ATRテクニカルレポート表紙

[公開]

TR-M-0050

剛球モデルに基づく遮音物体移動聴感の実時間生成

西村　竜一　　宮里　勉
Ryouichi NISHIMURA　　Tsutomu MIYASATO

2000.3.31

ATR 知能映像通信研究所

1 はじめに

より臨場感溢れる音場を再現したいという願いから，十数年来，3次元立体音響技術に関する研究が盛んに行われてきた．その基礎となる考え方は，空間中の任意の地点Aから鼓膜面 (あるいは外耳道入口) までの音響的な伝達関数を実測あるいはシミュレーションにより求め，それを無響室で録音した (ドライな) 音源に畳み込むことによって，その音源があたかも地点Aにあるかのように知覚させられるというものである [1]．ここで用いられる伝達関数は，頭部伝達関数 (HRTF: Head Related Tranfer Function) と呼ばれ，その効果的な測定手法に関する研究に始まり [2]，IIR フィルタによる近似手法 [3] や補間方法 [4] などについて，現在なお，精力的に研究が進められている．

これまでの3次元立体音響技術は，聴取される音から“音源”の位置を空間的に任意の位置で知覚させることで聴空間に広がりを持たせ，臨場感を与えるという点に主眼が置かれていた．視覚刺激の提示法が，映像をスクリーンあるいはディスプレイに映し出し，それを離れて見るという手法を用いる限りは，視覚的に頭の近傍に物体を知覚することが無いので，音の到来方向の再現が最も主要な要求項目となる．しかし，現在，IMAX theater や HMD のような，広視野角の立体視ディスプレイの登場により，正に眼前を移動する物体を視覚的に提示することが可能になるに至った．この場合，その物体による音の反射や回折に起因する音の変化は，十分に人の知覚閾内のものとなる．実際，視覚障害者は，これらの音の変化も利用して，障害物の衝突の回避を行っていることが調査されている [5]．報告によると，無響室において音源を障害物の反対側に置き，視覚物知覚の能力を有する盲人の被験者に障害物へ向かってゆっくり歩いてもらい，障害物の存在を感じたところで止まるように指示したところ，障害物のおよそ 1m 手前で停止することに成功した．この距離は，シミュレーションにより求まった，遮音効果が顕著に現れる距離と一致することから，人間には遮音による音の変化で物体の存在を知覚できる能力が備わっているものと判断することができる．

マルチモーダルな刺激提示は，没入感を高めるのに効果的であるが，各感覚系間で刺激に矛盾がある場合には，かえって現実感を損なわせる要因になりかねない．視覚系と聴覚系の間の関係を例に見てみると，視覚刺激から判断される空間的な位置と聴覚刺激から判断される空間的な位置の相違は，ある程度までは「腹話術効果」により音源位置が視覚情報から判断される位置へ補正されて知覚される．しかし，それよりも離れてしまうと別の物体から発せられている音として知覚されることになる [6]．また，言語音知覚における同様の現象に関しては,「マガーク効果」が有名である [7]．これは，「バ」と発音している口の映像を提示し，それに合わせて「ガ」の発音を聴覚刺激として提示すると，「バ」でも「ガ」でもない「ダ」の音として知覚されるというものである．視覚刺激から予測される音と実際に聴取された音が合致していないために，提示している音刺激とは事なる別の音として知覚されるという現象が起きている．これらの事例から予見されるように，視覚的に眼前に物体がある場合には，その場合に聞こえるべき聴覚刺激を再現する必要がある．もし，物体が音を発している場合には，反射や回折による音の変化よりも，音源位置が変化することによる音の変化のほうが影響力が大きいため，従来の3次元立体音響技術を用いることで対応が可能だと考えられる．しかし，反射や回折といった現象は，音を発していない物体に対して起こる現象であり，この場合には，従来の3次元立体音響技術による音源位置の制御だけでは十分とは言い難い．

仮想的に幾つもの部屋がドアで連結された空間を生成し，その中のある地点で音が発

せられている時に，この仮想空間内を自由歩行するに従って，そこで聞こえるべき音を提示するシステムを構築する研究が行われている [8, 9]．しかし，計算速度等の問題により実時間で動作させるために，壁での鏡面反射を対象とし，回折現象は再現していない．この手法は，壁に囲まれた空間の音場を再現するというような場面では効果的だが，屋外のように音を反射する物体までの距離は遠いが，音を遮蔽する物体は聴取者の比較的近傍を通過するというような場合には，効果が期待できない．しかし，このような状況は，屋外において車や他の通行人が傍を通り抜けるような場面で，しばしば遭遇するものである．

そこで，音を遮る物体が聴取者の近傍にあり，その周辺には，他に音の聴取に強く影響を与えるような反射壁が存在しない場合を考える．この場合，2 次反射や 2 次回折を無視することでモデルは極めて簡単になり，物体の形状を剛球と仮定することで，容易に解析解を得ることができる [10]．高い仮想現実感を実現するためには，インタラクティブ性は欠かすことのできない要素であり，その意味において実時間動作は，仮想現実感関連技術にとって重要度の高い要求事項である．そこで，本研究では，この条件を満足しつつ，物体が近傍を通過した音感をユーザーに提示することが可能なシステムの開発を行った．このシステムにより，仮想環境において眼前を物体が通り抜けた時に，それに対応する反射や回折も考慮した音を実時間で合成して提示することができるようになる．また，これまでの視覚刺激偏重の仮想空間提示方法では，ディスプレイに映し出されていない地点における情報をユーザーが知覚することは不可能であったため，ディスプレイの端から物体が現れると突然それが出現したかのような印象を与えるという問題があった．しかし，このシステムを使うことにより，視野内外の物体の移動をシームレスに知覚させることが可能になるものと期待される．

2 剛球モデルによる HRTF のシミュレーション

図 1 に剛球モデルの模式図を示す．剛球モデルでは，式 (1) で表される直接波の速度ポテンシャル ϕ_i と式 (2) で表される剛球からの反射波の速度ポテンシャル ϕ_r の和として，ある地点におけるポテンシャルが求められる [10]．

$$\phi_i = \phi_0 \sum_{n=0,1,2,\cdots}^{\infty} j^n (2n+1) j_n(kr) P_n(\cos\theta) e^{j\omega t} \tag{1}$$

$$\phi_r = \sum_{n=0,1,2,\cdots}^{\infty} a_n h_n^{(2)}(kr) P_n(\cos\theta) e^{j\omega t} \tag{2}$$

$$a_n = -\frac{j^n (2n+1) j_n'(x)}{h_n^{(2)'}(x)} \phi_0, \qquad x \equiv ka \tag{3}$$

ここで，P_n は Legendre 関数，j_n は n 次の球 Bessel 関数，$h_n^{(2)}$ は n 次の第 2 種球 Hankel 関数，ϕ_0 は $z = 0$ での速度ポテンシャルを表す．

MATLAB には，Legendre 関数や球 Bessel 関数，Hankel 関数などの特殊関数も用意されているので，シミュレーションには MATLAB を使用した．ただし，現在の MATLAB(Ver. 5.3) で用意されているこれらの関数には次数に制限があり，256 次が最大と

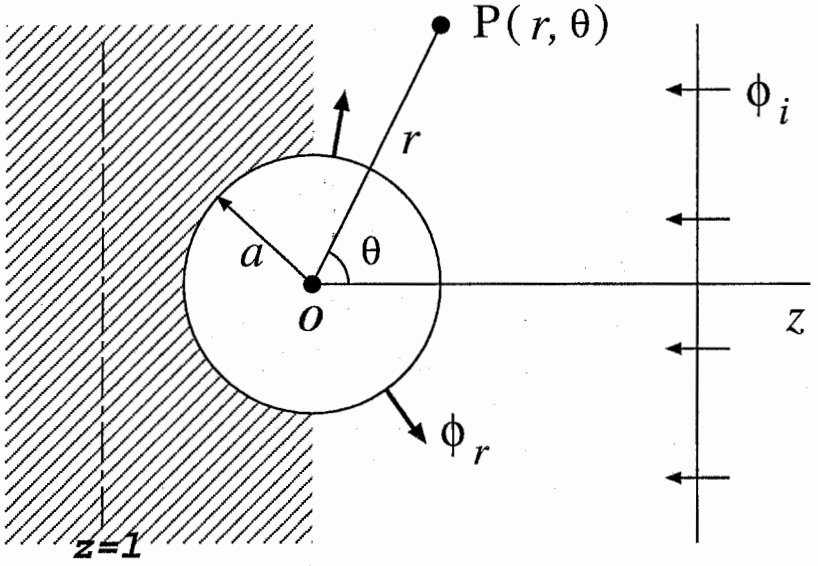


図 1: 剛球モデルによる音場シミュレーション

なっている．剛球が小さく，観測点も剛球の近傍に位置する場合には，比較的低次までの級数でよく近似できるが，剛球が大きく観測点が剛球から遠ざかると 256 次まででは，十分な近似ができなくなる．そこで，ここでは剛球の半径を 8cm とし，観測点も剛球の比較的近傍へ設定することにして，この問題を回避することにした．

図 1 における直線 $z = l$ に沿って求めた伝達関数の大きさを，3 次元表示で図示したものが図 2 である．この図から，音の到来方向から見て剛球の後方部では，高い周波数成分が連続的に減少している様子がよく再現されていることが伺える．これは，前を障害物が横切ると音がこもって聞こえるという日常的に経験する現象とよく合致している．また，剛球表面を観測点とすることにより剛球を人の頭と見立てて，左右耳の位置での伝達関数から，剛球モデルを頭部伝達関数の概形のシミュレーションに利用した例もある [11]．物体の近似形として剛球モデルを用いる場合の問題点は，聴取地点と音源位置および物体が一直線上に並んだ時である．この場合，剛球モデルでは完全な球形と球面上での全反射を仮定しているため，観測点における伝達関数が 1 になるという現象が起こる．このため，この伝達関数をそのまま利用すると，物体が顔の前を通り抜ける際に，正に真正面を通過する瞬間，音が突然大きくなってしまう．しかし，実在の物体で完全な球形を有しているものは殆ど存在しない．加えて，音響信号を全反射するという条件も完全に満たすものは，現実的には考え難い．そこで，この非現実的な現象を回避するために，ここでは剛球後方部に生じる伝達関数が上昇している部分を便宜的に削除し，この区間をスプライン補間で滑らかに結ぶことにした．こうすることにより，図 2 に示される伝達関数は，図 3 に示す伝達関数分布に置き換えられることになる．

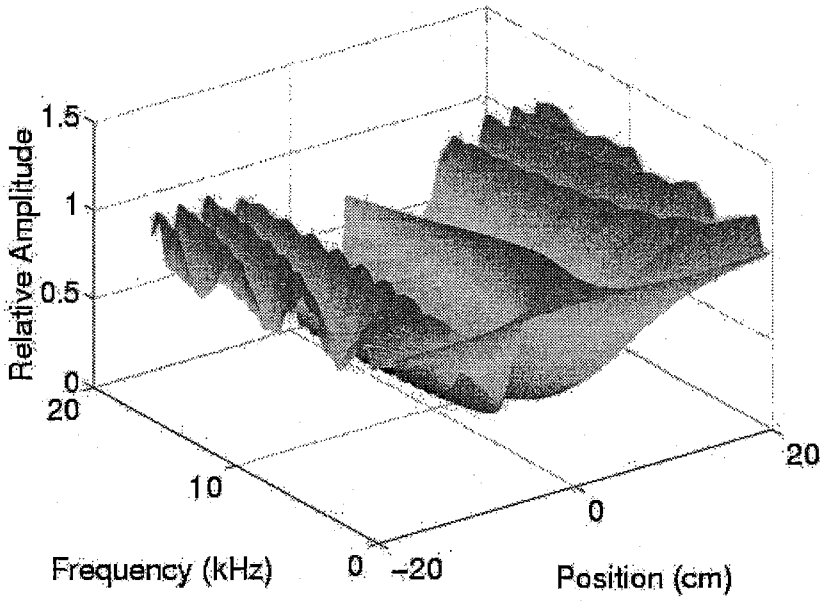


図 2: 伝達関数 (振幅) の分布 ($a = 8$cm，剛球の中心からの距離 10cm)

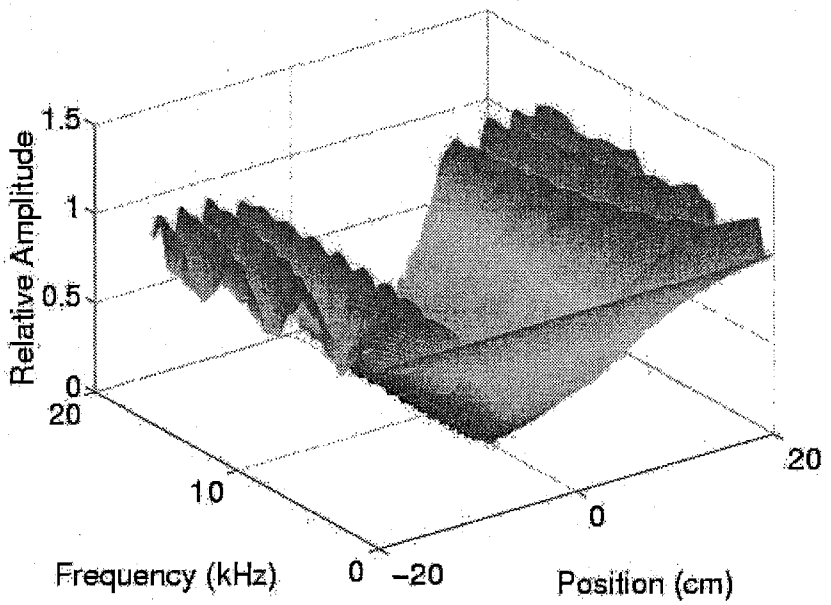


図 3: 剛球後方の丘陵部を削除し，スプライン関数で補間した伝達関数 (振幅) の分布 ($a = 8$cm，剛球の中心からの距離 10cm)

3 Karhunen-Loéve展開による省略化

剛球モデルを用いて空間上の全ての地点における HRTF をシミュレーションにより求め，これをハードディスク上に保持しておき，その時々の両耳の位置に対応する HRTF をその都度読み出して原音声信号に畳み込むことで，物陰の音場を生成することができる．しかし，物体と聴取者の全ての位置関係に対応する HRTF を保持しておくことは，大きな記憶容量を必要とするばかりでなく，補間処理を行う際の計算量も増加する．もし，計算速度の観点からも，必要となる容量の観点からも低コストで同等の効果が実現できれば，応用範囲が広がる可能性が高い．特に，インタラクティブ性により仮想現実感を出しているアプリケーションにおいては，実時間での動作可能性は必要不可欠な要求項目である．そこで本章では，前節で述べた剛球モデルを用いたシミュレーションにより求めた HRTF を，少ないデータから実時間で再現することを試みる．

直交変換の中で理論的に最も圧縮効率が高い変換手法は，Karhunen-Loéve 展開であることが知られている [12]．このことから，Karhunen-Loéve 展開は，画像データの圧縮や特徴抽出 [13] などの分野において広く利用されている．Karhunen-Loéve 展開は，ある N 次元ベクトル信号 x_i に対してその共分散行列

$$A = \frac{1}{P}\sum_{i=0}^{P-1} x_i x_i^T \tag{4}$$

を求め，この共分散行列を固有値分解して得られた固有ベクトル ϕ_i を基底関数として，

$$x_i = \sum_{j=1}^{N} p_{ij}\phi_j \tag{5}$$

のように展開する手法である．ここで，係数 p_{ij} は，対応する固有ベクトルと信号 x_i の内積により求まる．各固有値の値が，それに対応する固有ベクトル (基底関数) へ信号 x_i を写像した時のエネルギーの期待値と等価になることから，ϕ_j をそれに対応する固有値の大きい順に並び変え，適当な値 $K(<N)$ までを利用することにより，原信号と完全には一致しないものの十分に近い信号を再構成することができる．

$$x_i \approx \sum_{j=1}^{K} p_{ij}\phi_j \tag{6}$$

図 4 は，図 3 に示した伝達関数分布の自己相関行列を生成し，それを固有値分解した時の固有値分布の一部を拡大表示したものである．この図から，大きな値を持つ固有値が少数の固有ベクトルに集中していることが分かる．したがって，剛球モデルにおいて球後方の音の到来方向と直交する直線上の地点における伝達関数は，少数の固有ベクトルの線形和で良く再構成されることが予想される．そこで，この事を確認するために，得られた固有ベクトルを元にして伝達関数を再構成した時の残差パワーを求め，これを図 5 に示した．図 5 から，固有ベクトルをおよそ 4 つ程度まで使用することにより，概ね良好な再構成が実現できることが確認された．また，その時の誤差分布を図示したものが，図 6 である．ここで誤差は，再構成した伝達関数を $x(f,p)$，元の伝達関数を $x_o(f,p)$ として，

$$\mathrm{Error}(f,p) = \left|\frac{x(f,p) - x_o(f,p)}{x_o(f,p)}\right| \tag{7}$$

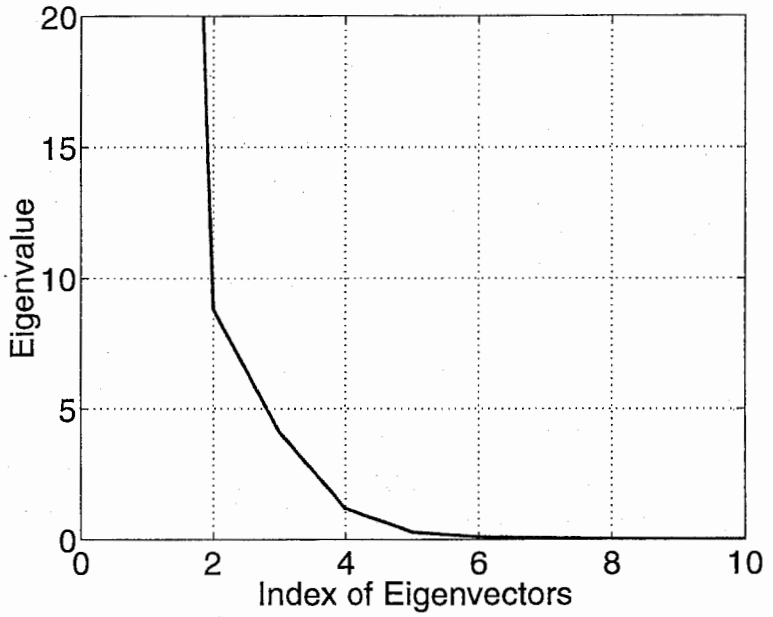


図 4: 図 3 に示した伝達関数分布の自己相関行列を固有値分解したときの固有値分布 (大きな固有値から 10 番目までを拡大表示)

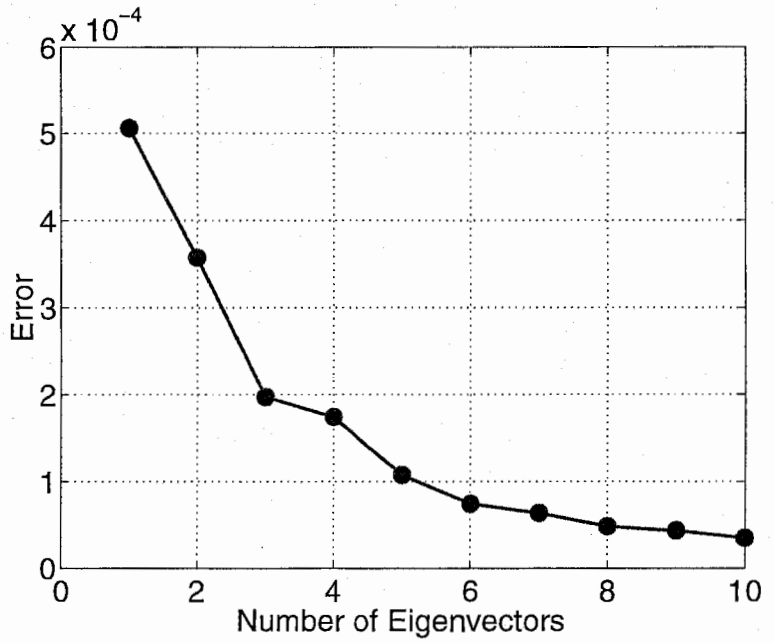


図 5: 再構成時に使用した固有ベクトルの数と平均残差パワーの関係

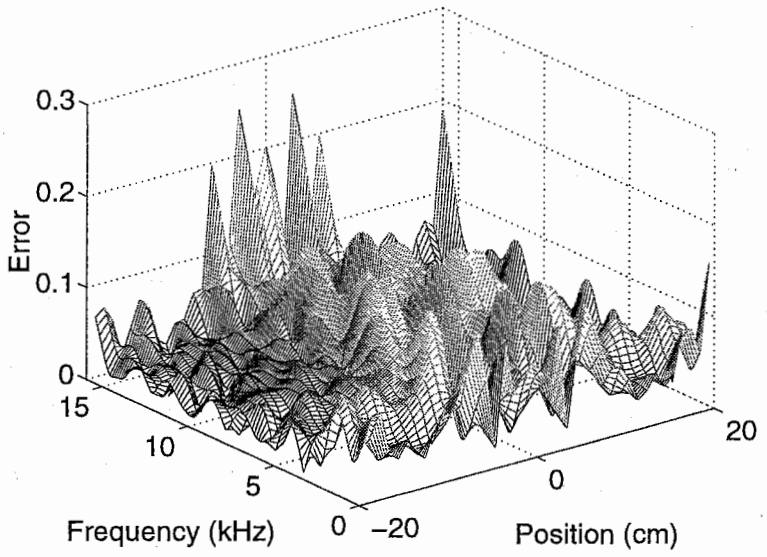


図 6: 4 つの固有ベクトルを用いて伝達関数を再構成した時の誤差分布

により求めたものである．図 6 を見ると，高い周波数で比較的大きな誤差が生じているところがあるのが分かる．剛球の真後から遠い所では，周波数特性がほぼ平坦であり，低い周波数成分の音も多く伝搬される．その結果，マスキング現象により高い周波数成分の音の聴取は困難になってくるため，この誤差は，あまり大きな影響を及ぼさないものと判断される．また，剛球の真後付近で誤差が大きくなっているのは，元々の伝達関数の値が小さいために，(7) の分母が小さくなり，見掛け上，誤差が大きくなっているのが原因だと考えられる．

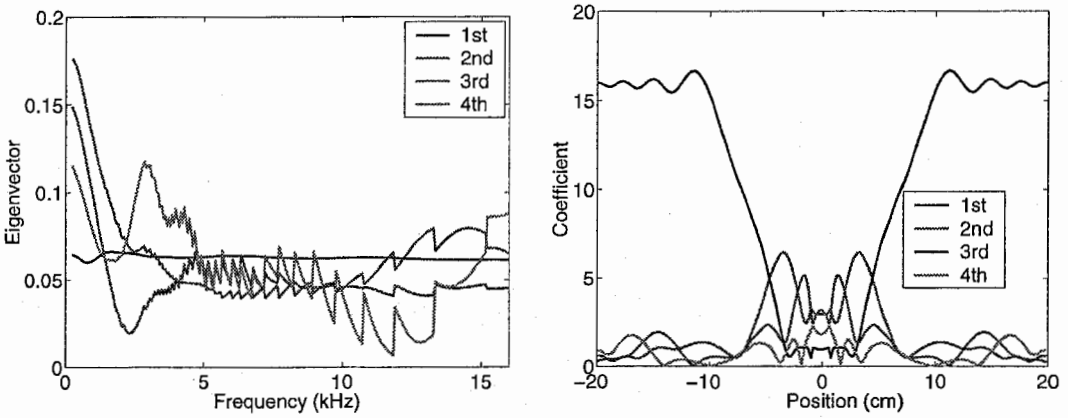
大きなほうから 4 つの固有値に対応する固有ベクトルまでを用いて，元の伝達関数分布を再構成する際の固有ベクトルおよび係数分布は，それぞれ，図 7(a) および図 7(b) のようになる．これ以降，これらのデータを用いて物陰の音場を再現し，ユーザーに提示する手法を **RSSC** (Real-Time Shaded Sound Creator) と呼ぶことにする．

4 システム構成

前節における検討により，音源に対して剛球後方に位置する聴取点での伝達関数が，少数の固有ベクトルを基にして概ね良好に再構成可能であることが示された．したがって，物陰の音感を生成するためには，無響室で録音された音源に対して，この伝達関数を畳み込めばよいことになる．

これを実現するシステムとしては，種々の構成が考えられるが，ここではフーリエ変換の実現方法が異なるふたつを例として示すことにする．

FFT を用いる手法 実時間動作を実現するにあたって，最も障害となることが予想されるのは音源信号と伝達関数の畳み込み演算である．この手法では，線形畳み込み演算を DFT を用いて，重畳加算法あるいは重畳保留法で実現することになる．この場合，正確に線形畳み込みを実現するためには，円状畳み込みに起因するエリアジ



(a) 固有ベクトルの形状　　(b) 再構成の際の重み係数

図 7: 大きな 4 つの固有値に対応する固有ベクトルからの再構成

ングを避けるために，時間軸上でそれぞれの信号の後ろに零を付加した上でフーリエ変換を行わなければならない．しかし，本手法では，伝達関数が既に周波数軸上で供給される上，実時間で処理を終えなければならないという必須要件がある．そこで，完全な線形畳み込みの実現を断念し，50 %オーバラップのハミング窓によるフレーム処理を行うことで，エリアジングを抑制するに留めることにする．この手法によるシステムの実際的な構成については，巻末の付録で詳しく述べる．

FIR フィルタを用いる手法 図 7 に示した固有ベクトルを FIR フィルタで実現して，物陰の音場を生成するシステムを構築することも可能である．この場合のシステム構成を，図 8 に示す．ここで，所望の特性を実現するために必要となる各 FIR フィルタのタップ長を，AIC [14] を元に検討する．ここで，AIC の計算には，推定誤差が正規分布に従うという仮定を利用した．AIC の変化を示す図 9(a)，および，各次数における残差を図示した図 9(b) から，図 7 に示した固有ベクトルは，64 タップ程度の FIR で概ね構成できることが判明した．したがって，この FIR フィルタは，現実的なタップ長で実現することが可能であり，このシステムは実現可能なものだと判断される．

これまでの検討から，剛球モデルにより求まった伝達関数を再構成し，原音源信号と畳み込んで実時間でユーザーへ提示することが可能であることが判明した．そこで，次節では，本システムの達成すべき目的である，物陰の音場の “感覚” の生成という観点から，聴取実験を通して本システムの性能の検討を行うことにする．

5　聴覚刺激のみを提示した場合の移動方向弁別

前節において，本手法により物理的には十分小さな残差で，シミュレーションにより得られた伝達関数を再現できることが示された．しかし，まだ次に示す 2 つの問題点が残されている．

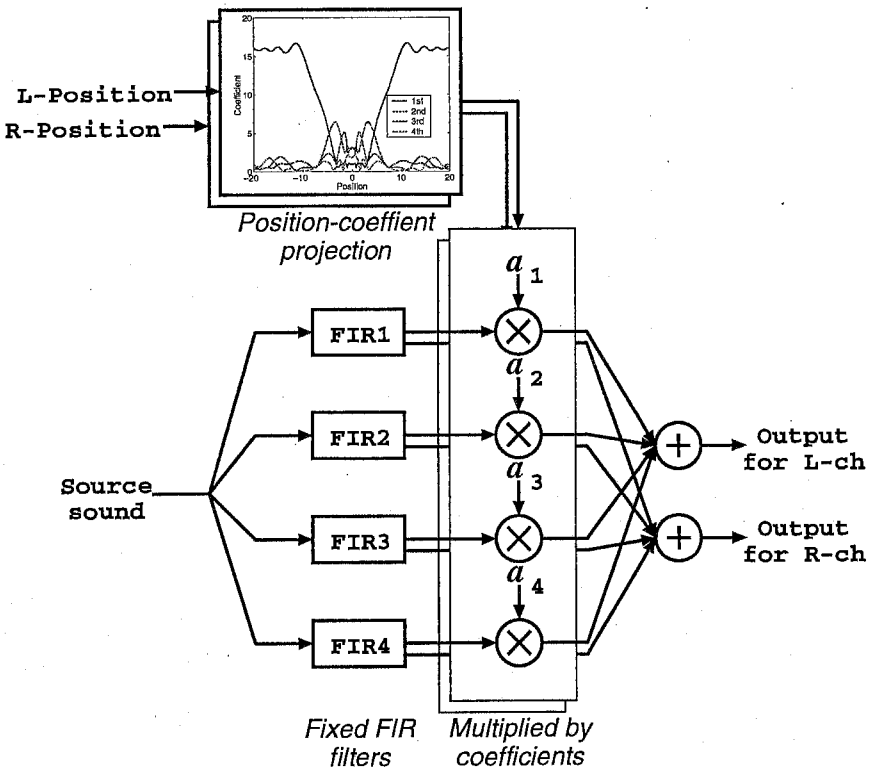
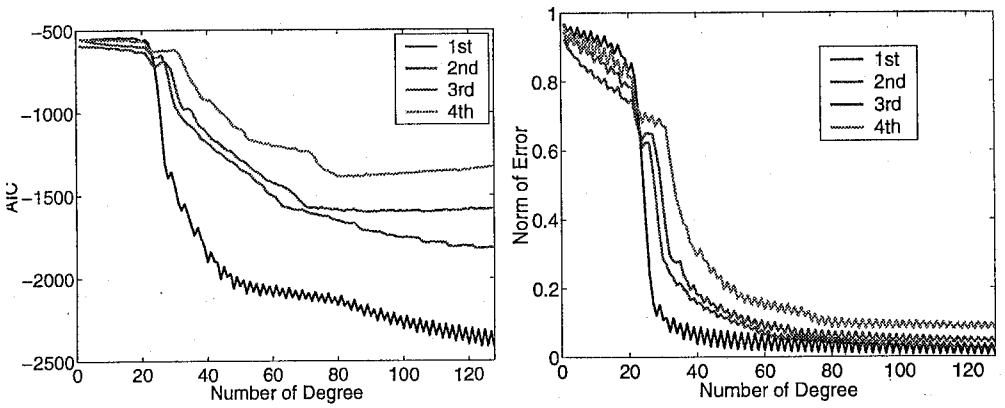


図 8: 4つの固有値から伝達関数を再構成し，実時間で物陰の音場を再現するシステムの構成図の例



(a) AIC の変化　　(b) 各次数の時の残差

図 9: AIC による最適な次数の検討

表 1: 実験に使用した刺激音の種類

番号	使用している HRTF の導出方法
1	剛球モデルでのシミュレーション結果を使用
2	上記のパワー変化のみを再現
3	1 次の係数まで使用した RSSC
4	2 次の係数まで使用した RSSC
5	3 次の係数まで使用した RSSC
6	4 次の係数まで使用した RSSC

1. 剛球モデルのシミュレーションにより得られた音場で，人が実際に聞いた時の物陰の音の感覚が生成されているのか．
2. 周波数特性までを厳密に再現しなくても，音響パワーの変化を再現するだけで十分なのではないか．

そこで，RSSC により物陰の音場を生成した場合に，知覚させたい物陰の音感を人間が知覚しているのかどうかを被験者実験により検討した．実験では，本手法の実際的な利用形態を考慮して，仮想的な遮蔽物の移動方向の弁別という観点から検証することにする．刺激音は，遮蔽物に対する両耳それぞれの聴取位置を，剛球モデルにおける剛球に対する観測点に対応づけて，観測点をずらしながら左右耳にそれぞれの音を提示することにより実現した．実験に使用した刺激音を，表 1 に示す．

7 名の被験者に対し，[移動方向 (左 → 右，右 → 左)，刺激音] の全ての組み合わせ (12 通り) をランダムにヘッドホン (STAX Lambda-pro) を通して提示し，被験者に音源の移動方向を回答させた．これを 1 セッションとし，1 被験者につき 10 セッションを行うことによって得られた正答率を図 10 に示す．図 10 から，被験者を 4 つのグループに分けることができる．

G1: パワーの変化を手掛かりとする被験者 – No.2

G2: 周波数特性の変化を手掛かりとする被験者 – No.7

G3: 両方を手掛かりとする被験者 – No.4

G4: 聞き分けられない被験者 – No.1，3，5，6

この結果から，RSSC により合成した音響情報を物体の移動方向の知覚に有効に活用できている人もいるが，G4 に分類された被験者のように，あまり活用できていない人もいることが判明した．ただし，G4 に分類された被験者でも，3 次の係数までを使用した RSSC に対しては，比較的高い正答率を示すようである．

6 視覚刺激と聴覚刺激を同時提示した場合の移動方向弁別

立体視ディスプレイを用いた映画のための音響刺激製作や 仮想現実感関係への利用など，実際の応用の場面では，視覚刺激も同時に提示される場合が多いことが予想される．

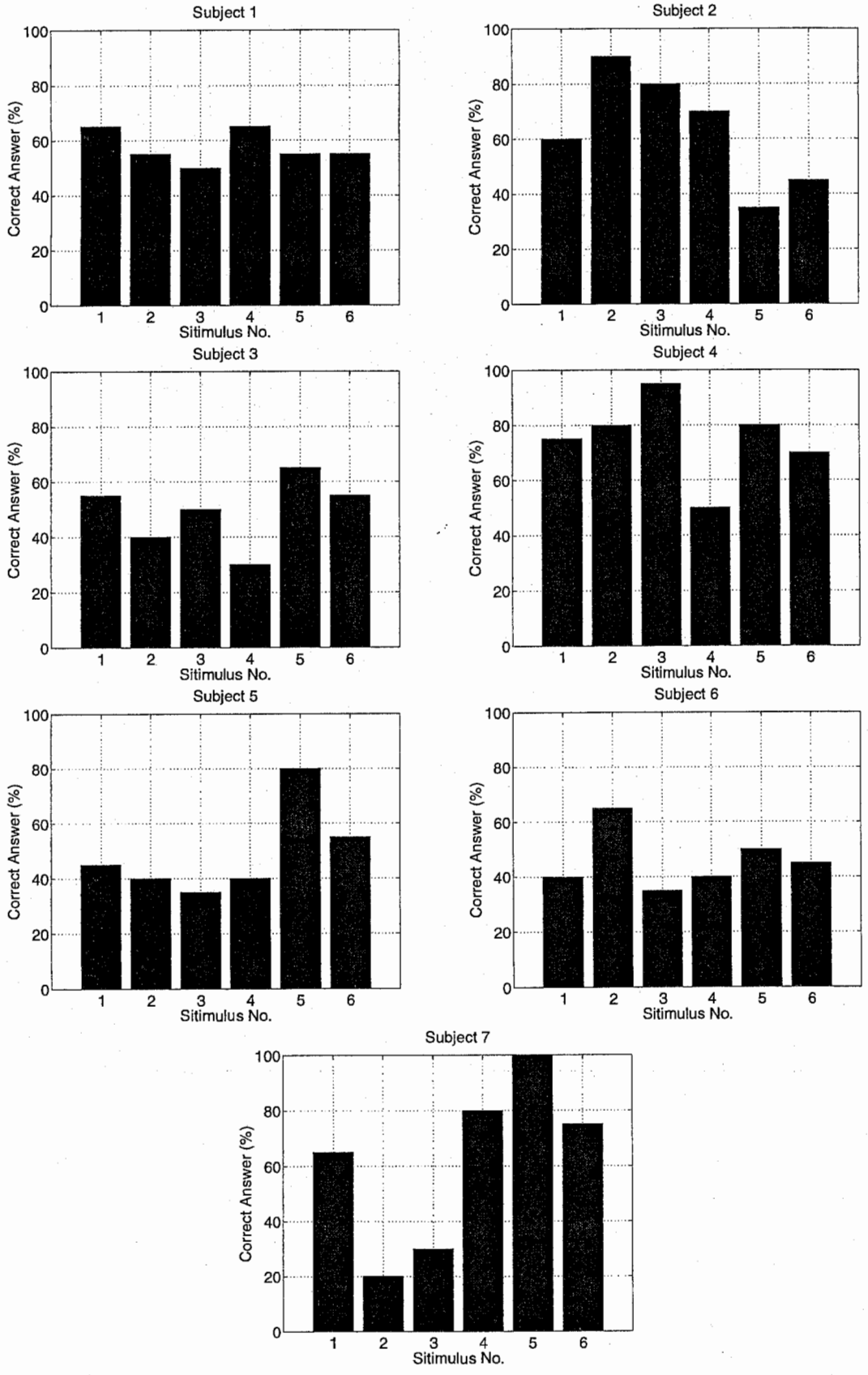


図 10: 聴覚刺激のみによる物体の移動方向弁別実験の結果

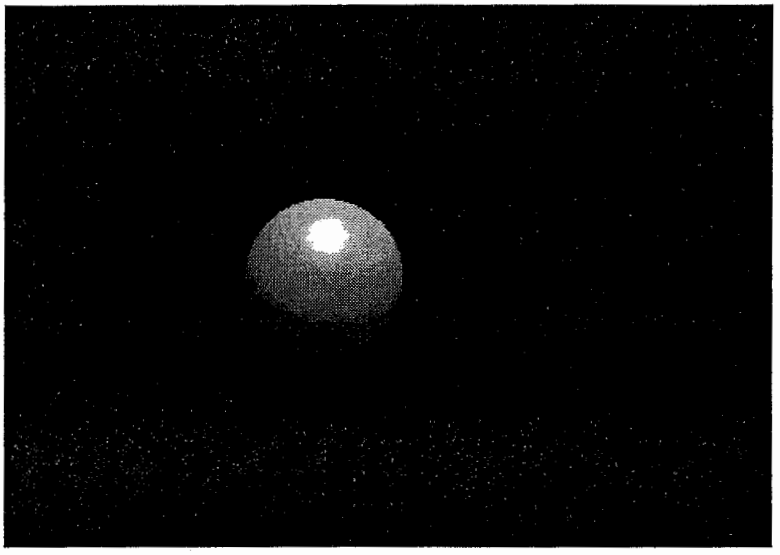


図 11: 視覚刺激として被験者に提示した画像

表 2: 視覚刺激と同時提示した場合の移動方向弁別実験に使用した聴覚刺激

刺激名	使用している HRTF の導出方法
Orig.	剛球モデルのシミュレーション結果
Power	上記の音響パワー変化のみを再現
Deg.1	1 次の係数までを利用した RSSC
Deg.3	3 次の係数までを利用した RSSC

そこで，視覚刺激も同時に提示されたときの弁別に関する実験も行った．これは，視覚刺激を提示することによって音の変化を被験者に予想させ，それと実際の聴刺激との比較を行うことで判別することにより，聴覚刺激のみを提示した場合と比較して，異なる結果が生じる可能性があると考えたからである．被験者には，「画面上の球体の移動方向と音から判断される遮蔽物体の移動方向が一致しているか否かを判断せよ」という課題を与えて，「はい」，「いいえ」の 2 肢強制選択で判断を行わせた．ここで視覚刺激には，図 11 に示すように，黒い背景の中を青いボールが音像の移動に合わせて移動する映像を用いた．映像は，B5 版ノートパソコンのディスプレイにある XVGA の 11 インチ TFT で提示し，音刺激は，ヘッドホン (STAX Lambda-pro) を通して被験者に提示した．実験に参加した被験者は，前節で行った実験に参加した被験者と同じ被験者である．各被験者が，全ての組み合わせの比較を行うとともに，順序効果を打ち消すために提示順序を入れ換えた刺激対についても回答を行わせた．表 2 に示す刺激音に対して行った実験結果を，図 12 に示す．図中の説明文では，視覚刺激を (V)，聴覚刺激を (A) と略記している．

図 12 において，各領域はそれぞれ，

(a) RSSC で生成した物体の移動方向と画面上の球体の移動方向が一致している時に「一

致」という回答が得られた割合
(b) RSSC で生成した物体の移動方向と画面上の球体の移動方向が逆の時に「不一致」という回答が得られた割合
(c) RSSC で生成した物体の移動方向と画面上の球体の移動方向が一致している時に「不一致」という回答が得られた割合
(d) RSSC で生成した物体の移動方向と画面上の球体の移動方向が逆の時に「一致」という回答が得られた割合

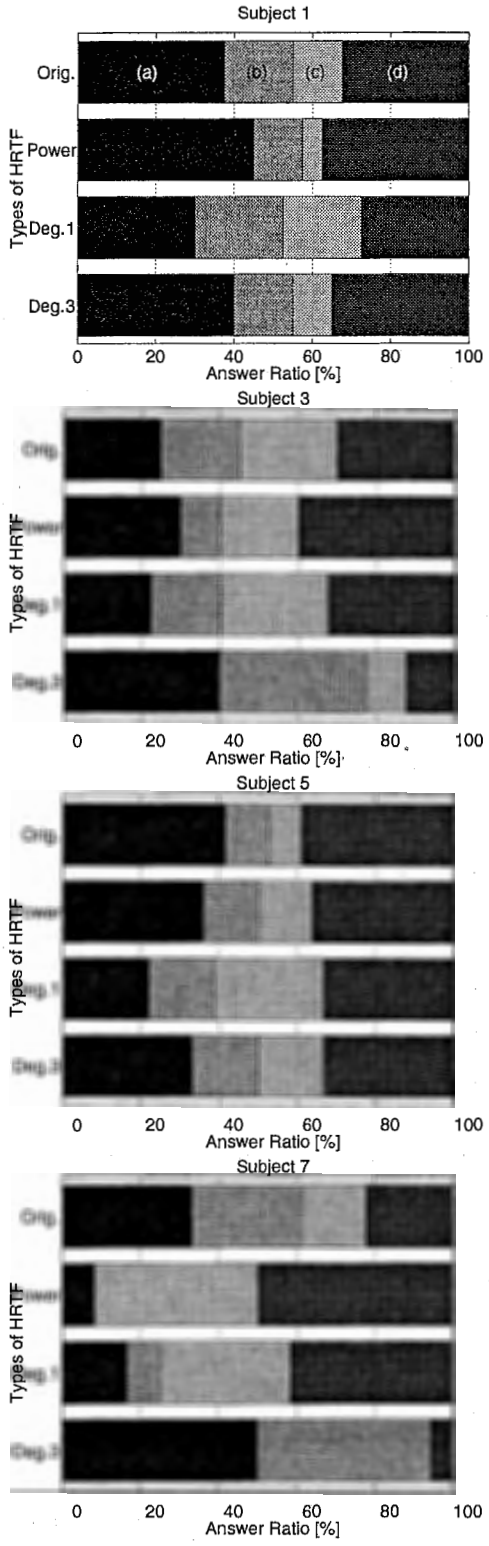
を表している．したがって，(a) と (b) を合計した領域が占める割合が正答率に対応する．この図 12 と聴覚刺激のみで判断を行わせた場合の実験結果である図 10 とを比較して判断すると，前節の実験で RSSC の効果が確認された被験者は，概ね，本実験でも高い正答率を示している．その一方で，前節の実験で RSSC により合成した音響情報をうまく解釈できないと判断された被験者は，視覚刺激を同時に提示しても，やはり同じ結果であった．

7 考察

そもそも，自ら音を放っているのではなく，音を遮っているだけの物体の移動は，雰囲気として捉えることができる程度のものであり，確信を持って知覚されるものでは必ずしもない．したがって，実験での正答率も 100%に近ければよいというものではなく，むしろ，50%よりも若干高い値に納まるのが，仮想現実感の生成における現実感の追及という観点からは好ましいと考えられる．その意味では，被験者によって正答率が高すぎる場合もあるが，これは，実験ということで特に意識を集中して判断を行っていたことが原因であると思われる．

被験者実験の結果，必ずしも全ての被験者に対して RSSC が有効というわけではなかったが，3 次までの固有ベクトルを用いて物陰の音場の伝達特性を再現しておけば，比較的広い範囲のユーザーに概ね対応できることが判明した．図 7(a) に示した固有ベクトルの形状から分かるように，1 次の固有ベクトルだけでは，パワーの変化だけしか再現することができない．一方，次数を増やして行くと，細かな周波数特性までも再現することができるようになってくる．人間は，パワーの変化も周波数特性の変化も，両方とも物体移動の知覚に利用できるため，それらの特徴の両方が共に残るような伝達関数が広いユーザーに対して有効であると考えられる．したがって，3 次までの固有ベクトルを用いて再構成した伝達関数が，正にそのような特徴を持つものになっているのかもしれない．

視覚刺激を同時に提示した場合の実験で，一部の被験者で，(a) および (d) の占める割合が大きくなっている．これは，人間の情報処理過程における視覚の優位性に起因するもので，聴覚刺激から判断される確信度の低い物体の移動情報を，より確信度の高い視覚刺激から判断される移動情報に結びつけて判断を行っているためと考えられる．この現象は，RSSC が有効でない被験者に対して顕著なことから，これらの被験者が，RSSC により合成された音響刺激を移動方向の弁別に活用できていない証拠でもあると判断される．



(a): V と A の移動方向が一致 &
「一致」の反応
(b): V と A の移動方向が不一致 &
「不一致」の反応
(c): V と A の移動方向が一致 &
「不一致」の反応
(d): V と A の移動方向が不一致 &
「一致」の反応

図 12: 聴覚刺激から判断される物体の移動方向と，視覚刺激から判断される物体の移動方向の一致/不一致を判別する実験の実験結果

8 結論

剛球モデルによりシミュレーションを行って得られた伝達関数を，Karhunen-Loéve展開した上で，少数のベクトルから再構成することで，物陰の音感を実時間で合成する手法(RSSC)を提案した．本手法を用いることにより，視覚刺激が無くとも音場中を音を発さない物体が移動する時の物体の移動方向を判別できることが判明した．本手法は，ヘッドマウントディスプレイを使用する 3D ビデオゲームでの挿入音の作成や，IMAX theatre のような 3D シアターで上映される映画の音響効果で利用することができる．RSSC を利用することにより，これまでの視覚偏重の仮想空間では無の世界であった視野外も含めて仮想環境の臨場感を高めることができるものと期待される．

参考文献

[1] 岡部馨, “ダミーヘッドを用いた音場再生,” 日本音響学会論文誌, Vol. 46, No. 8, pp. 650–656, August 1990.

[2] Yôiti Suzuki, Futoshi Asano, Hack-Yoon Kim, and Toshio Sone, “An optimum computer-generated pulse signal suitable for the measurement of very long impulse responses,” *J. Acoust. Soc. Am.*, Vol. 97, No. 2, pp. 1119–1123, February 1995.

[3] 春日正男, 目加田慶人, 長谷川光司, 安田晴剛, “IIR フィルタによる頭部伝達関数の近似方法,” 日本音響学会論文誌, Vol. 54, No. 7, pp. 482–488, July 1998.

[4] 西野隆典, 梶田将司, 武田一哉, 板倉文忠, “水平面上の頭部伝達関数の補間,” 日本音響学会論文誌, Vol. 55, No. 2, pp. 91–99, February 1999.

[5] 関喜一, 伊福部達, 田中良広, “盲人の障害物知覚における障害物の遮音効果の影響,” 日本音響学会誌, Vol. 50, No. 5, pp. 382–385, May 1994.

[6] Charles E. Jack and Willard R. Thurlow, “Effects of Degree of Visual Association and Angle of Displacement on the “Ventriloquism” Effect,” *Perceptual and Motor Skills*, Vol. 37, pp. 967–979, 1973.

[7] 日本音響学会（編）, 音のなんでも小事典, 講談社, 1996.

[8] Thomas Funkhouser, Ingrid Carlbom, Gary Elko, Gopal Pingali, Mohan Sondhi, and Him West, “A Beam Tracing Approach to Acoustic Modeling for Interactive Virtual Environments,” In *SIGGRAPH*, Vol. 23, pp. 48–64, 1999.

[9] 小宮山, 摂池沢龍, 大久保洋幸, 大谷眞道, 小野一穂, 正岡顕一郎, 浅山宏, “映像と連動した 3 次元音響のリアルタイム生成の検討,” 音響学会講演論文集, pp. 607–608, 9 1999.

[10] 早坂寿雄, 技術者のための音響工学, 丸善, 1986.

[11] 浜田晴夫 “第 58 回技術講習会資料,” Technical report, 日本音響学会, 1998.

[12] 高木幹雄, 下田陽久, 画像解析ハンドブック, 東京大学出版会, 1991.

[13] Dibyendu Nandy and Jezekiel Ben-Arie, "Generalized Feature Extraction Using Expansion Matching," *IEEE Transactions on Image Processing*, Vol. 8, No. 1, pp. 22–32, January 1999.

[14] 佐和隆光, 回帰分析, 朝倉書店, 1979.

付録

A S-RTP station 用プログラム

S-RTP station には，Texas Instruments 社の DSP ボード CPCI-DSP46701 が 1 枚搭載されている．このボード 1 枚に，DSP TMS320C6201 が 4 つ載っている．この内，A/D，D/A の回路に接続されているのは，DSP-A のひとつのみであるので，A/D，D/A 関係のプログラムに関してはこの DSP を使用しなければならない．しかし，1 つの DSP の計算処理能力 (約 1GFlops) では，提案手法のすべての処理を実時間で処理するのは困難である．そこで，左耳用の信号および右耳用の信号の生成のために，それぞれひとつずつ別の DSP を使用し，全部で 3 つの DSP を並列動作されることにより，これを実現している．ただし，本プログラムでは，HRTF との畳み込みを FIR フィルタとしてではなく，FFT を用いて行っている．したがって，この部分を FIR フィルタで実現することで，計算負荷が更に軽減される可能性がある．

TMS320C6201 の特徴のひとつは，浮動小数点型 DSP であることと同時に，C 言語でのプログラム開発を強く意識して開発されている点である []．したがって，アセンブリ言語ではなく，C 言語でも十分にその性能を発揮することができる．そこで，本システムは，可読性，改造の容易性等を考慮して，アセンブリ言語ではなく C 言語でコードを作成した．そのため，ソースコードの書き方がコンパイラでの最適化に大きな影響を与える．掲載したコードは，この点を考慮して書くことによって，実時間動作を達成しているため，下手に書き換えると実時間動作が保証されなくなる可能性がある．コンパイラに，効果的に最適化を行わせるためには，ソースコードを書く際に，スーパーコンピュータや並列サーバーでのプログラミングと同様な書き方をすると，高効率な最適化が実現される．特にメモリは，極力内部メモリを使用するようにし，どうしても内部メモリだけでは容量が足りない場合にのみ，malloc により外部メモリを確保して利用するようにする．また，TMS320C6201 のマニュアルによると double 型の変数も扱えるはずであるが，実際には float 型しか扱えなかったので，本プログラムでは <mathf.h> をインクルードして float 型の算術関数を利用している．CPCI-DSP46701 が発売され始めてすぐに購入したマシンであるため，まだコンパイラの開発が十分ではなく，それが原因で double 型が扱えないのかもしれない．

ソースコード中で使用している DSP 制御用の関数は，別途 SDS へ問い合わせて送付していただいたものであるため，これらに関する記述は S-RTP station のマニュアルには掲載されていない．SDS 制御用の関数ライブラリと一緒に送付されて来たサンプルプログラムには，十分な注釈がソースコード中に付記されているので，それを読むことで使用方法を知ることができる．また，このライブラリのソースコードも添付されているので，そちらを参照してもよい．

本システム全体の流れを図 13 に示す．DSP 間でのデータ通信は共有メモリを介して行われる．通信の帯域幅および共有メモリの正確なサイズは，マニュアルを参照して欲しいが，掲載したプログラムはそのほぼ全てを利用している．DSP-A において A/D されたデータは，共有メモリへ書き出される．その後，そのデータは，左耳用の提示音を生成する DSP-B および右耳用の提示音を生成する DSP-C で，それぞれ読み取られる．

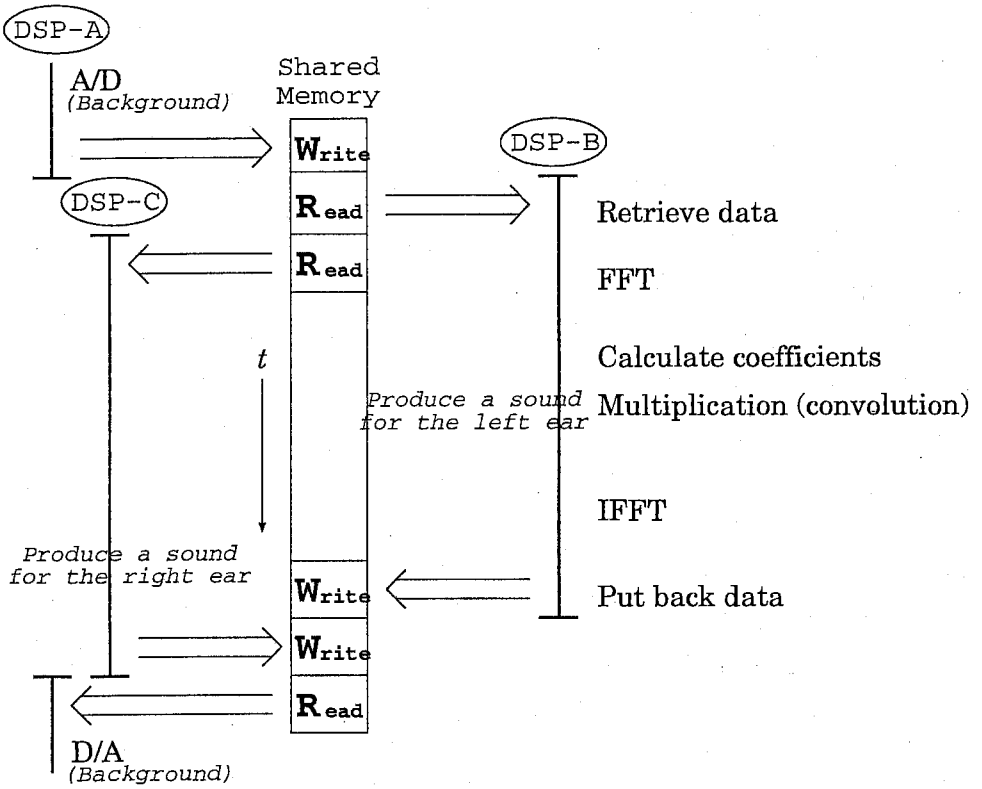


図 13: RSSC システムの処理の流れ図

共有メモリとDSP-BおよびDSP-Cとの間でのデータ通信では，その後の計算で使用されない余分なデータも転送されているが，必要とするデータのみを転送するためのデータのリフォーマットに要する時間が，実時間動作を破綻させてしまうため，本プログラムではあえて不要なデータも一緒にして共有メモリへの読み書きを行っている．共有メモリへのデータの受け渡しの際には，音響データの最後に剛球と現在の聴取位置との関係に関する情報を32bitデータとして付加して伝送している．

DSP-BおよびDSP-Cでは，共有メモリから読み込んだデータを音響データと位置情報データに分離し，音響情報データは，畳み込み演算に備えて高速フーリエ変換される．一方，位置情報データは，それを基にして係数を算出し，固有ベクトルと積和することにより，対応する伝達関数を生成する．この伝達関数と先に求めた音響データの周波数特性とを掛けて，逆高速フーリエ変換を実行することにより提示すべき音響信号が生成される．これを再び共有メモリへ戻し，DSP-Aがそれを読み出してD/Aすることによって，所望の信号がユーザーへ提示されることになる．

```
/*
SAD2060アクセスルーチンのサンプルプログラム
DSP1で実行し、A/D、D/Aを受け持つ。

注：処理の順番や利用アドレスを変更すると，間に合わなくなるので，
    リライトはしないほうがよい．(コンパイラの最適化の関係です)
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "c6xsys.h"
#include "sad6x.h"
#include "sad2060.h"
#include "intr.h"

#define M_PI 3.14159265358979323846
#define FRAME 256 // 一回当たりの処理点数
#define HALF  128
#define CHANNELS 4 // チャネル数

#define Taplen_Byte_float (4*FRAME)       // sizeof(float)=4
#define Taplen_Byte_Int (4*FRAME)         // sizeof(int)=4

const int Clock = 48 * 1000; // 48KHz
//const int Clock = 44 * 1000; // 44.1KHz
const int FifoSize = CHANNELS * FRAME * 4; // FIFOサイズ
enum {AD,DA};
```

```
void main(void)
{
short buf[FRAME+1][CHANNELS];
SAD_ARG ad_arg[2];
int i,ch;
//int loop=1000;
int ibuff1, ibuff2;
short sbuff;

c6xconf();
printf("\n\n\n\n\n\n\n\n\n\nDSP start!.\n");

reg_10k->SH_ADDR_REG = 0x101 + 0x8000;
reg_10k->SH_DATA_REG = 0;

// パラメータ初期化
memset(&ad_arg[0],0,sizeof(SAD_ARG)*2);

// 初期設定、FIFO設定
  dma_reset();
SAD_INIT(1);
SAD_ADAFIFO(FifoSize,FifoSize);

// ADアーギュメント設定
ad_arg[AD].channels = CHANNELS;
ad_arg[AD].mode = ADNORM;
// ad_arg[AD].clkmode = OSC48K;
ad_arg[AD].clkmode = OSC44K;
// ad_arg[AD].clkmode = INTCLK;
ad_arg[AD].clock = Clock;     //clkmode=INTCLK の時のみ有効
ad_arg[AD].frame1 = 0;
for( i = 0 ; i < CHANNELS ; i++ ) ad_arg[AD].chanum[i] = i+1;
if( SAD_PACKET( &ad_arg[AD] ) < 0 ) {
printf( "SAD_PACKET error. (AD)\n" );
exit(0);
}
// DAアーギュメント設定
ad_arg[DA].channels = CHANNELS;
ad_arg[DA].mode = DANORM;
// ad_arg[DA].clkmode = OSC48K;
ad_arg[DA].clkmode = OSC44K;
```

```
// ad_arg[DA].clkmode = INTCLK;
ad_arg[DA].clock = Clock;
ad_arg[DA].frame1 = 0;
for( i = 0 ; i < CHANNELS ; i++ ) ad_arg[DA].chanum[i] = i+1;
if( SAD_PACKET( &ad_arg[DA] ) < 0 ) {
printf( "SAD_PACKET error. (DA)\n" );
exit(0);
}

// DA 遅延分のバッファを確保
for(i=0;i<FRAME;i++){
for(ch=0;ch<CHANNELS;ch++){
buf[i][ch]  = 0;
}
}
for(i=0;i<2;i++){ // 2Frame 分、DA 用バッファに書き込み。
SAD_WRITE(0,buf,CHANNELS*FRAME>>1);
}

        // AD, DA を同時にスタートさせる
printf( "SAD_ADASTART\n" );
if( SAD_ADASTART(0) < 0 ) {
printf( "SAD_ADASTART error\n" );
exit(0);
}

while(1){ // 無限ループ
// FIFO バッファの中身を読む
reg_10k->SH_ADDR_REG = 0x101 + 0x8000;
while( reg_10k->SH_DATA_REG != 0 ){;}
if( SAD_READ(0,buf,CHANNELS*FRAME>>1) < 0 ) {
printf( "SAD_READ error\n" );
exit(0);
}

        // 球体の位置に関する情報を A/D する
sbuff = (buf[HALF][1]&0xff00) | ((buf[HALF][3]&0xff00)>>8);
buf[FRAME][0] = sbuff;
buf[FRAME][2] = sbuff;

// A/D したデータを共有メモリへコピーする
        // 32bit 通信のため, LR チャネルをひとつにまとめる
```

```
for( i=0; i<=FRAME; i++ ) {
ibuff1 = (int)buf[i][0];
ibuff2 = (int)buf[i][2];
// INTR_DISABLE( CPU_INT6 );     // 排他処理関係
reg_10k->SH_ADDR_REG = 0x8000 + i;
reg_10k->SH_DATA_REG = (ibuff1<<16 | (0x0000ffff&ibuff2));
// INTR_ENABLE( CPU_INT6 );
}
reg_10k->SH_ADDR_REG = 0x8101;
reg_10k->SH_DATA_REG = 1;

// データを共有メモリから受け取る
reg_10k->SH_ADDR_REG = 0x101 + 0x8000;
while( reg_10k->SH_DATA_REG != 5 ) {;}
for( i=0; i<HALF; i++ ) {
// 左耳音データ取得
// INTR_DISABLE( CPU_INT6 );
reg_10k->SH_ADDR_REG = 0x8000 + i;
ibuff1 = reg_10k->SH_DATA_REG;
// INTR_ENABLE( CPU_INT6 );
buf[2*i][0] = (short)(0x0000ffff&ibuff1);
buf[2*i+1][0] = (short)((ibuff1&0xffff0000)>>16);

// 右耳音データ取得
// INTR_DISABLE( CPU_INT6 );
reg_10k->SH_ADDR_REG = 0x8080 + i;
ibuff1 = reg_10k->SH_DATA_REG;
// INTR_ENABLE( CPU_INT6 );
buf[2*i][2] = (short)(0x0000ffff&ibuff1);
buf[2*i+1][2] = (short)((ibuff1&0xffff0000)>>16);
}
reg_10k->SH_ADDR_REG = 0x101 + 0x8000;
reg_10k->SH_DATA_REG = 0;

// FIFOへ、DAデタの書き出し
if( SAD_WRITE(0,buf,CHANNELS*FRAME>>1) < 0 ) {
printf( "SAD_WRITE error\n" );
exit(0);
}
}
// 終了しません。
```

```
SAD_STOP();
printf("DSP finish.\n");
exit(0);
}
```

B　左耳用処理プログラム

```
/*
左耳のデータ処理
*/

#include <stdio.h>
#include <stdlib.h>
#include <mathf.h> // double 系の math.h ではダメ(中の関数も同様)
#include <string.h>
#include "c6xsys.h"
#include "intr.h"
#include "evector256.h"
#include "coeffs.h"

#define M_PI 3.14159265358979323846
#define FRAME 512  // 一回当たりの処理点数
#define HALF  256
#define QUATER 128
#define POW 9                              // 2^POW=FRAME

#define Taplen_Byte_float (4*FRAME)        // sizeof(float)=4
#define Taplen_Byte_Int (4*FRAME)          // sizeof(int)=4
#define Half_Byte_float (2*FRAME)          // sizeof(float)=4

static int ik[FRAME];
static float dft_cc[FRAME], dft_ss[FRAME];
static float idft_cc[FRAME], idft_ss[FRAME];
static float hamm[FRAME];
static float factor1 = 26. / 255.;
static float factor2 = 46. / 255.;

/***** ここから FFT のルーチン *****/
void table ( int qss, int *ikk, float *cc, float *ss, int pow)
{
  int i, j, mn, nc;
```

```
  float q;

  *ikk = 0; mn = HALF; nc = 1;
  for(i=1; i<=pow; i++){
    for(j=0; j<nc; j++){
      *(ikk + j + nc) = *(ikk + j) + mn;
    }
    nc <<= 1; mn >>= 1;
  }
  q = qss * 2.0 * M_PI / (float) FRAME;
  for( i=0; i<HALF; i++){
    cc[i] = cosf((float)i * q);
    ss[i] = sinf((float)i * q);
  }
}

void FFTSetup( void ) {
  int i;
  for( i=-HALF; i<HALF; i++ )
    hamm[i+HALF] = 0.54+0.46*cosf(2.*M_PI*(2.*(float)i+1.)/2./(float)(FRAME-1));
  table( 1, ik, dft_cc, dft_ss, 9);      /* 2^9 = 512 = FRAME */
  table( -1, ik, idft_cc, idft_ss, 9);
}

void fft( float *xx, float *yy, float *cc, float *ss)
{
  int lg, lf, mf, nf, ka, kb, n2, ix;
  float tr, tj;

  lg = 1;
  for(lf=1; lf<=POW; lf++){
    n2 = HALF / lg;
    for(mf=1; mf<=lg; mf++){
      for( nf=0; nf<=n2-1; nf++){
ix = nf * lg;
ka = nf + 2 * n2 * (mf - 1);
kb = ka + n2;
tr = *(xx + ka) - *(xx + kb);
tj = *(yy + ka) - *(yy + kb);
*(xx + ka) += *(xx + kb);
*(yy + ka) += *(yy + kb);
*(xx + kb) = tr * *(cc + ix) + tj * *(ss + ix);
```

```
*(yy + kb) = tj * *(cc + ix) - tr * *(ss + ix);
      }
    }
    lg *= 2;
  }
}

void dft( float *datax, float *datay, float *xout, float *yout)
{
   int i;
  float xout1[FRAME], yout1[FRAME], corr;

  memcpy( &xout1[0], datax, Taplen_Byte_float );
  memcpy( &yout1[0], datay, Taplen_Byte_float );

  fft( xout1, yout1, dft_cc, dft_ss );

  corr = 1. / sqrtf((float) FRAME);
  for(i=0; i<FRAME; i++){
    *xout++ = xout1[*(ik + i)] * corr;
    *yout++ = yout1[*(ik + i)] * corr;
  }
}

void idft( float *datar, float *datai, float *xout, float *yout)
{
  int i;
  float xout1[FRAME], yout1[FRAME], corr;

  corr = sqrtf((float) FRAME);
  for(i=0; i<FRAME; i++){
    xout1[i] = *datar++ * corr;
    yout1[i] = *datai++ * corr;
  }
  fft( xout1, yout1, idft_cc, idft_ss);
  corr = 1. / (float) FRAME;
  for(i=0; i<FRAME; i++){
    *xout++ = xout1[*(ik + i)] * corr;
    *yout++ = yout1[*(ik + i)] * corr;
  }
}
/***** ここまでルーチン *****/
```

```
void Convolv( int xyp, float *datax, float *datay )
{
  int i, j;
  int lx;
  float scale, diff, rx, xp, xx, yy;
  float coefi[4], coefr[4], weight, rr, ii;

  xx = ((float)((xyp & 0x0000ff00) >> 8)) * factor1;     // from 0 to 26
  yy = ((float)(xyp & 0x000000ff)) * factor2;            // from 0 to 46

  diff = 4.;
  scale = (46. - yy) * 0.2 + 1.;
  xp = (13. - xx) * scale - diff + 19.;

  //  printf( "L = %f\n", xp );

  if( xp > 38.99 ) xp = 38.99;    // 39未満でないとだめ
  if( xp < 0. ) xp = 0.;

/* 係数の線形1次補完 */
  lx = (int) floorf( xp );
  rx = xp - (float) lx;
  lx *= 2;

  for( i=0; i<4; i++ ) {
    coefr[i] = rx * coeffs[lx+2][i] + (1. - rx) * coeffs[lx][i];
    coefi[i] = rx * coeffs[lx+3][i] + (1. - rx) * coeffs[lx+1][i];
  }

  for( i=2; i<QUATER; i++ ) {
    rr = ii = 0.;
/*** この下の式の計算が最も負荷が掛かります ***/
    for( j=0; j<4; j++) {
      rr += evector[2*j][i] * coefr[j] - evector[2*j+1][i] * coefi[j];
      ii += evector[2*j][i] * coefi[j] + evector[2*j+1][i] * coefr[j];
    }
    weight = sqrtf(rr * rr + ii * ii);

    weight *= 0.75;  // オーバーフロー回避のため

/* 重み掛け */
```

```
    datax[i] *= weight;
    datay[i] *= weight;
    datax[FRAME-i-1] *= weight;
    datay[FRAME-i-1] *= weight;
  }

  for( i=QUATER; i<HALF; i++ ) {
    datax[i] *= weight;
    datay[i] *= weight;
    datax[FRAME-i-1] *= weight;
    datay[FRAME-i-1] *= weight;
  }
}

void main(void)
{
int i;
float idatax[FRAME], idatay[FRAME];
float odatax[FRAME], odatay[FRAME];
float oodatax[HALF];
float zeros[FRAME], buffer[HALF];
int xyp, ibuff;

c6xconf();
printf("\n\n\n\n\n\n\n\n\n\nDSP start!.\n");

for( i=0; i<HALF; i++ )
  zeros[i] = zeros[FRAME-i-1] = buffer[i] = 0.;
// FFT のセットアップ
FFTSetup();

// 無限ループ
while(1){
                memcpy( idatax, oodatax, Half_Byte_float );
// A-DSP から共有メモリを通してデータを取得
reg_10k->SH_ADDR_REG = 0x8101;
while( reg_10k->SH_DATA_REG != 1 ){;}
//i=FRAME is necessary for getting position of the ball.
for( i=0; i<=HALF; i++ ) {
// INTR_DISABLE( CPU_INT6 );
reg_10k->SH_ADDR_REG = 0x8000 + i;
ibuff = reg_10k->SH_DATA_REG;
```

```
// INTR_ENABLE( CPU_INT6 );
oodatax[i] = (float) ((short) ((ibuff & 0xffff0000) >> 16));
}
xyp = ibuff;
reg_10k->SH_ADDR_REG = 0x8101;
reg_10k->SH_DATA_REG = 2;

// 左耳用処理
memcpy( &idatax[HALF], oodatax, Half_Byte_float );
memcpy( idatay, zeros, Taplen_Byte_float );
// ハミング窓による窓掛け
for( i=0; i<FRAME; i++ )  idatax[i] *= hamm[i];
dft( idatax, idatay, odatax, odatay );
Convolv( xyp, odatax, odatay );
idft( odatax, odatay, idatax, idatay );
for( i=0; i<HALF; i++ )   idatax[i] += buffer[i];
memcpy( buffer, &idatax[HALF], Half_Byte_float );

reg_10k->SH_ADDR_REG = 0x8101;
while( reg_10k->SH_DATA_REG != 3 ){;}
// A-DSP へ、DA データの書き出し
for( i=0; i<QUATER; i++ ) {
   ibuff = (((int)idatax[2*i])&0xffff)|(((int)idatax[2*i+1])<<16);
// INTR_DISABLE( CPU_INT6 );
reg_10k->SH_ADDR_REG = 0x8000 + i;
reg_10k->SH_DATA_REG = ibuff;
// INTR_ENABLE( CPU_INT6 );
}
reg_10k->SH_ADDR_REG = 0x8101;
reg_10k->SH_DATA_REG = 4;
}
// ここへは来ません

printf("DSP finish.\n");
exit(0);
}
```

C　右耳用処理プログラム

FFT 関連の関数部分は，前節の「左耳用処理プログラム」に掲載したソースプログラムも全く等価なので，ここでは省略する．左耳用のソースプログラムとの相異点は，Convolv 内で音源位置をシフトさせる方向が逆であることと，DSP 間の同期を取るため

に付されているフラグの値が違う点だけである.

```
/*
右耳用データ処理
*/

void Convolv( int xyp, float *datax, float *datay )
{
  int i, j;
  int lx;
  float scale, diff, rx, xp, xx, yy;
  float coefi[4], coefr[4], weight, rr, ii;

  xx = (float) ((xyp & 0x0000ff00) >> 8) * factor1;    // from 0 to 26
  yy = (float) (xyp & 0x000000ff) * factor2;     // from 0 to 46

  diff = 4.;
  scale = (46. - yy) * 0.2 + 1.;
  xp = (13. - xx) * scale + diff + 19.;

  //  printf( "R = %f %f %f\n", xp, yy, scale);

  if( xp > 38.99 ) xp = 38.99;   // should be not more than 39.
  if( xp < 0. ) xp = 0.;

/* 係数の線形1次補完 */
  lx = (int) floorf( xp );
  rx = xp - (float) lx;
  lx *= 2;

  for( i=0; i<4; i++ ) {
    coefr[i] = rx * coeffs[lx+2][i] + (1. - rx) * coeffs[lx][i];
    coefi[i] = rx * coeffs[lx+3][i] + (1. - rx) * coeffs[lx+1][i];
  }

  //  for( i=2; i<HALF; i++ ) {
  for( i=2; i<QUATER+10; i++ ) {
    rr = ii = 0.;
/*** ここのループがめちゃくちゃ重い ***/
    for( j=0; j<4; j++ ) {
      rr += evector[2*j][i] * coefr[j] - evector[2*j+1][i] * coefi[j];
      ii += evector[2*j][i] * coefi[j] + evector[2*j+1][i] * coefr[j];
    }
```

```
    weight = sqrtf(rr * rr + ii * ii);

    weight *= 0.75;  // To prevent overflow.
    // for deformation
    //    weight *= weight;

/* 重み掛け */
    datax[i] *= weight;
    datay[i] *= weight;
    datax[FRAME-i-1] *= weight;
    datay[FRAME-i-1] *= weight;
  }

  for( i=QUATER+10; i<HALF; i++ ) {
    datax[i] *= weight;
    datay[i] *= weight;
    datax[FRAME-i-1] *= weight;
    datay[FRAME-i-1] *= weight;
  }
}


void main(void)
{
int i;
float idatax[FRAME], idatay[FRAME];
float odatax[FRAME], odatay[FRAME];
float oodatax[HALF];
float zeros[FRAME], buffer[HALF];
int xyp, ibuff;

c6xconf();
printf("\n\n\n\n\n\n\n\n\n\nDSP start!.\n");

for( i=0; i<HALF; i++ )
  zeros[i] = zeros[FRAME-i-1] = buffer[i] = 0.;
// setup for FFT
FFTSetup();

// infinit roop
while(1){
                memcpy( idatax, oodatax, Half_Byte_float );
```

```
// read data from ADSP via the shared memory
reg_10k->SH_ADDR_REG = 0x8101;
while( reg_10k->SH_DATA_REG != 2 ){;}
// i=FRAME is necessary for getting position of the ball.
for( i=0; i<=HALF; i++ ) {
// INTR_DISABLE( CPU_INT6 );
reg_10k->SH_ADDR_REG = 0x8000 + i;
ibuff = reg_10k->SH_DATA_REG;
// INTR_ENABLE( CPU_INT6 );
oodatax[i] = (float)((short)(ibuff & 0x0000ffff));
}
xyp = ibuff;
reg_10k->SH_ADDR_REG = 0x8101;
reg_10k->SH_DATA_REG = 3;

// 右耳用処理
memcpy( &idatax[HALF], oodatax, Half_Byte_float );
memcpy( idatay, zeros, Taplen_Byte_float );
// Windowing by the Hamming window
for( i=0; i<FRAME; i++ )  idatax[i] *= hamm[i];
dft( idatax, idatay, odatax, odatay );
Convolv( xyp, odatax, odatay );
idft( odatax, odatay, idatax, idatay );
for( i=0; i<HALF; i++ )   idatax[i] += buffer[i];
memcpy( buffer, &idatax[HALF], Half_Byte_float );

reg_10k->SH_ADDR_REG = 0x101 + 0x8000;
while( reg_10k->SH_DATA_REG != 4 ){;}
// DSP1へ、DAデータの書き出し
for( i=0; i<QUATER; i++ ) {
ibuff = (((int)idatax[2*i])&0xffff)|(((int)idatax[2*i+1])<<16);
// INTR_DISABLE( CPU_INT6 );
reg_10k->SH_ADDR_REG = 0x8080 + i;
reg_10k->SH_DATA_REG = ibuff;
// INTR_ENABLE( CPU_INT6 );
}
reg_10k->SH_ADDR_REG = 0x101 + 0x8000;
reg_10k->SH_DATA_REG = 5;
}
// never reach here

printf("DSP finish.\n");
```

```
exit(0);
}
```

D Makefile

参考のために，Makefile も付記しておく．

```
PROJECT1 = share1
PROJECT2 = share2
PROJECT3 = share3
TARGET1 = $(PROJECT1).out
TARGET2 = $(PROJECT2).out
TARGET3 = $(PROJECT3).out
TARGETS =  $(TARGET1) $(TARGET2) $(TARGET3)
USRCOBJS = $(PROJECT).obj
COBJS = $(SYSCOBJS) $(USRCOBJS)
AOBJS = $(SYSAOBJS) $(USRAOBJS)
OBJS1 = $(SYSCOBJS) $(PROJECT1).obj $(SYSAOBJS) $(USRAOBJS)
OBJS2 = $(SYSCOBJS) $(PROJECT2).obj $(SYSAOBJS) $(USRAOBJS)
OBJS3 = $(SYSCOBJS) $(PROJECT3).obj $(SYSAOBJS) $(USRAOBJS)

CMD = c6xmap0.cmd
CMD2 = c6xmap1.cmd
###
TI_ROOT = c:\ti\c6000\cgtools
TI_BIN = $(TI_ROOT)\bin
TI_INCLUDE = $(TI_ROOT)\include
TI_LIB = $(TI_ROOT)\lib
###
SRTP_DIR = c:\srtp\station6x\dsp
DEV_INCLUDE = $(SRTP_DIR)\devlib
DRV_INCLUDE = $(SRTP_DIR)\drvlib
SRTP_INCLUDE = $(SRTP_DIR)\sys\src
SAD_INCLUDE = $(SRTP_DIR)\sadsys
LIBDIR = $(SRTP_DIR)\sys
RTSLIB = $(SRTP_DIR)\sys\srcv3
SADLIB = $(SAD_INCLUDE)

CC = $(TI_BIN)\cl6x
AS = $(TI_BIN)\asm6x
LN =  $(TI_BIN)\lnk6x
```

```
CFLAGS = -g -o1 -mi6 -ss -ml0 -mv6701 -I$(TI_INCLUDE) -I$(DEV_INCLUDE) \\
-I$(DRV_INCLUDE) -I$(SRTP_INCLUDE) -I$(SAD_INCLUDE) \\
-Ic:\Temp\sadlib\sadsys

AFLAGS = -l

LKLAGS =  -ic:\Temp\sadlib\sys -l dev6x.lib -l c67sys.lib -lrts6701.lib \\
-l drv6x.lib -l c:\Temp\sadlib\sadsys\sad6x.lib -x

all:: $(TARGETS)

$(TARGET1): $(OBJS1) $(CMD) $(CMD2) makefile
$(LN) $(LKLAGS) $(CMD2) $(PROJECT1).obj -o $(PROJECT1).out -m $(PROJECT).map
$(TARGET2): $(OBJS2) $(CMD) $(CMD2) makefile
$(LN) $(LKLAGS) $(CMD2) $(PROJECT2).obj -o $(PROJECT2).out -m $(PROJECT).map
$(TARGET3): $(OBJS3) $(CMD) $(CMD2) makefile
$(LN) $(LKLAGS) $(CMD2) $(PROJECT3).obj -o $(PROJECT3).out -m $(PROJECT).map

$(PROJECT1).obj: $(PROJECT1).c makefile
$(CC) $(CFLAGS) $(PROJECT1).c
$(PROJECT2).obj: $(PROJECT2).c makefile coeffs.h
$(CC) $(CFLAGS) $(PROJECT2).c
$(PROJECT3).obj: $(PROJECT3).c makefile coeffs.h
$(CC) $(CFLAGS) $(PROJECT3).c

drv$(PROJECT1)$(COBJS1): $(INCLUDES)
drv$(PROJECT2)$(COBJS2): $(INCLUDES)
drv$(PROJECT3)$(COBJS3): $(INCLUDES)
```

E　使用方法

RSSC を使用するにあたって，必要となる装置は，

- S-RTP station (DSP ボード内蔵パソコン)
- HP 7000 J200 (DASBOX 制御用)
- DASBOX (A/D，D/A 装置)

の 3 つである．これらの装置を図 14 に従って接続する．

RSSC の実際の処理は，全て S-RTP station の DSP で行っている．したがって，他のアプリケーションにより制御される，聴取者と障害物との位置関係を何らかの方法で DSP へ供給する必要がある．ひとつの方法として，S-RTP station のホストコンピュー

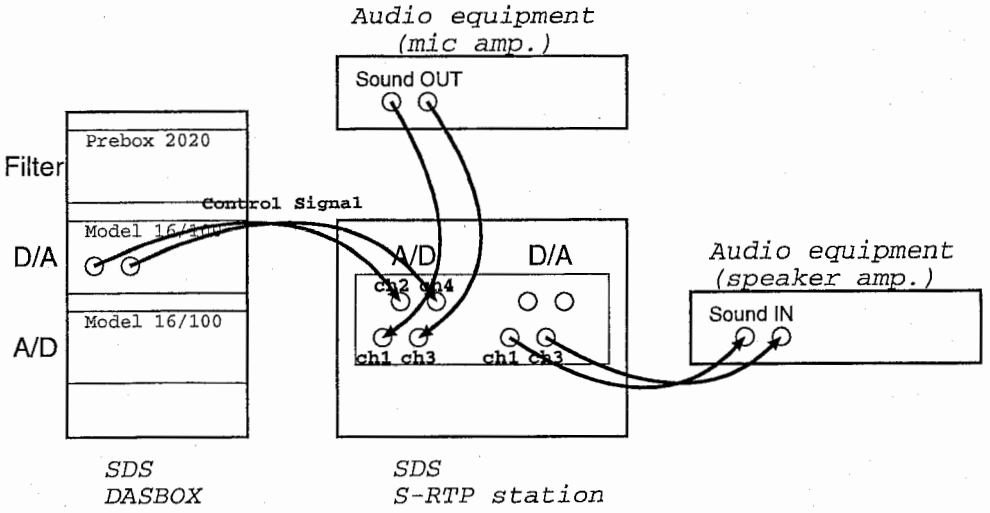


図 14: RSSC を使用する際の，各装置間の接続

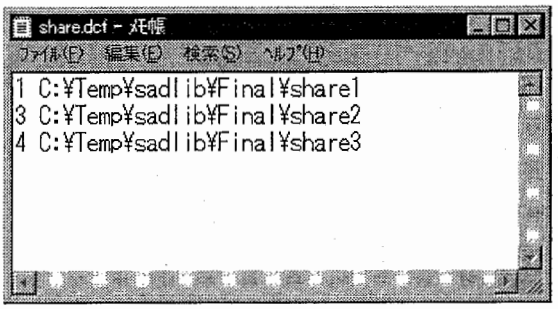


図 15: S-RTP station の DSP で実行させるための DCF ファイルの例

タ (Windows NT) と DSP 間での通信を利用することにより，位置関係の情報を供給する方法が考えられる．しかし，この方法は通信処理に時間が掛かるため，実時間動作という必須の要求項目が破綻してしまう恐れがある．そこで，A/D，D/A 装置と DSP が直結していることを利用して，DSP へ位置関係の情報を直接供給することにする．この方法は，アナログ信号で制御用信号を供給し，S-RTP station で A/D することで DSP へその情報を供給する方法である．その結果，この制御用信号を何らかの方法で生成する必要がある．ここでは，DASBOX および HP マシンを使用しているが，適当な代替手段を用いても全く支障はない．

DSP でプログラムを実行するには，図 15 に示すような DCF ファイルを用意し，DSP ローダーを使って実行させる．ここで，ファイルの左側に書かれている数字は，どの実行ファイルをどの DSP で実行させるのかを指定するものであり，1，2，3，4 が，DSP-A，DSP-B，DSP-C，DSP-D にそれぞれ対応している．左右耳それぞれに提示する音を生成するプログラムは，どの DSP を使用しても構わないが，A/D，D/A を行うプログラムは 1 (DSP-A) を指定する．これは，S-RTP station のマニュアルに掲載されているブロック図を参照すれば分かるように，A/D，D/A の回路が DSP-A へしか接続されていないためである．DSP ローダーは，c:¥temp¥sadlib¥drv ディレクトリの中にある．こ

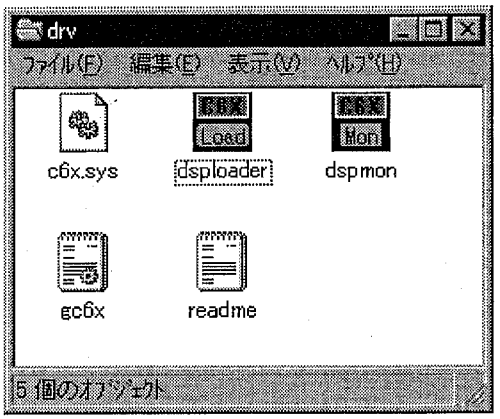


図 16: drv ディレクトリの中身

図 17: dsploader のウィンドウ

のディレクトリの内容を図 16 に示す．この dsploader を実行すると，図 17 に示すウィンドウが表示されるので，ここに，先の DCF ファイルを指定して実行すればよい．同じディレクトリにある dspmon は，各 DSP に対する標準出力を提供するものである．したがって，プログラム中に printf 関数を挿入することにより，DSP の現在の変数をモニターすることができる．ただし，この標準出力は，DSP の内部メモリの内容は表示可能であるが，外部メモリの内容は表示できないので注意が必要である．また，ディスプレイへの表示は，かなり時間の掛かる処理であるため，printf を挿入したがために実時間動作が破綻する場合がしばしばある．使用の際には，そのことを念頭に置いておく必要がある．