Raumsimulation mittels Spiegelquellen für zeitvariante Binauralsysteme

Projektarbeit

von

Wolfgang Irnberger

Institut für Elektronische Musik und Akustik
an der Universität für Musik und darstellende Kunst in Graz
Projektleiter: o.Univ.Prof.Mag DI Dr.techn. Robert Höldrich

Graz, im Juni 2002

Inhaltsverzeichnis

1 Aufgabenstellung

Dieses Projekt ist eine Weiterarbeit am bestehenden System, entwickelt von *Markus Noisternig* und *Piotr Majdak*. Eine genaue Beschreibung des bestehenden Systems und eine umfangreiche Bedienungsanleitung kann im IEM Report (2001)

„A Head Position Related Binaural Sound System“

nachgelesen werden. Diese Projektbeschreibung befasst sich mit den Änderungen und Erweiterungen des Systems.

Ein bereits bestehendes System zur virtuellen Abbildung von Schallfeldern mittels Ambisonic auf dem DSP TMS320C6711 soll erweitert werden. Im bisherigen System werden die Schallquellen im freien Raum angenommen. In diesem Projekt soll nun eine Raumsimulation eingebunden werden. Der Raum kann mit Hilfe von virtuellen Quellen, sogenannten Spiegelquellen nachgebildet werden. Da dies einen erhöhten Speicheraufwand erfordert, muss der auf dem Evaluation Board vorhandene externe Speicher angesprochen und der Speicherzugriff hinsichtlich der Verarbeitungsgeschwindigkeit optimiert werden.

Die Programmierung des DSP erfolgt in C. Die Programmoberfläche für den Benutzer ist in Visual Basic programmiert, wo auch ein Teil der Berechnungen bereits vorab erledigt wird um am DSP die Resourcen für die Berechnungen in Echtzeit freizuhalten.

2 Ambisonic

Um die ursprüngliche Welle nachzubilden wird ein System 3. Ordnung verwendet. Es ergeben sich für den zweidimensionalen Fall daher $(2m+1)$ 7 Übertragungskanäle. Es wurde der vorhandene Algorithmus um den Ringbuffer erweitert (*Abbildung 2.1*) und gering modifiziert.

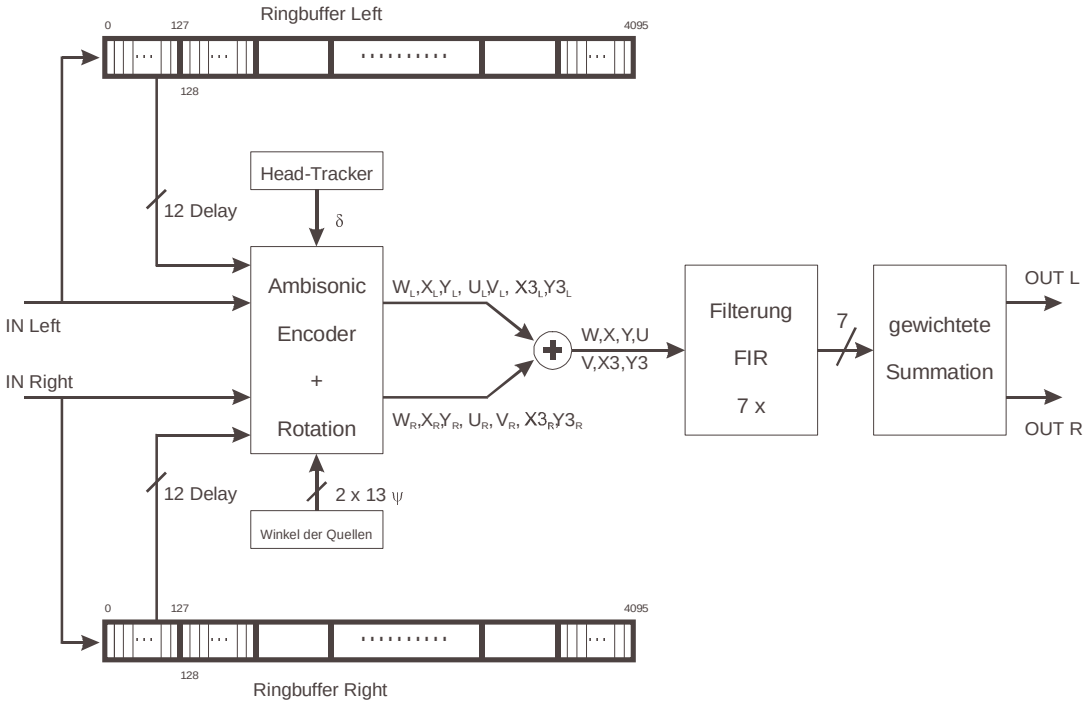


Abbildung 2.1

Ein großer Vorteil der Ambisonic – Technik liegt darin, dass die Anzahl an Eingangssignalen beliebig erhöht werden kann und nach dem Ambisonic - Encoder immer nur mehr in diesem Fall 7 Kanäle zu den weiteren Berechnungen benötigt werden.

3 Raumberechnung

3.1 Spiegelquellen

Der Raum wird durch Spiegelquellen beschrieben. Die Anzahl der Spiegelquellen errechnet sich aus

$$k = 2n(n+1) \tag{3.1}$$

wobei n die Ordnung angibt.

Bei 2. Ordnung, Stereo sind also 2 * 12 Spiegelquellen zu rechnen. Die Verteilung der Quellen ist in *Abbildung 3.1* zu sehen.

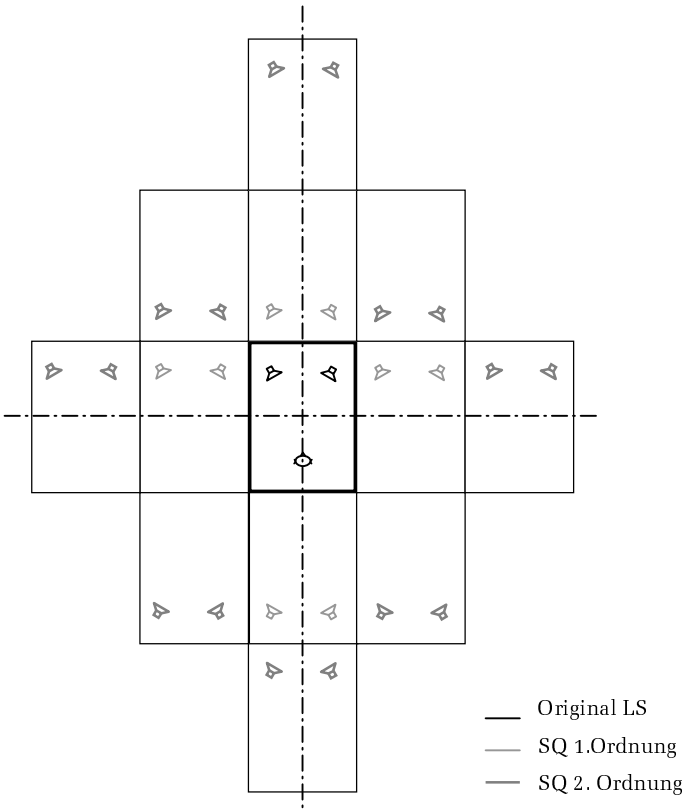


Abbildung 3.1

Für jede Quelle muss für das Ambisonic-System der Einfallswinkel und die Verzögerung der Schallwelle berechnet werden.

3.2 Delay

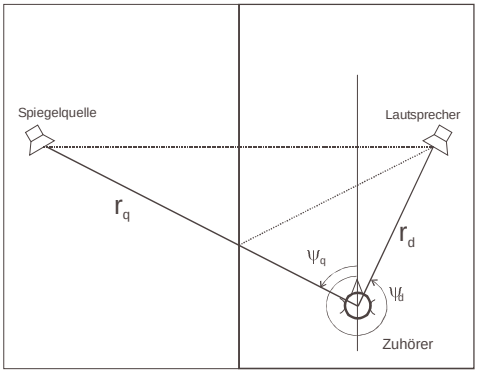


Abbildung 3.2

Aus der Geometrie des Raumes und der Position des Lautsprechers (*Abbildung 3.2*) lässt sich einfach der Winkel ψ und die Verzögerung d_m berechnen.

$$
\begin{aligned}
& d_m = r_q - r_d \quad in\,Meter \\
& f_T = 48kHz \qquad \frac{1}{f_T} = 20{,}833\mu s \qquad c = 340\frac{m}{s} \\
& \lambda = \frac{c}{f_T} = \frac{340}{48000} = 7{,}083mm \\
& d_s = \frac{d_m}{\lambda} \qquad in\,Samples
\end{aligned}
\tag{3.2}
$$

Ein Delay von 1 Sample entspricht 20,8µs und einem Weg von 7mm. Für einen Raum mit den Abmessungen 14m x 14m ergibt sich somit ein maximales Delay von 3954 Samples oder 31 Frames.

Weiters geht die Entfernung der Spiegelquelle mit $\frac{1}{r}$ in die Berechnung ein.

Der Abstand der Lautsprecher wird mit Faktor 1 ≅ 32767 bei 16 Bit Auflösung festgelegt, für die Spiegelquellen gilt dann der Faktor

$$\frac{r_d}{r_q} \cdot 32767 \tag{3.3}$$

3.3 Wandeigenschaften

An den Wänden erfolgt eine Filterung mit einem Tiefpass 1. Ordnung (*Abbildung 3.3*)

$$y_n = \beta \cdot [\alpha \cdot y_{n-1} + (\alpha - 1) \cdot x_n] \qquad \begin{matrix} 0 \le \alpha \le 1 \\ 0 \le \beta \le 1 \end{matrix} \tag{3.4}$$

wobei α der Filterkoeffizient ist und β der Reflexionsfaktor der Wand. Es werden alle Wände mit gleichen Eigenschaften angenommen.

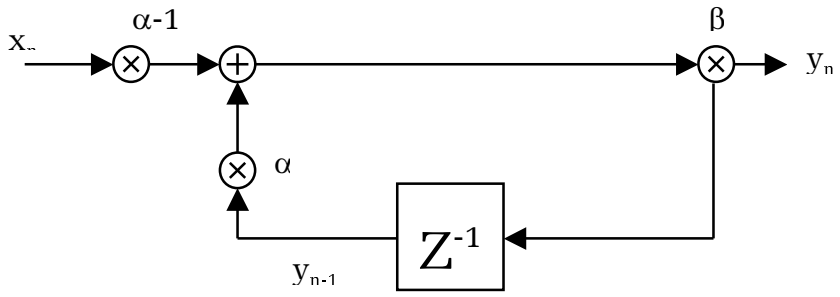


Abbildung 3.3

4 Programmierung

4.1 Host Port Interface HPI des DSP

Über die HPI läuft die Kommunikation von PC mit dem DSP, sie wird vom PC gesteuert. Der gemeinsame Speicherbereich wird in der Datei `command.6lx` auf eine bestimmte Größe und in einen eigenen Bereich festgelegt:

```
MEMORY
{
I_HPI_MEM:  o = 00000180h   l = 00000230h
}

SECTIONS
{
    HPI_Section >     I_HPI_MEM
}
```

In der Header Datei `hpi.h` ist die dazugehörige Variable für den DSP deklariert:

```
#pragma DATA_SECTION(uiHPIBuffer, "HPI_Section")
#define HPI_BUFFER_LEN     139
uint uiHPIBuffer[HPI_BUFFER_LEN];
```

In `uiHPIBuffer` werden nun folgende Daten übergeben:

Parameter	Index	Bedeutung
HPI_Mirror	(uiHPIBuffer[0])	Anzahl der zu rechnenden Spiegelquellen
	(uiHPIBuffer[1])	frei
HPI_DeltaD	(uiHPIBuffer[2])	Erwünschte Stellung des Kopfes
HPI_Control	(uiHPIBuffer[3])	Kontrollwort für Programmablauf
HPI_OutputGain	(uiHPIBuffer[4])	Verstärkungsfaktor für den Ausgang
HPI_MaxInput	(uiHPIBuffer[5])	Maximum des Signals am Eingang
HPI_MaxOutput	(uiHPIBuffer[6])	Maximum des Signals am Ausgang (vor dem Verstärkungsfaktor)
HPI_MaxW	(uiHPIBuffer[7])	Maximum des W-Signals
HPI_MaxX	(uiHPIBuffer[8])	Maximum des X-Signals
HPI_MaxY	(uiHPIBuffer[9])	Maximum des Y-Signals
HPI_MaxU	(uiHPIBuffer[10])	Maximum des U-Signals
HPI_MaxV	(uiHPIBuffer[11])	Maximum des V-Signals
HPI_MaxX3	(uiHPIBuffer[12])	Maximum des X3-Signals
HPI_MaxY3	(uiHPIBuffer[13])	Maximum des Y3-Signals
HPI_CPULoad	(uiHPIBuffer[16])	Die momentane Auslastung des DSPs
HPI_ISRCount	(uiHPIBuffer[17])	Anzahl der berechneten Frames
HPI_DIPSwitches	(uiHPIBuffer[18])	Die momentane Stellung der DIP-Schalter
HPI_DeltaN	(uiHPIBuffer[19])	die Stellung des Kopfes, momentan

HPI_Buf20	(uiHPIBuffer[20])	frei zum debuggen
HPI_Buf21	(uiHPIBuffer[21])	
HPI_Buf22	(uiHPIBuffer[22])	
HPI_Buf23	(uiHPIBuffer[23])	
HPI_Buf24	(uiHPIBuffer[24])	
HPI_Buf25	(uiHPIBuffer[25])	
HPI_Buf26	(uiHPIBuffer[26])	
HPI_Buf27	(uiHPIBuffer[27])	
HPI_Buf28	(uiHPIBuffer[28])	
HPI_Buf29	(uiHPIBuffer[29])	
HPI_Buf30	(uiHPIBuffer[30])	
HPI_Buf31	(uiHPIBuffer[31])	
DelayL	(uiHPIBuffer[32])	Delay Originalsignal L
	(uiHPIBuffer[34])	Delays für linken Kanal
	(uiHPIBuffer[54)	
	(uiHPIBuffer[56])	
DelayR	(uiHPIBuffer[33])	Delay Originalsignal R
	(uiHPIBuffer[35])	Delays für rechten Kanal
	(uiHPIBuffer[55])	
	(uiHPIBuffer[57])	
uiHPI_Phi	(uiHPIBuffer[58])	Winkel Originalsignal L
	(uiHPIBuffer[60])	Winkel der Spiegelquellen L
	(uiHPIBuffer[80])	
	(uiHPIBuffer[82])	
	(uiHPIBuffer[59])	Winkel Originalsignal R
	(uiHPIBuffer[61])	Winkel der Spiegelquellen L
	(uiHPIBuffer[81])	
	(uiHPIBuffer[83])	
uiFactorA	(uiHPIBuffer[84])	Reflexionsfaktor, Abschwächung aufgrund der Entfernung, Tiefpass der Wand
	(uiHPIBuffer[86])	
	(uiHPIBuffer[106])	
	(uiHPIBuffer[108])	
	(uiHPIBuffer[85])	
	(uiHPIBuffer[87])	
	(uiHPIBuffer[107])	
	(uiHPIBuffer[109])	
uiFactorB	(uiHPIBuffer[110])	
	(uiHPIBuffer[112])	
	(uiHPIBuffer[132])	
	(uiHPIBuffer[134])	
	(uiHPIBuffer[111])	
	(uiHPIBuffer[113])	
	(uiHPIBuffer[133])	
	(uiHPIBuffer[135])	

4.2 Implementierung des Algorithmus am DSP

4.2.1 Ringbuffer

Für den Ringbuffer wird ein Speicherbereich im Externen Speicher zugewiesen. Dies erfolgt wieder in der Datei `command.6lx`:

```
MEMORY
{
    CE0:        o = 80000000h   l = 00010000h
}

SECTIONS
{
    ringbuffer  >     CE0
}
```

In `TLV320AIC27_Driver.h` werden die beiden Variable `iRingbufferL` und `iRingbufferR` definiert:

```
#pragma DATA_SECTION(sRingbufferL, "ringbuffer")
short sRingbufferL[RING_SIZE];         //Ringbuffer for left channel
#pragma DATA_SECTION(sRingbufferR, "ringbuffer")
short sRingbufferR[RING_SIZE];         //ringbuffer for right channel
```

Beschrieben wird der Ringbuffer unmittelbar nachdem Eingangssignal vom Codec geholt wurde. `iCount` gibt immer die aktuelle Position in `iRingbufferL` und `iRingbufferR` an und wird bei Erreichen des Endes wieder auf Null gesetzt.

```
for(iX=0;iX<FRAMES_NUMBER;iX++)
{    // copy received samples to last half of input buffer
     // only the significant 16-bits will be used
   sInputL[iX] = (short)(uiPtrIn[CODEC_LEFT+iX*SLOTS_NUMBER]>>4);

   sInputR[iX] = (short)(uiPtrIn[CODEC_RIGHT+iX*SLOTS_NUMBER]>>4);

     // save the samples in the ringbuffer
   sRingbufferL[iCount]=sInputL[iX];
   sRingbufferR[iCount]=sInputR[iX];

   iCount++;

   if (iCount==RING_SIZE)
   {
     iCount=0;
   }
}
```

4.2.2 Berechnung im Hauptprogramm

Zuerst müssen aus der `HPI_Section` die Winkel und Multiplikationsfaktoren für alle Quellen eingelesen werden

```
for (iX=0;iX<iMIRROR;iX++)
{
  uiHPI_Phi[iX*2] = uiHPIBuffer[58+iX*2];
  uiHPI_Phi[iX*2+1] = uiHPIBuffer[59+iX*2];
  uiFactorA[iX*2] = uiHPIBuffer[84+2*iX];
  uiFactorA[iX*2+1] = uiHPIBuffer[85+2*iX];
  uiFactorB[iX*2] = uiHPIBuffer[110+2*iX];
  uiFactorB[iX*2+1] = uiHPIBuffer[111+2*iX];
}
```

Die `uiHPI_Phi` werden dann zur Berechnung der Ambisonic - Faktoren benötigt. Hier geht bereits durch `HPI_DeltaN` die Rotation in die Berechnung mitein.

```
for (iN=0;iN<iMIRROR*2;iN++)
{

  iX = (uiHPI_Phi[iN] + HPI_DeltaN) % 0x100;
  sFactorX_1[iN] = sCos[iX];
  sFactorY_1[iN] = sCos[(iX+64) & 0xFF];
  sFactorU_1[iN] = sCos[(iX*2) & 0xFF];
  sFactorV_1[iN] = sCos[(iX*2+64) & 0xFF];
  sFactorX3_1[iN] = sCos[(iX*3) & 0xFF];
  sFactorY3_1[iN] = sCos[(iX*3+64) & 0xFF];

}
```

Die Buffer für die Ambisonic - Kanäle werden zuerst mit den gewichteten und summierten Originalsamples gefüllt und somit für jedes Frame neu initialisiert.

```
for (iX=0;iX<FRAMES_NUMBER;iX++)
{
     // samples direct from the speakers
  lY = sInputL[iX];
  lY += sInputR[iX];
  sWBuffer[iX+FRAMES_NUMBER] = (sFactorW_1*lY)  >> 16;
  lY = sFactorX_1[0]*sInputL[iX];
  lY += sFactorX_1[1]*sInputR[iX];
  sXBuffer[iX+FRAMES_NUMBER] = lY  >> 16;
  lY = sFactorY_1[0]*sInputL[iX];
  lY += sFactorY_1[1]*sInputR[iX];
  sYBuffer[iX+FRAMES_NUMBER] = lY  >> 16;
  lY = sFactorU_1[0]*sInputL[iX];
  lY += sFactorU_1[1]*sInputR[iX];
  sUBuffer[iX+FRAMES_NUMBER] = lY  >> 16;
  lY = sFactorV_1[0]*sInputL[iX];
  lY += sFactorV_1[1]*sInputR[iX];
```

```
    sVBuffer[iX+FRAMES_NUMBER] = lY  >> 16;
    lY = sFactorX3_1[0]*sInputL[iX];
    lY += sFactorX3_1[1]*sInputR[iX];
    sX3Buffer[iX+FRAMES_NUMBER] = lY  >> 16;
    lY = sFactorY3_1[0]*sInputL[iX];
    lY += sFactorY3_1[1]*sInputR[iX];
    sY3Buffer[iX+FRAMES_NUMBER] = lY  >> 16;

  }
```

Anschließend werden je nach Wert von `iMirror` (Anzahl der zu rechnenden Spiegelquellen) die verzögerten Samples zur Berechnung herangezogen. `iCount` gibt dabei die momentane Position des Schreibpointer vom Ringbuffer an. Damit und mit dem Wert der jeweiligen Verzögerung wird die Ausleseposition im Ringbuffer bestimmt.

```
for (iN=2;iN<iMIRROR*2;iN++)
{
     // find position in ringbuffer
   iDelay = iCount + (RING_SIZE - (uiHPIBuffer[32+iN] * FRAMES_NUMBER));
   iDelay &= 0xfff;         // Delay = 0 to RING_SIZE-1

     // actual Factors for one mirror source
   iA = uiFactorA[iN];
   iB = uiFactorB[iN];
   iFacX = sFactorX_1[iN];
   iFacY = sFactorY_1[iN];
   iFacU = sFactorU_1[iN];
   iFacV = sFactorV_1[iN];
   iFacX3 = sFactorX3_1[iN];
   iFacY3 = sFactorY3_1[iN];
```

In der inneren Schleife über `FRAMES_NUMBER` werden dann die Samples mit den Faktoren multipliziert, der IIR - Filter berechnet und dann zu den Ambisonic - Buffern summiert.

```
   #pragma MUST_ITERATE (FRAMES_NUMBER, , 8);
   for(iX=0;iX<FRAMES_NUMBER;iX++)
   {    // calculate
     iTemp1 = (iA*iTemp) >>16;
     iTemp2 = sRingbufferL[iX+iDelay];
     iTemp2 = (iB*iTemp2 )>>16;
     iTemp = (iTemp1 + iTemp2);
     lW = sFactorW_1*iTemp;
     sWBuffer[iX+FRAMES_NUMBER] += lW  >> 16;
     lX = iFacX * iTemp;
     sXBuffer[iX+FRAMES_NUMBER] += lX >> 16;
     lY = iFacY * iTemp;
     sYBuffer[iX+FRAMES_NUMBER] += lY >> 16;
     lU = iFacU * iTemp;
     sUBuffer[iX+FRAMES_NUMBER] += lU >> 16;
     lV = iFacV * iTemp;
     sVBuffer[iX+FRAMES_NUMBER] += lV >> 16;
     lX3 = iFacX3 * iTemp;
```

```
        sX3Buffer[iX+FRAMES_NUMBER] += lX3 >> 16;
        lY3 = iFacY3 * iTemp;
        sY3Buffer[iX+FRAMES_NUMBER] += lY3 >> 16;
    }

}
```

4.2.3 Probleme

Der Zugriff auf den Speicher des DSP ist der zeitaufwendigste Teil, er dauert 5 Zyklen. Beim Programmieren muss versucht werden diese Zeit zu nützen und parallel bereits Berechnungen durchzuführen. Dies ist nicht immer möglich, was im Assemblerfile NOP – Einträge zur Folge hat und damit eine Wartezeit verursacht. Diese NOP – Einträge können noch durch Veränderungen von Schleifen, Berechnungsabfolge, etc. eventuell wegoptimiert werden.

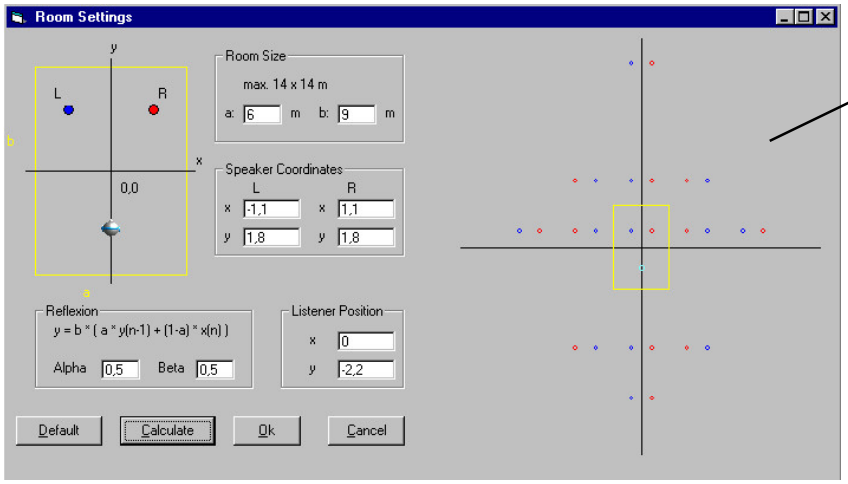
4.3 Berechnungen in Visual Basic

Im Visual Basic Programm werden für die weiteren Echtzeitberechnungen des DSP gewisse Daten vorab berechnet. So erfolgt nach der Eingabe der Raumkoordinaten, Lautsprecher- und Zuhörerpositionen und der Reflexions- und Filterfaktoren die gesamte Berechnung der Delays, Winkel und Faktoren für jede Spiegelquelle.
Die Abschwächung Aufgrund der Entfernung einer Spiegelquelle, der Tiefpassfilter und der Reflexionsfaktor werden zusammengefasst auf 2 Faktoren, die dem DSP zur Berechnung übergeben werden.
Mit dem `uiFactorA` wird das vorherige Ausgangssample y(n-1) multipliziert, mit `uiFactorB` das jeweils aktuelle Sample aus dem Ringbuffer.

5 Programm

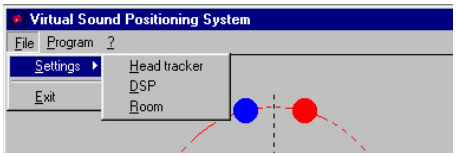
Änderungen zum bestehenden System.

Beim Programmstart erscheint neben dem Hauptfenster sofort das Fenster für das Raum – Setting. Mit **<Default>** können Standartwerte geladen werden, durch klicken von **<Calculate>** wird die Berechnung der Delays, Winkel und Faktoren gestartet und mit **<Ok>** wird das Fenster geschlossen , die Werte dem DSP übergeben und das Programm gestartet.



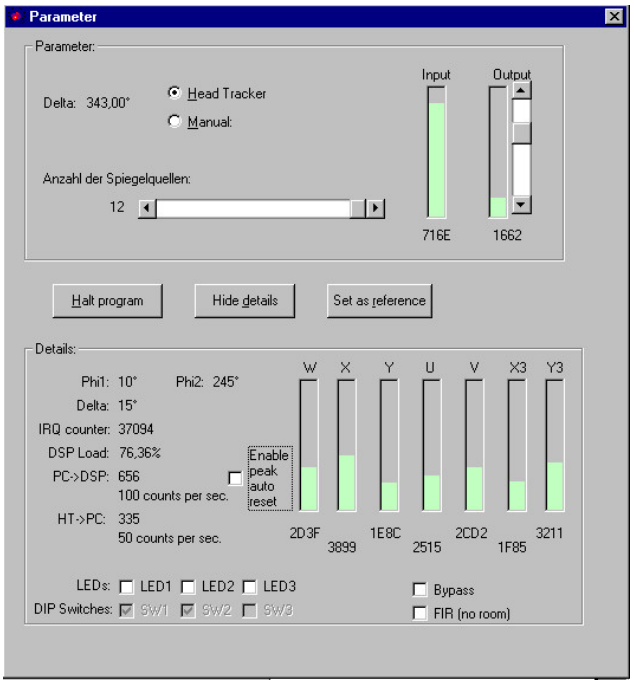
Graphische Anzeige der Verteilung der Spiegelquellen

Um die Raumeinstellungen zu ändern muss das Programm angehalten werden. Über den Menüpunkt **<File/Settings/Room>** kann obiges Fenster wieder geöffnet und einzelne Parameter verändert werden.
Wichtig: Neue Werte werden nur durch Klicken von **<Calculate>** übernommen.



Im Punkt **<Head Tracker>** kann der COM-Port für den Head Tracker ausgewählt werden. Bei **<DSP>** kann eine neue COFF-Datei (*.out) geladen werden.

Im Fenster Parameter kann mit einem Slider die Anzahl der zu rechnenden Spiegelquellen eingestellt werden. Bei Aktivierung des Kontrollkästchens **<FIR (no room)>** wird die Einbindung der Spiegelquellen gestoppt; dies entspricht dem Slider – Stellung links – 0 Spiegelquellen



- *Phi 1, Phi 2, Delta:*

Aktueller Winkel der Lautsprecher L/R relativ zur Hauptachse, sowie Winkel der Kopfdrehung.

- *IRQ Counter:*

Zähler für die Anzahl der durchgeführten Interrupts.

- *DSP Load:*

Momentane CPU Auslastung.

- *PC -> DSP:*

Datenaktualisierungsrate der Kommunikation des PC mit dem DSP.

- *HT -> PC*

Datenaktualisierungsrate der Kommunikation des HT mit dem PC.

- *DIP Switches:*

Abfrage der Stellung der DIP-Schalter des Evaluationboards.

- *LEDs:*

Die LEDs des Evaluationboards lassen sich setzen sowie rücksetzen.

- *Bypass:*

Das Audiosignal wird ohne Filterung an den Ausgang weitergeleitet.

- *FIR (no room)*

Es wird die komplette Raumsimulation weggeschaltet

- *Enable peak auto reset:*

Ist dieser Menüpunkt gewählt, werden die Spitzenwerte der jeweiligen Pegelanzeigen nicht
eingefroren.

- *Pegelanzeige:*

Die Pegel der einzelnen Ambisonic-Kanäle werden angezeigt.