, numObjects, 0, numDmxChannels, 0);
  getSaocCoreGroups(CrossCov, numObjects, numDmxChannels, groupInfo);

  return 0;
}
```

Fig. 9 (3)

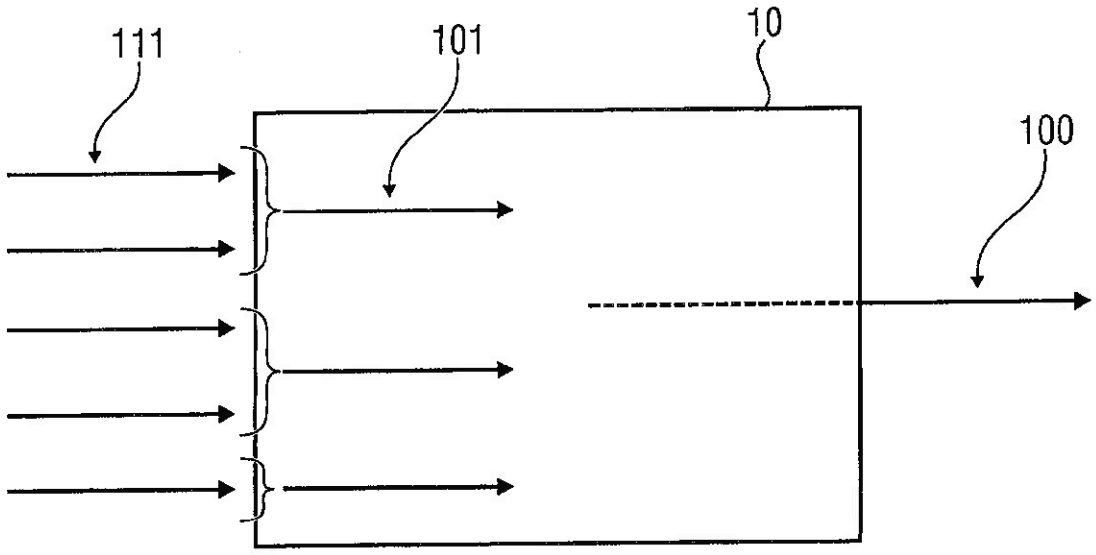

Fig. 10

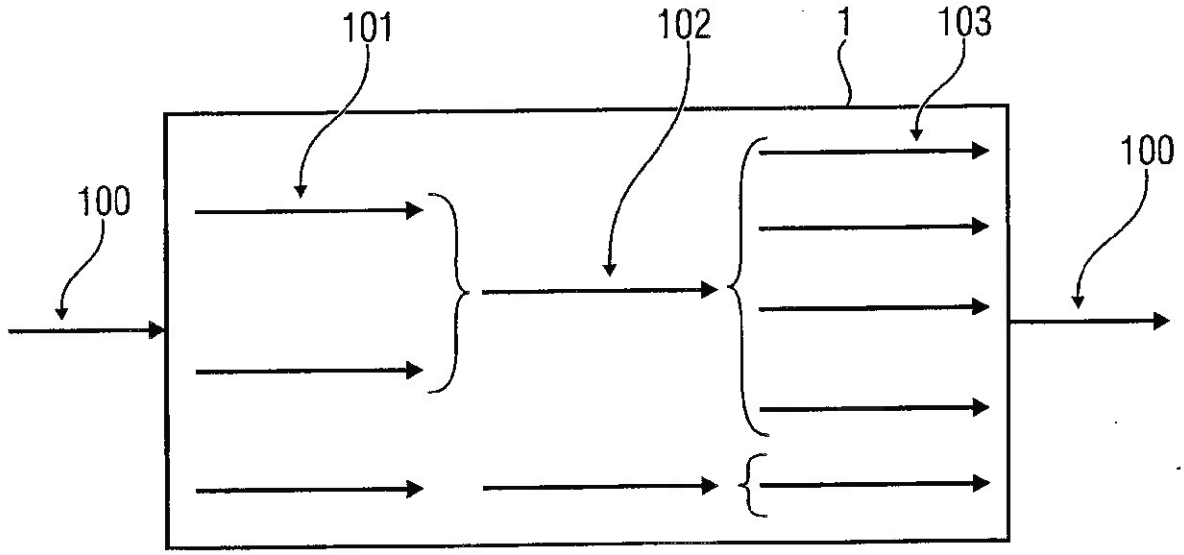

Fig. 11

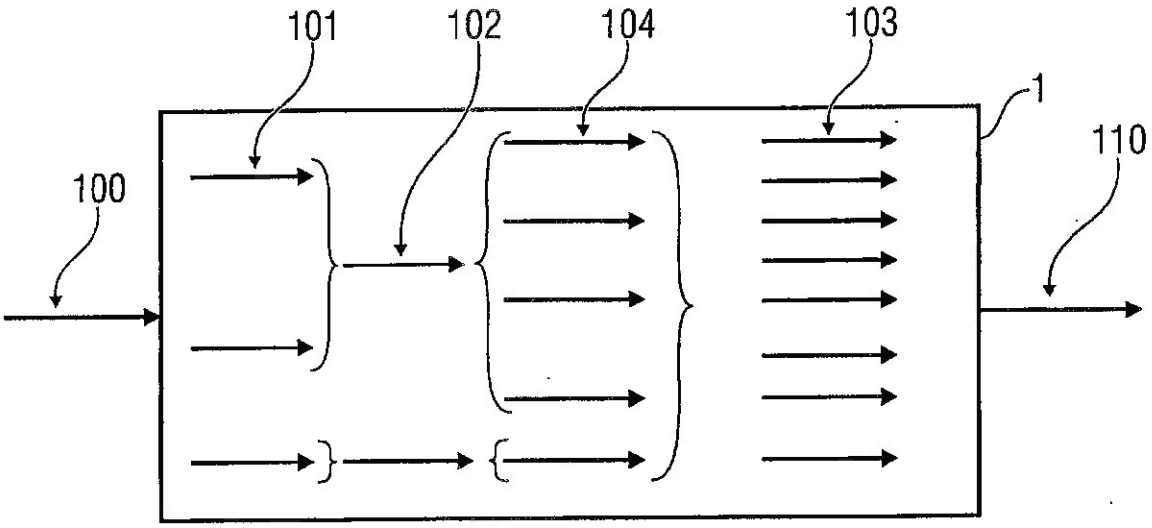

Fig. 12

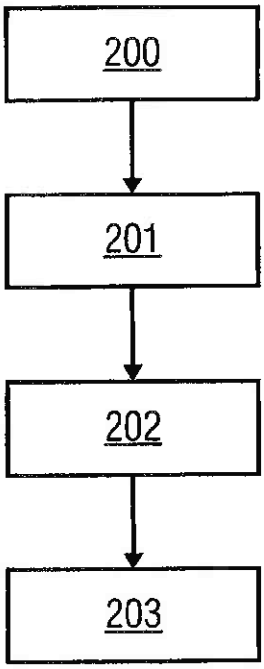

Fig. 13

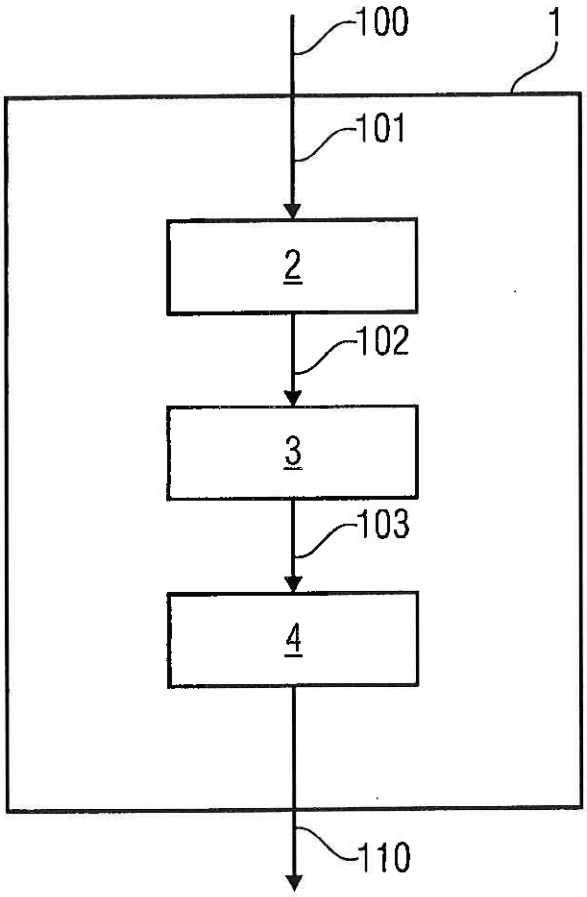

Fig. 14

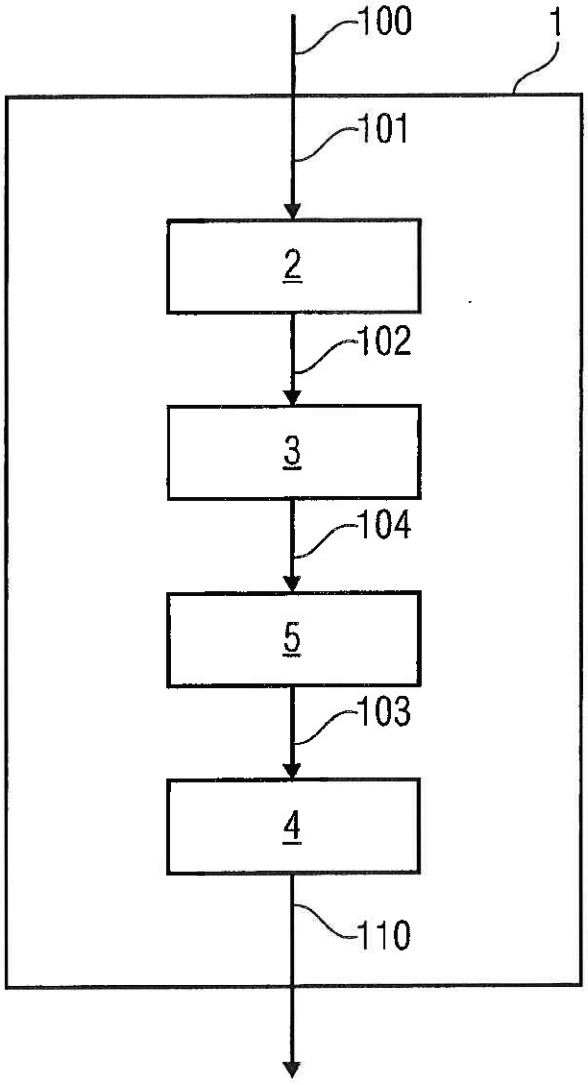

Fig. 15

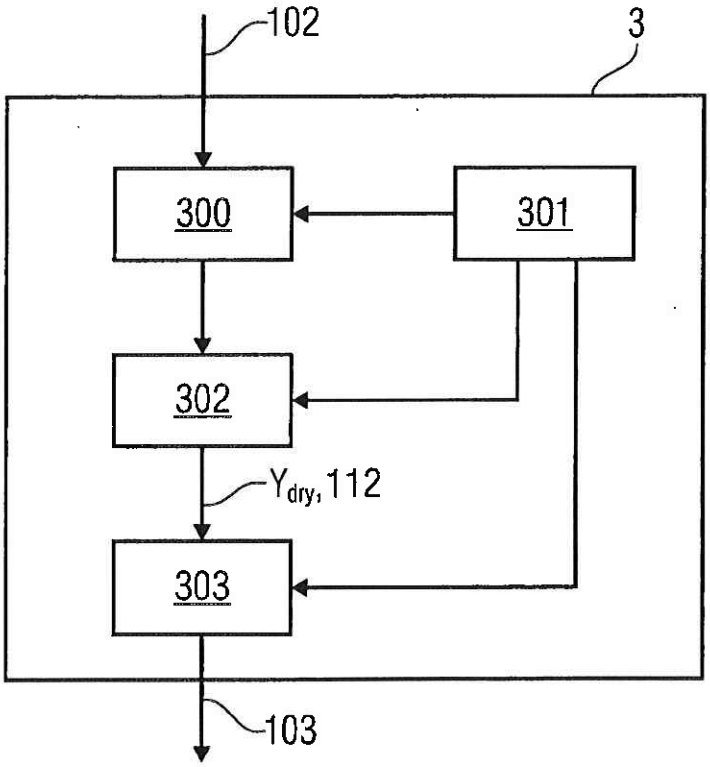

Fig. 16

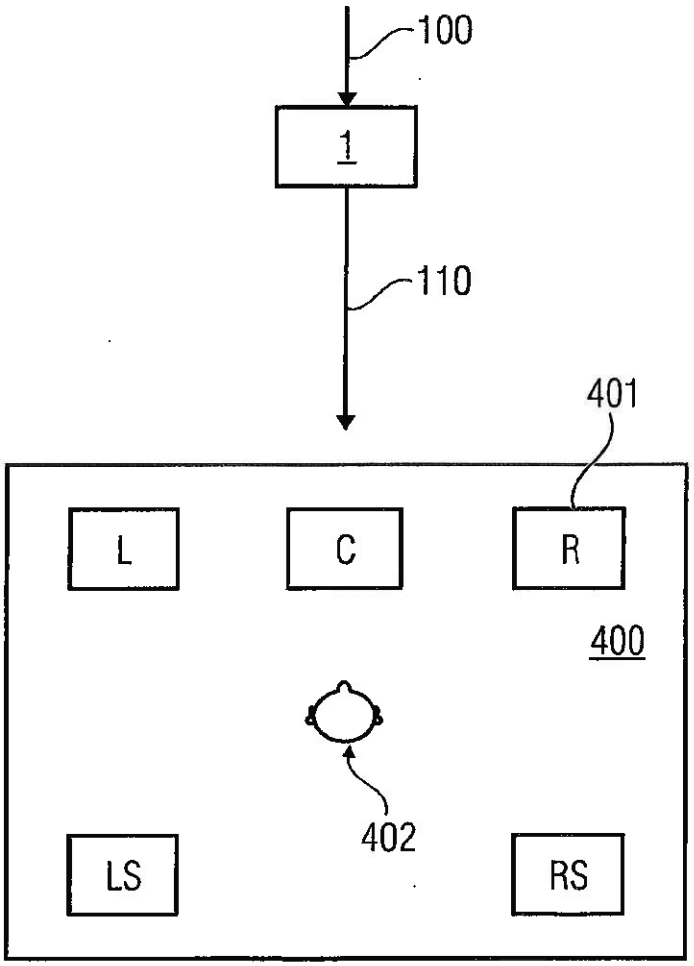

Fig. 17