TR-H-304

奥行き運動観察時における重心動揺

宇和伸明，金子寛彦（ATR-HIP／東京工業大）

2001.2.1

ATR人間情報通信研究所

〒619-0288　京都府相楽郡精華町光台2丁目2-2　TEL: 0774-95-1011

ATR Human Information Processing Research Laboratories

2-2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan

Telephone: +81-774-95-1011

Fax　　: +81-774-95-1008

目　次

１．はじめに

人間の姿勢制御は前庭系，視覚系，体勢感覚系からの情報をもとに行われているが，視覚系が大きな役割を果たしていることは広く知られている．視覚情報の姿勢制御における有効性は，直立姿勢をとった場合，閉眼時には開眼時に比べて身体の動揺が増加するというEdwardsの報告[1]に示されている．視覚による姿勢制御において，視野の大きさが重要なファクタと考えられる．Paulusら[2]は，静止物を観察しているときに視野を制限すると，制限のない場合よりも直立姿勢時の身体の動揺が増加する，また，両眼視よりも単眼視の方が動揺が増加すると報告している．しかし，この後者の報告において両眼視が単眼視よりも姿勢制御に有効であるのは両眼性の視覚情報のためであるのか，それとも単に視野が広くなったためであるのかは明らかではない．

運動する対象の観察時には，体もそれにつれて動くことが報告されている[3]．対象が右に動いた場合，体は右に動き，対象が左に動けば左に動く．手前に別の物体を置いて，右に動く背景となる対象を固視した場合，体は左に動く．前者の場合，観察者は体が左に動いたと判断，後者の場合，背景が右に動けば手前の物体が左に動くように見え，その結果，体は右に動いたと判断するためであると考えられる．また，先のEdwardsの報告では，直立姿勢の被験者に対して左右に振った大きな振り子を呈示した場合，左右方向に動揺が増加したことも示されている．これらは視覚情報とその他の情報が矛盾した場合でも視覚情報が強い影響力を持つことを示している．

両眼視の効果については，両眼立体映像を呈示した研究として，両眼視差による奥行き情報は左右方向の重心動揺に影響を与えるという尾島ら[4]の報告や，広視野立体画像を用いた清水ら[5]の報告がある．しかし，これらの研究において立体であることの最大の特徴であると思われる，前後に動く映像と前後方向の姿勢制御の関係について報告されたものはない．

そこで，前後に動く奥行き運動を広視野立体映像として観察したときの，前後方向の重心動揺と奥行き知覚特性を測定した．このとき，姿勢制御のための情報として両眼視差のような両眼性の情報も用いていると考えられる．しかし，現実に前後に運動する物体を見た場合，奥行き知覚手がかりとして両眼性の情報だけでなく，近くにあるものは大きく見え，遠くにあるものは小さく見えるというような視角の変化もある．

Erkelensら[6]の報告によると，運動する視覚刺激のほかにリファレンスとなる映像がない場合に，視差変化のみの画像を呈示すると，その奥行き変化は知覚できない．ここで視差変化が姿勢制御に及ぼす影響として，次のふたつの可能性が考えられる．ひとつ目は視差変化は奥行き知覚に関与しないのと同様に姿勢制御にも関与しない．ふたつ目は視差変化は奥行き知覚には影響しないが，姿勢制御には影響する．もしふたつ目の仮定が正しいならば，奥行き知覚と姿勢制御は脳の中の異なったシステムにより実現されている可能性が高い．実験の結果より，視差変化は奥行き知覚にはほとんど影響しないが姿勢制御に強い影響を与えることが示された．

２．基本的実験環境

２．１．実験装置

視覚刺激呈示用装置は，液晶プロジェクタ，平面スクリーン，フレームメモリから構成された．立体表示は偏光眼鏡方式の背面投射型液晶ハイビジョンプロジェクタ２台により実現した．スクリーンサイズは縦１４００mm，横２４３０mm，有効画素数は縦１０２５画素，横１８６６画素で，視距離１mにおいて画角は縦７０．０°，横１０１．１°であった。また、表示はフィールド周波数６０Ｈｚのインタレース方式であった．プロジェクタへの映像入力方法は、フレームメモリのＲプレーン及び、Ｇプレーンをそれぞれ右目、左目用画像としてプロジェクタの輝度入力へ接続し、表示はグレースケールで行われた。フレームメモリは任意のフレーム間を往復再生するモードで使用した。プロジェクタの輝度は可能な限り入力信号に対して線形に変化するように調整した。

被験者の重心動揺測定装置として重心動揺計，ローパスフィルタ，データ記録用コンピュータを用いた．データのサンプリング周波数は１００Ｈｚ，分解能は０．０１５mm（±５Ｖ（１００mm）の出力を１６ビットで量子化）であった．重心動揺計の出力はカットオフ周波数１０Ｈｚのローパスフィルタを通した後サンプリングされた．刺激のタイミングはスクリーンの外枠に取り付けた光センサを用いて，視覚刺激上の

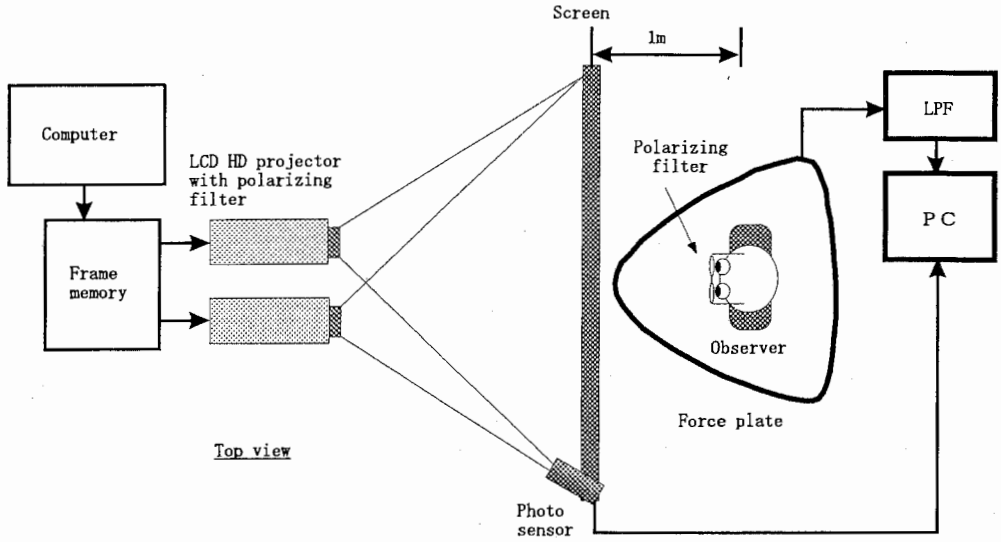

図１　実験装置の概要

タイミング信号を直接検出し，重心動揺データと同時に記録した．

２．２．呈示刺激

実験に用いた視覚刺激はランダムドットステレオグラムであり，以下に示す手順で作成した．図２に示すように各ドットの輝度値の分布は，正規分布曲線状であった．このドットの輝度分布曲線の中心位置を２００分の１ピクセルで制御することにより，実際の画面解像度より細かな視差を付けることが可能であり，奥行きの量子化誤差を軽減することができた．左右眼像用の２台の液晶プロジェクタそれぞれについて，実際の観察状態と同様に偏光フィルタを通して入力データ２５６階調のうち０から８階調刻みでデータを入力し輝度値を測定した。その結果を３次スプライン補間を用い補間して得られた入出力特性（図３）をもとに，輝度が線形になるように校正した．刺激は，このドット５０００個をランダムに配置した画像の中央の四角形の領域，もしくは全部に以下に示す奥行き運動知覚手がかりを与えたものであった．

３．重心動揺実験１

３．１．目的

奥行き運動を知覚するための情報は大きく２つに分けられる。ひとつめは単眼性情報であり、もうひとつは両眼性情報である。これらの情報が奥行き方向の姿勢制御に対してどのように影響を与えるのかを調べるために、情報で表現された奥行き運動刺激を呈示して重心動揺を測定した。

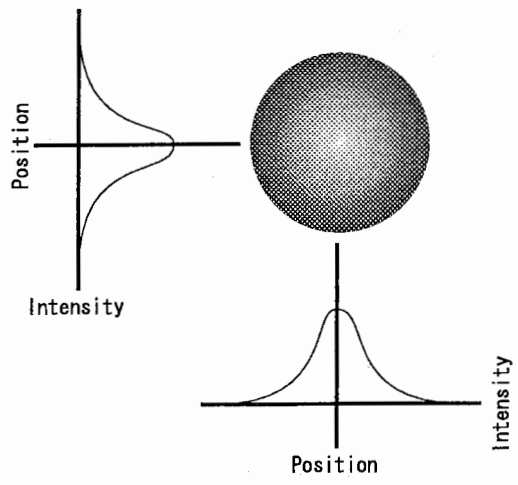

図２　ドットの概要

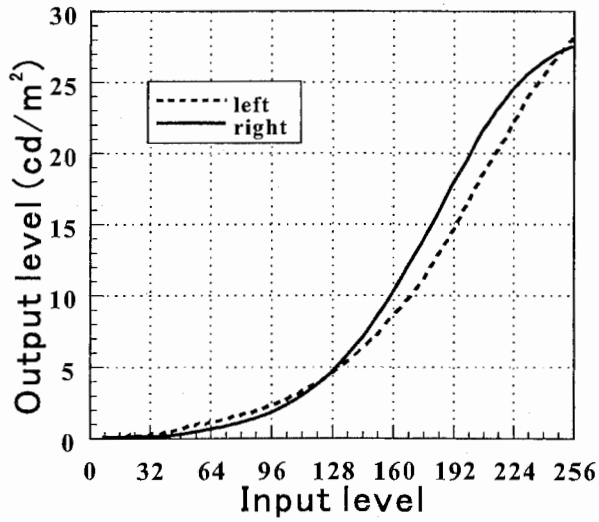

図３　プロジェクタの入出力特性

３．２．方法

視覚刺激はランダムドットで構成され、奥行き知覚手がかりの与え方は，（１）運動する領域に対して図４に示すように周波数０．５Ｈｚ振幅３０ｃｍで奥行き運動をする平面をシミュレートした動きをつけたもの，（２）（１）の刺激の動きより視差変化の要素のみを取り出して与えたもの，（３）（１）の刺激の動きより視角変化の要素のみを取り出して与えたものの３通りであった．この運動は平面位置、速度、加速度の時間変化が異なるため、重心動揺がどの要因と関連しているのかを観測しやすいためである。また、人間は周期０．３Ｈｚ程度で重心動揺しやすいといわれているが、もし人間の共振周波数であるとすれば、刺激に対する正確なレスポンスが得られないため、やや高めの０．５Ｈｚに設定している。それぞれの場合の左右眼の画像の座標変化は以下に示す式で表される．

（１）平面の動きをシミュレートした場合（視差と視覚変化を持つ場合）

$$x_L = \frac{D}{z}(x + \frac{d}{2}) - \frac{d}{2},\ y_L = \frac{D}{z}y. \qquad (1)$$
$$x_R = \frac{D}{z}(x - \frac{d}{2}) + \frac{d}{2},\ y_R = \frac{D}{z}y.$$

（２）視差変化のみを持つ場合

$$x_L = x + (\frac{D}{z} - 1)\frac{d}{2},\ y_L = y. \qquad (2)$$
$$x_R = x - (\frac{D}{z} - 1)\frac{d}{2},\ y_R = y.$$

（３）視覚変化のみを持つ場合

$$x_L = x_R = \frac{D}{z}x,\ y_L = y_R = \frac{D}{z}y. \qquad (3)$$

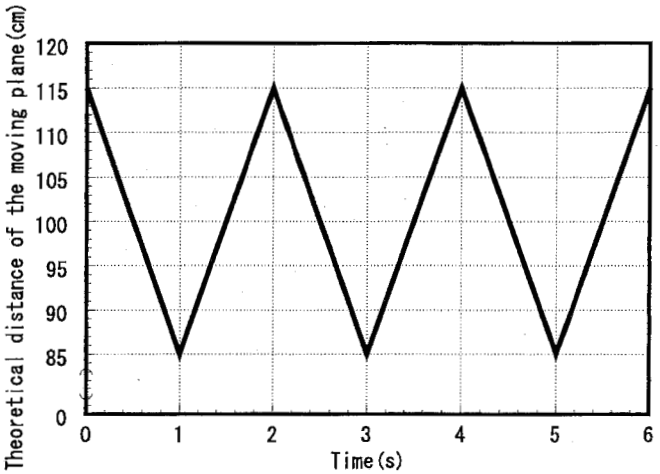

図４　運動平面の動き

ここでDは視点からスクリーンまでの距離，dは眼間距離，x_L，y_Lは原点を視点にとり左右方向をx，上下方向をy，奥行き方向をzとした空間座標(x, y, z)に立体画像を見せるときの左眼用画像のスクリーン上の座標，同様にx_R，y_Rは右眼用画像のスクリーン上の座標である．

運動する領域は表１に示される３種類の大きさを使用した．この運動する領域を以下運動平面と呼び，その大きさの小さいものから順に運動平面Ｓ，Ｍ，Ｌとする．これらの大きさは，観察者から見てスクリーンの位置に運動平面がある場合の視角を示しており，見かけの大きさではない．視角変化を伴なう刺激の場合，見かけの大きさは観察者に運動平面が近づいたときに大きくなり，遠ざかるときに小さくなる．このとき平面を構成するドットやドット間の距離もそれに応じた大きさに変化する．視差のみ変化する刺激の場合は，ドットの大きさやドット間の距離は常に一定である．実際の視覚刺激の一例を図５に示す．

被験者は，重心動揺計の上に両足をそろえ，スクリーンに正対した直立姿勢をとった．このとき固視点は設けなかったが，スクリーンの中心を見るよう指示した．　視覚刺激は，奥行き手がかり３種類，運動平面の大きさ３種類とコントロールとして静止画の１０通り（３×３＋１）で，呈示した順序はランダムである．視覚刺激は常時動いており，開始の合図とともに２分間重心動揺を測定した．測定した重心動揺の方向はスクリーンに向かって前後の方向である．この測定を運動する視覚刺激に対して１０回ずつ，静止画に対して３０回行い，被験者１人につき合計１２０回測定した．被験者は成人３名であった．

データの解析としてＦＦＴによるパワースペクトルの分析と，時間領域での１周期分の波形の平均を行っ

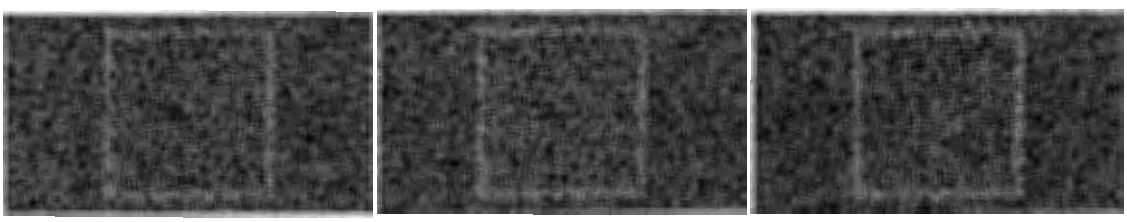

図５　実際の刺激の１例

表１　運動する領域の大きさ

control	静止画
S	縦31.4° ×横28.6°
M	縦58.7° ×横54.0°
L	縦70.0° ×横101.1°

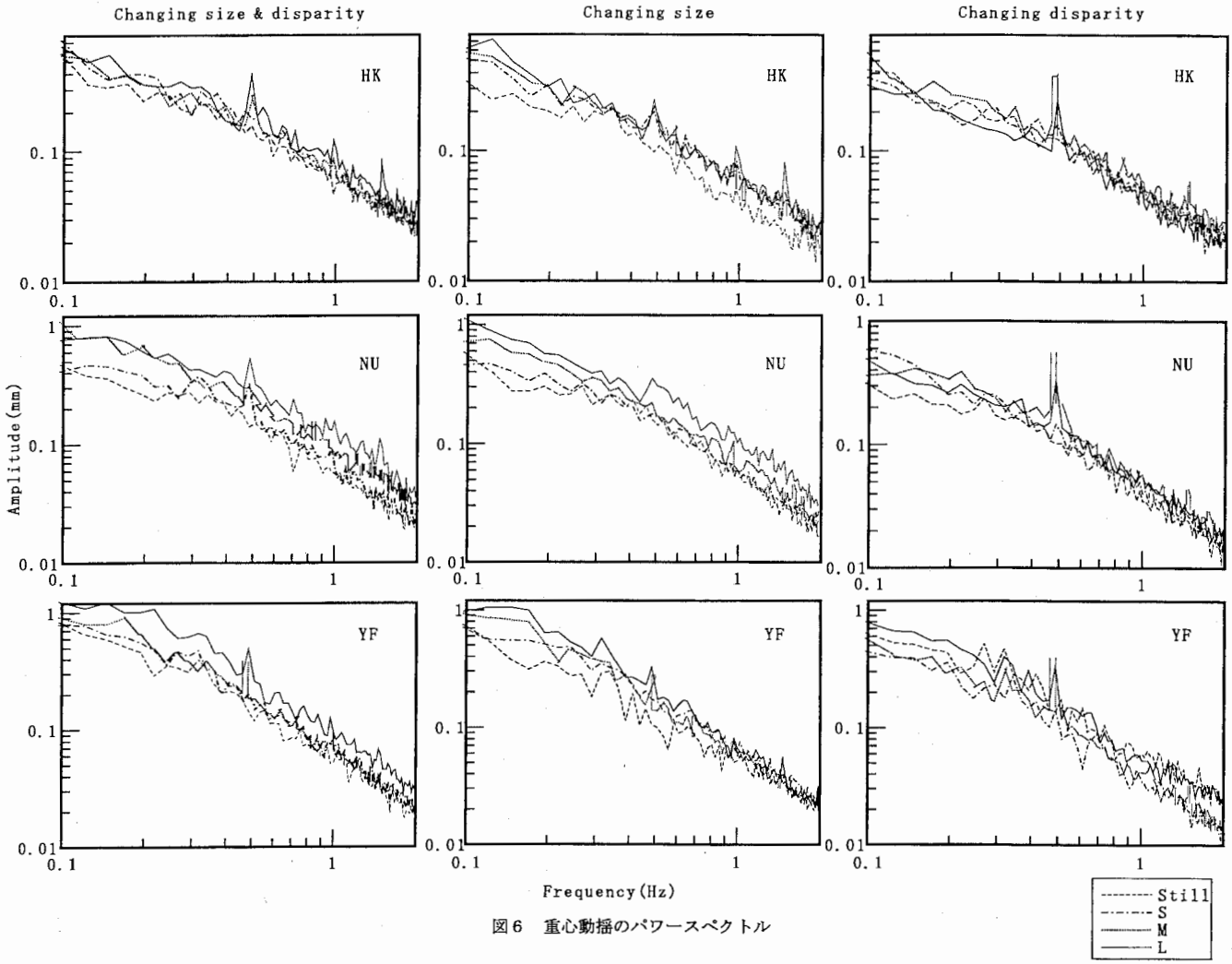

図６　重心動揺のパワースペクトル

た．ＦＦＴによる解析では，データの先頭から１０秒の位置から約４１秒（４０９６点）と，７０秒の位置から約４１秒（４０９６点）を取り出し，各々にＦＦＴをかけた結果の平均を求めた．ＦＦＴの計算に用いた区間が視覚刺激の１周期の長さ（２秒）に対し充分に長いので，有限データであることの影響はないと考えられる．時間領域での解析では，データの先頭から１０秒後から１１０秒後の間を視覚刺激の１周期毎に切り出して，１周期の平均の波形を求めた．

３．３．結果

パワースペクトルの結果を図６に示す．左から順に，視差と視角が変化する刺激の場合，視角変化のみの刺激の場合，視差変化のみの刺激の場合の結果を示したものである．横の列は各被験者の結果を表している．また各グラフの横軸は重心動揺の周波数を，縦軸は振幅を示している．視差と視角変化の刺激の場合は，刺激の運動の基本周波数である０．５Ｈｚ付近にローカルピークが観測され，運動平面が大きくなるに従い全体のパワーが上昇し，それに伴いピークも大きくなっている．視角変化のみの場合は，運動平面が大きくなるに従い全体のパワーは上昇するが，あまり明確なローカルピークは観測されない．視差変化のみの場合は，明確なローカルピークが観測されるが，全体のパワーは運動平面の大きさに影響されない．

時間領域での重心動揺の結果を図７に示す．各グラフはパワースペクトルのグラフと同様に，左から順に視差と視角変化，視角変化のみ，視差変化のみの結果を，各行は各被験者の結果を表している．グラフの横軸は時刻を表し，縦軸は重心の位置を表している．重心位置は値が正のときは後ろ（スクリーンの反対方向），負のときは前（スクリーン方向）へ移動していることを表す．対応する刺激画像の理論的位置は時刻０のとき最も被験者に近く(正方向)なり，時刻１のとき最も遠く（負方向）なる．そして，時刻２で再び被験者に近くなる．　視差と視角変化をする刺激の場合，０．５Ｈｚの波はある程度見られるが波形が一定せず，揺れの位相がそろっていない．視角変化のみの場合は，被験者ＨＫでは０．５Ｈｚの波がある程度見られるが波の位相が一定せず，他の２人の被験者では波そのものがほとんど観測できない．視差変化のみの場合，刺激の運動平面が小さい時（Ｓ）はほとんど動揺は見られないが，それ以外ではほぼ同位相の波形を示

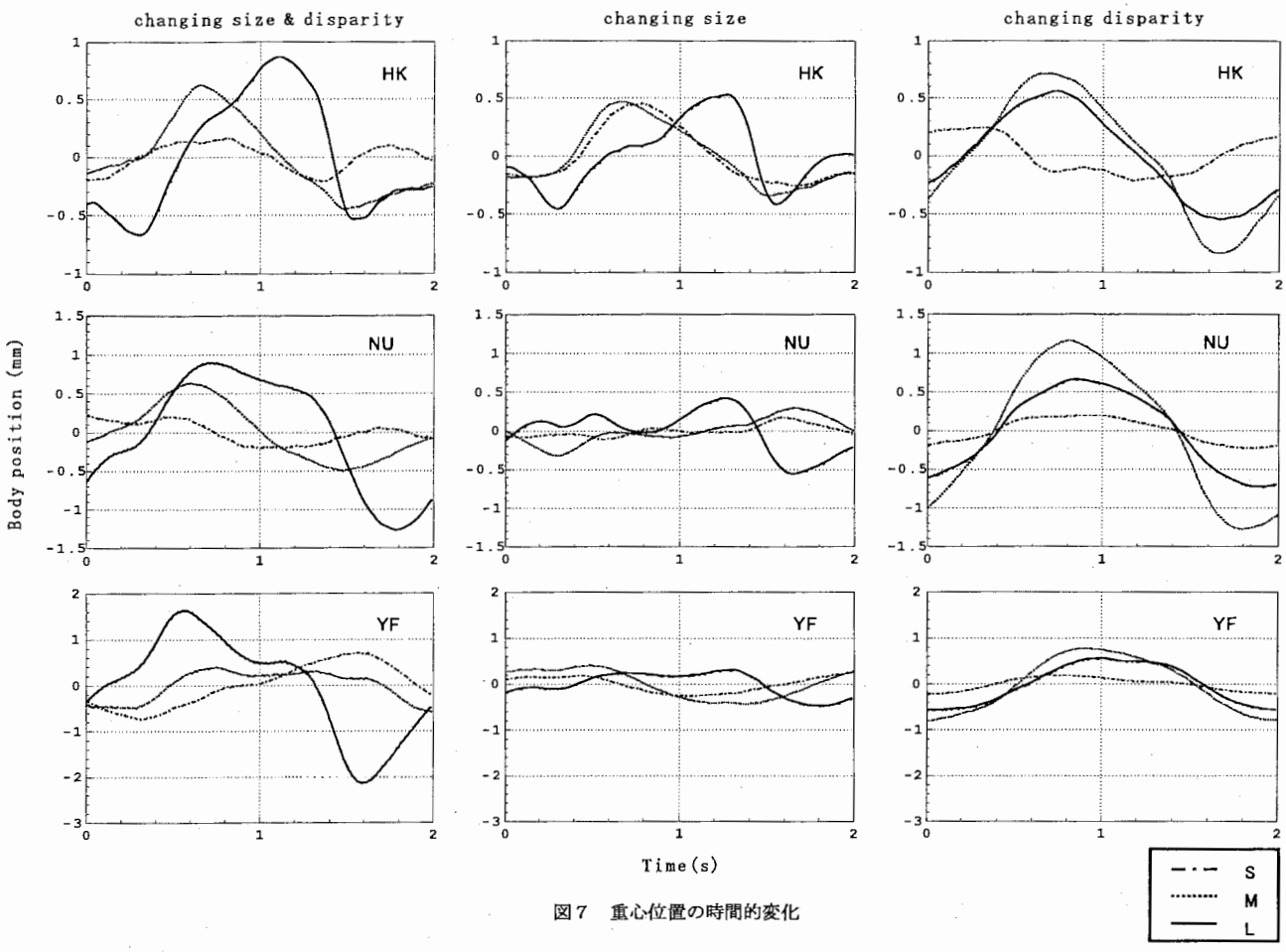

図７　重心位置の時間的変化

しており，正確に０．５Ｈｚの運動を引き起こしていることがわかる．

３．４．考察

視差変化のみを持つ刺激観察時の重心動揺波形において，すべての被験者で運動平面の動く周波数と同じ周波数にローカルピークが観察された．このローカルピークは静止画観察時には存在しないため，明らかに視覚刺激によって引き起こされたものであるといえる．また，運動平面が小さいときは，刺激に応じた動揺がほとんど起こらなかったが，大きくなると動揺が起こった．これは画面サイズの増加により，動いているドットの総量が増加したため，あるいは，周辺刺激の運動が加わったことが重心動揺を引き起こす要因となっている可能性が考えられる．

視角変化のみを持つ刺激のときは，刺激の周期と同じ周期の重心動揺は明確には観察されなかった．しかし，パワースペクトルの全体的なレベルは，静止画観察時よりも上昇しており，運動平面が大きくなるにつれて，その度合いが大きくなっている．この総重心動揺量の上昇の理由については，総合的考察で述べることにする．一方，視差変化のみを持つ刺激の場合は，総重心動揺量の上昇は見られない．この違いは両者の情報を処理する機構が異なっているためではないであろうか．

４．奥行き知覚実験

４．１．目的

重心動揺と奥行き知覚の関係を調べるために，重心動揺測定実験と同じ視覚刺激を用いて，被験者の主観的な奥行き知覚量を測定した．

４．２．方法

刺激の呈示装置は重心動揺測定実験と同じである．測定装置として図８に示すような直線形ポテンショメータからなる奥行き知覚量測定器を用いた．被験者はスクリーンから１ｍの距離に直立姿勢で立ち，体の右側に腰の高さで固定されている奥行き知覚測定器のＡ点とＢ点を持った．スクリーン中心部を見ながらＡ

点を引き出し，視覚刺激の運動平面の知覚的奥行き量とＡＢ点間の距離を合わせ，同じであると思ったところでＡ点についているボタンを押し、このときのポテンショメータの長さを抵抗値の変化で読み取った．奥行き知覚測定器は特に被験者に対し隠されてはいなかったが，その固定場所から実際にはほとんど見えなかった．

被験者は重心動揺測定実験と同じ被験者を含む５名である．奥行き知覚手がかりや運動平面の大きさは重心動揺測定実験と同様であった．奥行き運動の理論的振幅は１０，２０，３０ｃｍの３種類用いた．２７種類の刺激（３[奥行き手がかり]×３[大きさ]×３[振幅]）をそれぞれ６回ランダムに呈示し，応答を採った．呈示時間に制限は設けなかった．

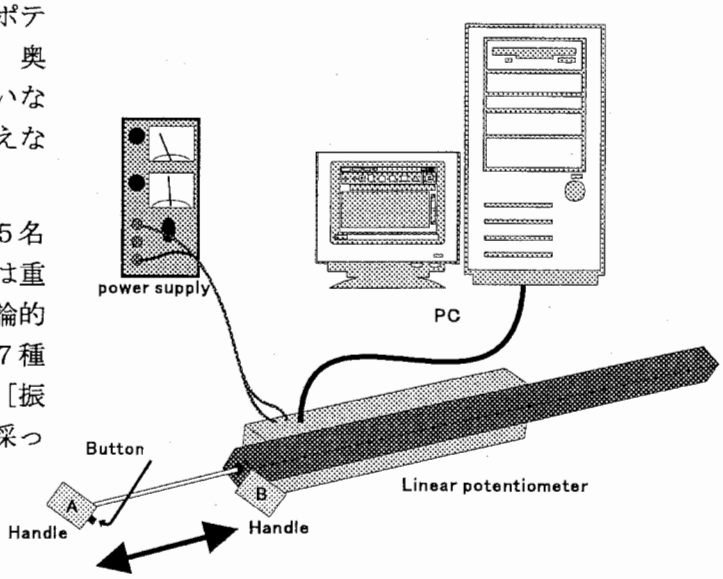

図８　奥行き知覚量測定器

４．３．結果

知覚された奥行きの平均の結果を図９に示す．左から順に刺激に用いた運動平面の理論的振幅が１０，２０，３０ｃｍの時の結果を示し，横の列は運動平面の大きさが上から順にＳ，Ｍ，Ｌの時の結果を示している．横軸は奥行き知覚手がかりで，縦軸は知覚された奥行き量である．

すべての振幅において視差変化と視角変化の両方を持

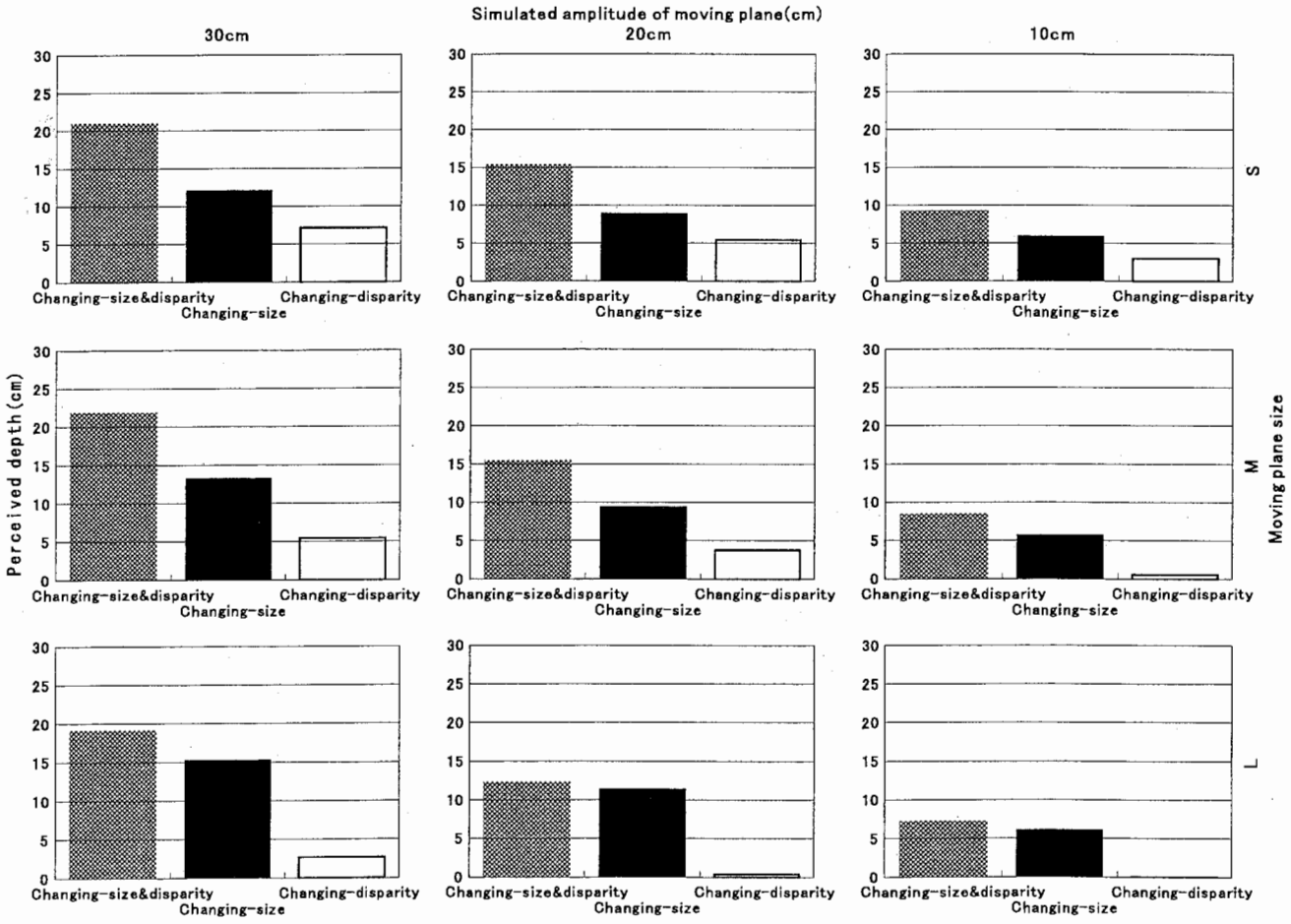

図９　知覚された奥行き量

つ刺激の場合に，もっとも大きく，かつ理論的な振幅に近い奥行きが知覚された．次に視角変化だけを持つ刺激の場合に大きな奥行きが知覚されており，視差変化だけを持つ刺激の場合は，呈示した振幅よりも非常に小さな奥行きしか知覚されなかった．また，運動平面が大きくなるに従って，視差変化のみを持つ刺激の場合は知覚される奥行き変化量が減少していき，視角変化のみを持つ刺激の場合は逆に増加していった．両方を持つ刺激の場合は運動平面の大きさによる変化はほとんどなかった．

４．４．考察

これらの結果から，現実平面をシミュレートした刺激の場合に，知覚される奥行き量が最も大きく，奥行き知覚要因が制限されると知覚される奥行き量が減少することがわかった．特に視差変化のみの画像で運動平面が大きくなるとほとんど奥行きを知覚できない．このことから視差変化によって奥行きを知覚するためには，視差変化のないリファレンスとなる対象が視野内に存在することが必要であると考えられる．

この結果はErkelensらの報告[6)]を裏付けるものである．

５．重心動揺タイミング測定実験

５．１．目的

重心動揺実験１では視覚刺激が同じ運動を繰り返し行うものであったため、刺激の運動と重心動揺との間の位相関係が明らかにならなかった。そこで刺激を受けてから重心動揺が発生するまでの間の潜時や、重心動揺が刺激の位置、速度、加速度のいずれに対応するものであるかを調べるために重心動揺測定実験で用いた視覚刺激に対し、１周期毎に運動平面が停止したインターバルを入れた視覚刺激観察時の重心動揺を測定した。

５．２．方法

視覚刺激は重心動揺実験１と同様で、視差変化と大きさ変化によって奥行き運動を表現した動く平面であった。運動は図１０のように被験者から１１５ｃｍの位置から８５ｃｍの位置まで１秒間で移動し、また１１５ｃｍの位置まで１秒間で戻った後、４秒間静止する。この周期を繰り返す刺激を呈示しながら、重心動揺を測定した。被験者は３名で、各刺激に対して１分間１０回測定を行った。

５．３．結果

その結果を図１１に示す。視角変化の場合、動く平面の大きさによって重心動揺のピークの位置が異なり一定でない。

視差変化の場合、初めの700msは刺激が刺激は被験者に対し前方方向に動いているが重心位置は反対方向に動いている。これは刺激の動きに追従するため、後ろに体を動かそうと力を前向きにかけるからである。次に体の動きにつれて重心が後方へ移動する。１秒の時点で刺激の運動方向が反転し，その後700ms程度遅れて重心が前方に移動し始める．２秒の時点で刺激は停止するが、体を制止させるためにしばらく前方に力をかけつづけるため、重心位置は前方に行き、その後安静時位置へもどる。

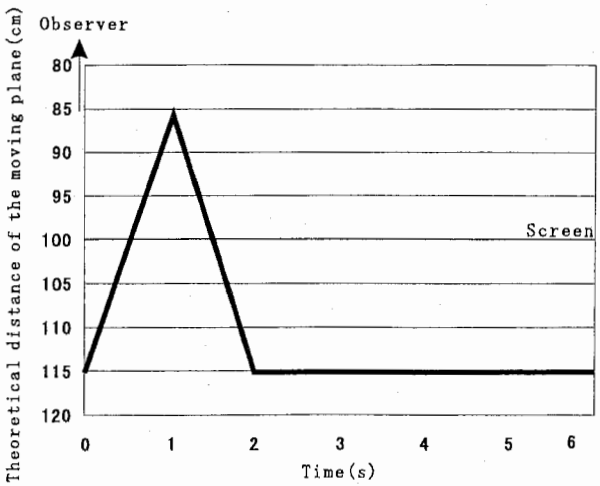

図１０　運動する平面の動き

５．４．考察

視角変化の場合、刺激の運動開始後重心動揺は発生しているので、何らかの影響を受けているとは考えられるが、動揺開始までの潜時が一定でないということは同じメカニズムによって動いているとは考えにくい。視差変化の場合、動く平面の大きさの差による位相のずれは見られない。重心動揺の位相の視覚刺激に対する遅れは700ms程度であったが、最初に身体を刺激の方向に動かすための体の運動の開始までの潜時は200msであり、輻輳眼球運動の潜時とほぼ等しい。これは眼球運動を起こす信号と重心動揺を引き起こす信号が同様のメカニズム

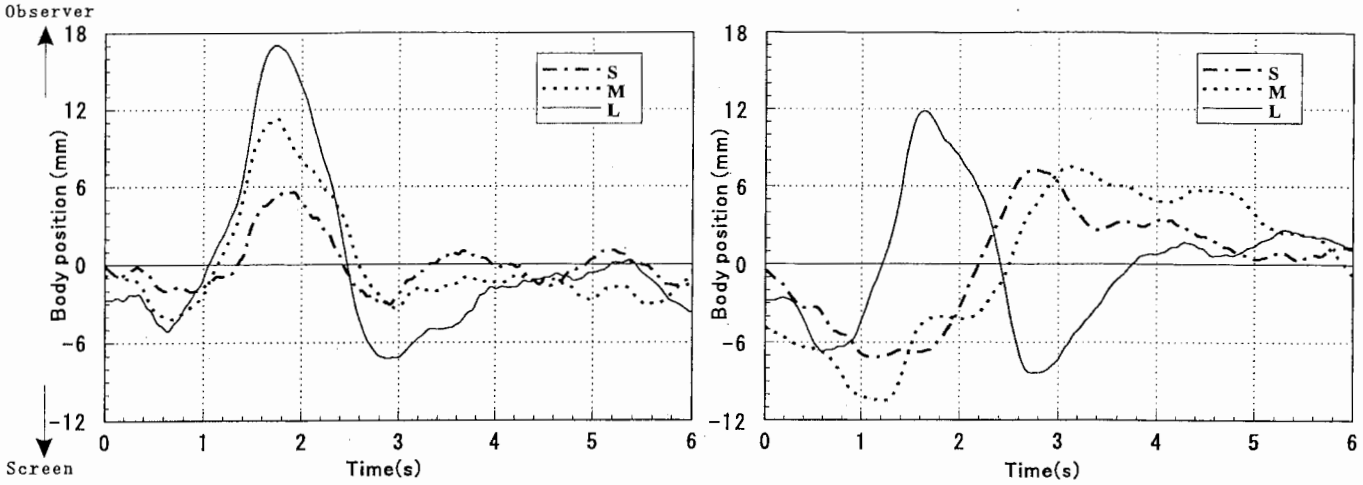

図１１　重心動揺の時間的変化

により発生している可能性を示唆している。

６．重心動揺実験２

６．１．目的

j　重心動揺実験１により、視差変化刺激が重心動揺に強い影響を与えることがわかった。この視差変化刺激をさらに２つの要素に分解し、それぞれの要素が重心動揺にどのような影響を与えるかを調べた。

６．２．方法

視差変化を図１２に示すように（１）網膜像のみの変化（Differential Perspective）¥cite{brian}と（２）輻輳性眼球運動を引き起こす変化に分解した。重心動揺実験１と同様のランダムドットで構成された平面の奥行き運動を、これらの要素それぞれのみで表した刺激画像を作成した。これらの視覚刺激を呈示しながら被験者の重心動揺を測定した。また、differential perspectiveの変化刺激に対して眼球運動がないこと、輻輳変化を引き起こす刺激に対して輻輳性眼球運動が起こることを確認するために、重心動揺の測定とは別に、眼球運動を測定した。測定は被験者３名、各刺激ごとに１分間、１２回ずつ測定した。

６．３．結果

各測定結果を１０秒から約４１秒間を切り出しＦＦＴした結果のパワースペクトルを図１３に示す。どちらの刺激の場合にも、刺激の周波数である0.5Hzの位置にピークが観測された。網膜像のみの変化の場合、すべての被験者において運動する平面が最も大きな時にピークが観測されたが、輻輳変化を起こす刺激の場合は被験者ごとに大きな重心動揺の発生するときの運動する平面の大きさが異なっていた。

各測定結果を１０秒後から５０秒の間を刺激の周期ごとに切り出し平均したものを図１４に示す。網膜像のみの変化する刺激の場合も、すべての被験者において運動する平面が大きな時に重心動揺が見られた。輻輳変化を起こす刺激の場合、すべての被験者において刺激の運動に対応する重心動揺が見られたが、運動する平面の大きさに対する重心動揺の大きさが被験者ごとに異なっていた。

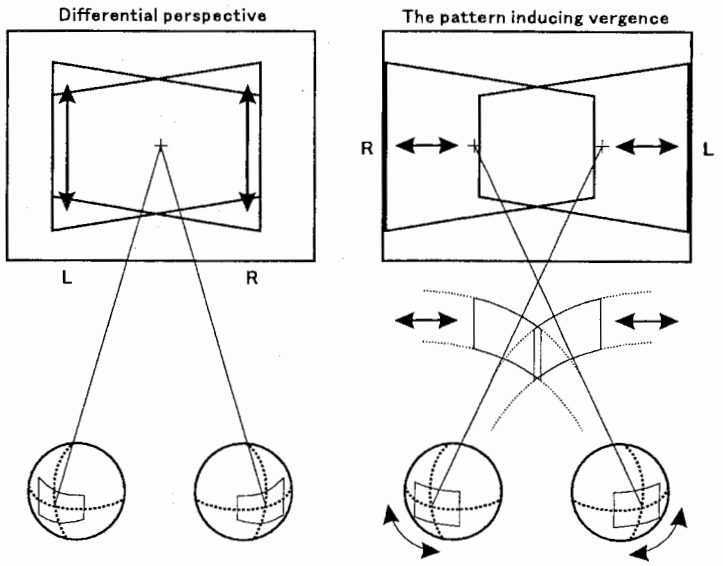

図１２　視差変化の２つの要素

６．４．考察

網膜像の変化する刺激の場合、運動平面の面積が大きいときに刺激の動きに対応した重心動揺がすべての被験者において発生した。

これは広視野の網膜像の変化の情報が姿勢制御に使用されていることを示している。輻輳眼球運動を引き起こす刺激の場合、すべての被験者で重心動揺は発生していたが、重心動揺の発生した運動平面の面積が被験者ごとに異なっていた。これは輻輳眼球運動を引き起こす刺激の情報は、姿勢制御において何らかの関与をしていると考えられるが、網膜像の変化する刺激の情報とは異なるメカニズムで処理されているか、この実験だけでは明らかではない他の情報による影響を受けていることが考えられる。いずれにせよ、この結果だけでは判断できないので、さらに実験を行う必要がある。

7．眼球運動測定実験

7．1．目的

重心動揺実験２において網膜像のみを変化させる刺激と輻輳を発生させる刺激を用いたが、網膜像のみを変化させる刺激の場合に固視が、輻輳を発生させる刺激の場合は輻輳眼球運動が起こっていたかどうかを検証するために眼球運動を測定した。

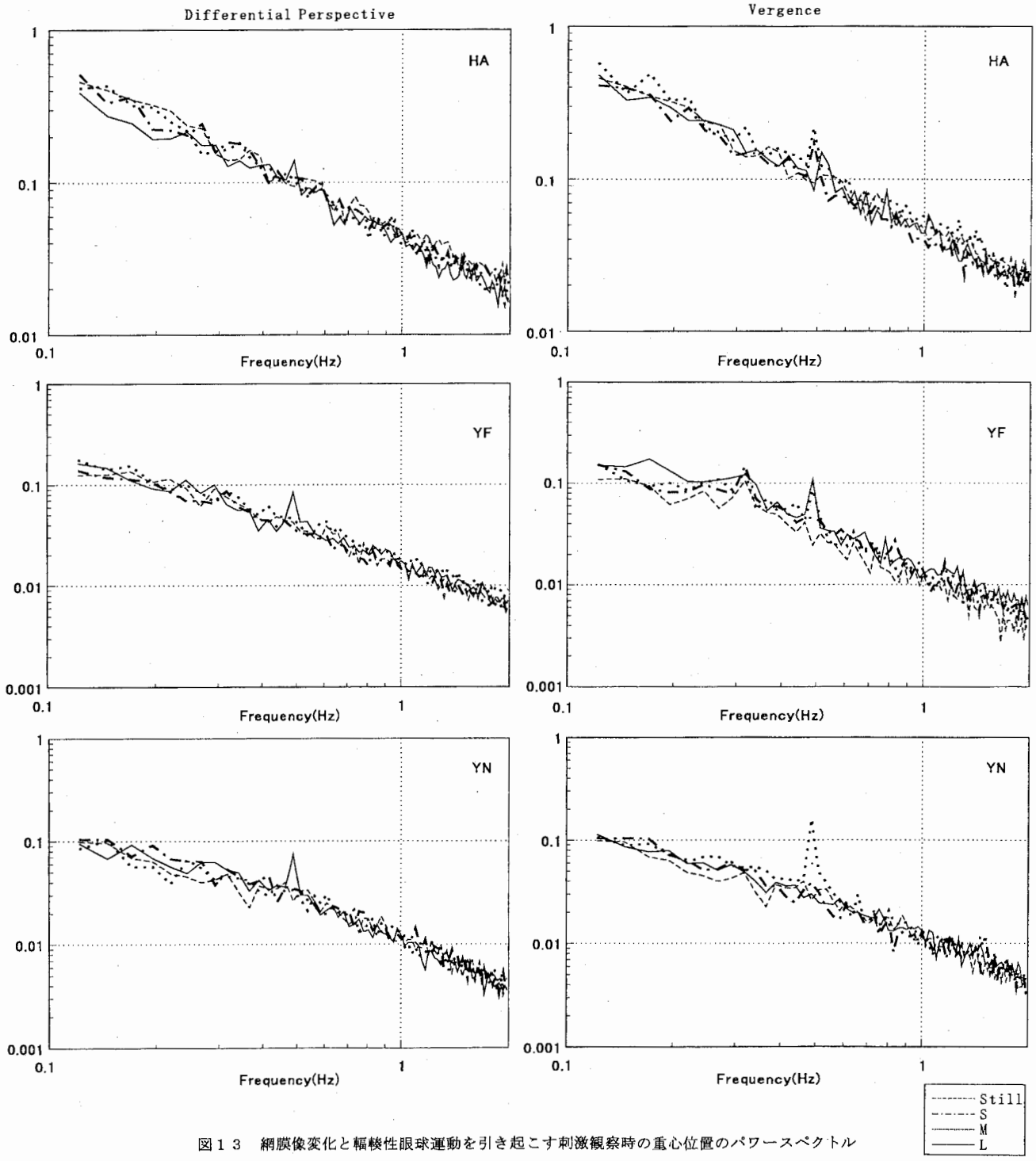

図１３　網膜像変化と輻輳性眼球運動を引き起こす刺激観察時の重心位置のパワースペクトル

７．２．方法

重心動揺実験２で用いた刺激を観察しているときの眼球運動を測定した。被験者は頭部をあご台で固定された。眼球運動測定装置としては竹井機器工業製の強膜反射方式眼球運動測定装置を使用した。被験者１名に対し各刺激ごとに１分間１２回ずつ測定した。

７．３．結果

網膜像のみを変化させる刺激のとき、図１５に示すように眼球運動は発生しなかった。輻輳眼球運動発生刺激のとき眼球運動は全ての刺激に対し発生し、その量や位相に変化はなかった。潜時は200ms程度であった。

７．４．考察

網膜像のみ変化する刺激の場合、眼球運動が起こっていないことが確認できた。したがって、この場合発生した重心動揺は眼球運動とは無関係であるといえるであろう。また、輻輳変化を起こす刺激の場合、常に

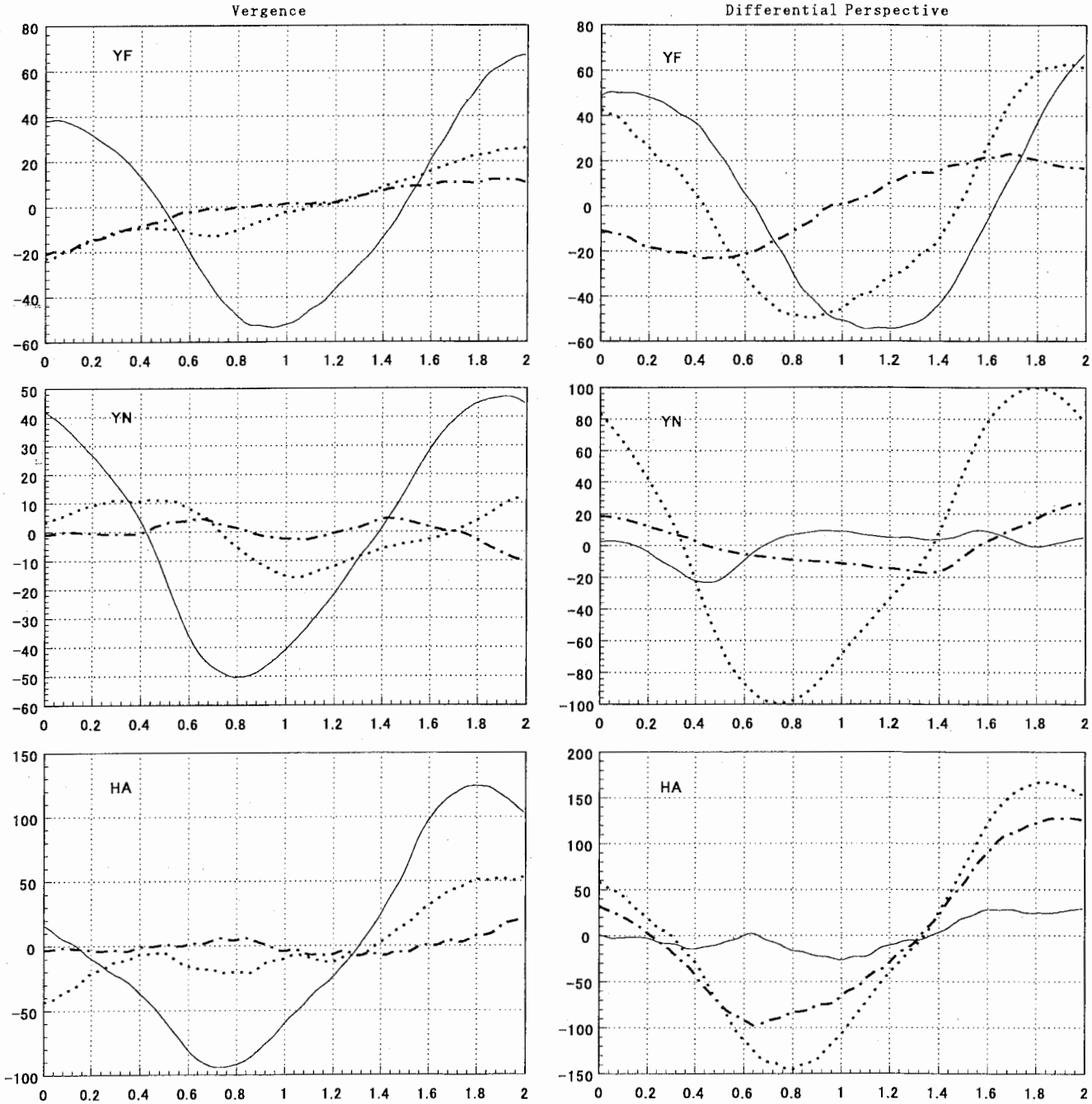

図１４　網膜像変化と輻輳性眼球運動を引き起こす刺激観察時の重心位置の時間的変化

輻輳が発生していたのに対し、重心動揺は必ず発生してはいないため、重心動揺は輻輳によって引き起こすされるものではないと言えるであろう。

8．総合的考察

視角変化のみを持つ刺激の場合，奥行き運動をする運動平面の面積の増加に伴う総重心動揺量の増加する傾向は見られたが，この動揺は運動平面の動きとは相関が見られなかった．また、重心動揺開始までの潜時や運動平面の運動との位相関係なども、運動平面の面積によって異なり一定ではなかった。ところが奥行き知覚実験の結果では，視角変化は知覚される奥行き量に強い影響を与えた．視差変化のみを持つ刺激の場合，運動平面の面積が増加したときに，奥行き運動が重心動揺に対して強い影響を与えた．また、運動平面の面積に関わらず一定の潜時で同様が発生していた。ところが，運動平面の面積が増加するほど，視差変化刺激に対する知覚的奥行き量は減少した．視差変化と視角変化の両方を持つ刺激の場合，刺激の運動に対応する重心動揺は見られたが，視差変化のみを持つ刺激の場合に比べて明確ではなかった．また，刺激の面積が増加するほど総重心動揺量は増加した．奥行きの知覚量は前記の２種類の刺激の場合に比べて大きく，理論的奥行き量にもっとも近かった．このように，共に奥行き変化の情報であると考えられる視差変化と視角変化に対し，重心動揺反応と奥行き運動知覚は大きく異なった．このことから，奥行き方向の姿勢制御のための情報を処理する機構と，奥行きを知覚するための情報を処理する機構は異なっていると考えられる．それでは，奥行き方向の姿勢制御のための情報は何であろうか．もし視差情報だけが姿勢制御に影響するとすれば，視差変化のみの刺激の場合に重心動揺が見られたことは当然の結果といえる．しかし，この仮定では，視角変化のみの刺激の場合に，視差は変化しないにもかかわらず総重心動揺量が増加している理由を説明できない．ここで，考えられる理由のひとつとして以下のものが上げられる．視角変化が奥行き運動知覚に大きく影響することから，知覚的には奥行き運動を感じている．したがって，この知覚的な奥行き運動感が何らかの動作を起こそうとするが，視差は静止状態のままなので，姿勢制御の系は体を動かそうとはしない．これらの矛盾する情報が，体の不規則な運動を起こし総重心動揺量が増加したと考えられる．あるいは，姿勢制御系への視角変化からの信号と，視差からの信号はレベルが違うのかもしれない．前者は比較的高次の過程を経て，後者は低次の過程から知覚と無関係に，姿勢制御系へ信号を直接的に与える．通常，高次の過程からの信号は多くの処理が加わるため不規則になり，低次の過程からの信号は規則的になると考えられる．この考えは今回の実験結果とは矛盾がないと思われる．

また、視差変化をさらに２つの成分に分解した場合に、網膜像だけが変化する刺激の場合、運動平面が大きければ重心動揺が起きた。輻輳性眼球運動を引き起こす刺激においては、輻輳性眼球運動は常に発生していたにもかかわらず、重心動揺の発生は不定で、運動する平面の面積との相関が明確ではなかった。また、重心動揺の潜時が眼球運動の潜時とほとんど同じであることから、眼球運動が重心動揺の直接の原因ではなく、両眼視差の時間的変化のうち周辺視における網膜像の垂直の変化と、眼球運動を引き起こす網膜像の水平の変化が影響しているのではないかと考えられる。しかし、ここで１つ疑問がある。眼球運動は常に起こっていることから眼球運動を引き起こす網膜像の変化は常にあったと考えられるにもかかわらず、重心動揺が常に発生しなかったのはなぜかである。考えられる可能性としてはつぎのようなものがある。輻輳性眼

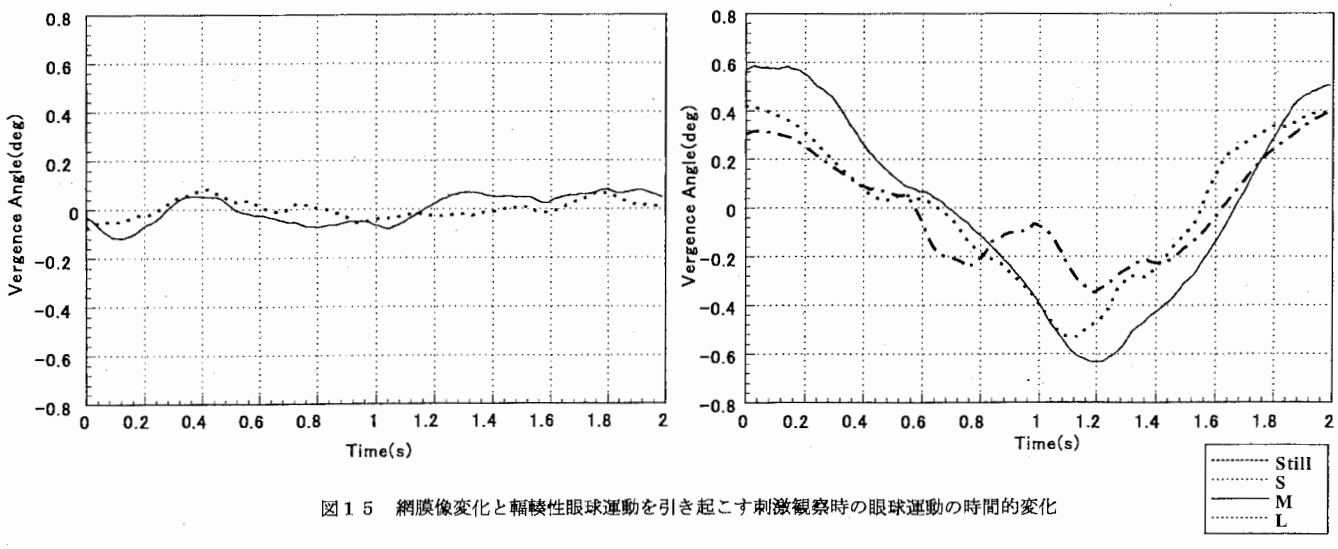

図１５　網膜像変化と輻輳性眼球運動を引き起こす刺激観察時の眼球運動の時間的変化

球運動を引き起こす刺激は眼球が完全に追従すれば網膜像の変化は生まれない。しかし、現実にそのようなことは考えられず刺激の位置と眼球の位置には誤差が生まれ、網膜像に水平方向の運動成分が生じる。この水平成分の量は、眼球運動が非常に安定して起こっていれば一定になるはずであるが、眼球運動に変動があれば変動する。もし、あるレベル以上の水平方向網膜像の変化がないと重心動揺が起こらないとすれば、被験者の眼球運動の追従の仕方によって重心動揺に変化が生まれた可能性がある。もう一つの可能性として、水平方向の網膜像変化は、何かものを見るときに日常的に発生するため、姿勢制御の情報としては信頼の置けるものではないと考えられる。従って、何らかの要因による情報の取捨選択の発生も考えられる。いずれにしても，現段階では明確な説明はなく今後の課題である．

9．むすび

奥行き運動刺激が奥行き運動知覚と姿勢制御に与える影響を調べるため，奥行き知覚情報を制限した奥行き運動刺激を呈示し，重心動揺と知覚された奥行きを，刺激の大きさをパラメータに取り測定した．

その結果，広視野に呈示される両眼視差のみによる奥行き運動刺激が，奥行き知覚にはほとんど影響を与えないことと，姿勢制御に強い影響を与えることが示された．また、眼球運動ではなく網膜像の変化から姿勢制御の情報を得ていることも示された。視角変化のみによる奥行き運動刺激が奥行き知覚に影響を与え，姿勢制御にはあまり影響しないことがわかった．この結果から奥行き知覚のための情報を処理する機構と，姿勢制御のための情報を処理する機構は異なっていることが示唆される．また，知覚の有無にかかわらず重心動揺が生じる場合があるという事実は，臨場感や現実感を生み出すメカニズム，あるいは酔いのメカニズムと深い関係があるものと考えられ，広視野立体表示システムの開発者や，そのようなシステムに提示される立体画像コンテンツの制作者にとっても考慮すべき重要な発見であると考えている．

最後に今後の課題について述べる．本実験により網膜像の変化が重心動揺を引き起こすことがわかったが、特に水平方向の網膜像の変化について、重心動揺への関与の仕方が明かになっていない。これを調べるためには眼球運動と網膜像をより正確に制御して、網膜像の運動量との関連を明らかにすることが，メカニズムを知る上でもそれを応用する上でも重要である．

参考文献

(1) Edwards A.S.: "Body sway and vision", J. exp. Psych., 36, pp.526-535 (1946)

(2) W.M.Paulus, A.Straube and TH.Brandt: "Visual Stabilization of Posture", Brain, 197, 1143-1163 (1984)

(3) A.M.Bronstein, D.Buckwell: "Automatic control of postural sway by visual motion parallax", Exp Brain Res, 113, 243-248 (1997)

(4) 尾島 修一，矢野 澄男: "両眼融合視画像における奥行き感が重心動揺に与える影響"，信学論，J79-A, 2, 354-362 (1996)

(5) 清水 俊宏，三橋 哲雄: "広視野立体画像の提示画角と姿勢制御系における空間認知機構の関係"，信学論，J80-A, 6, 1014-1021 (1997)

(6) C.J.Erkelens and H.Collewijn: "Motion Perception During Dichoptic Viewing of Moving Ramdom-Dot Stereograms", Vision Research, 25, 4, 583-588 (1985)

(7) B.J.Rogers and M.F.Bradshaw: "Disparity scaling and the perception of frontoparallel surfaces",Perception, 24, pp.155-179, (1995)

参考：プログラムリスト

本研究で用いられた呈示刺激を作成するプログラム。１回の実行で１フレームの画像が作成されるので、動画像を作成する場合は、奥行きを変えながら複数回実行する。

ファイル構成
プログラムファイル
generator.c　　メインプログラム
generator.h
brightness.c　　輝度補正プログラム
brightness.h
vsd.c　　垂直視差刺激作成プログラム
vsd.h
savedfm.c　画像データ保存プログラム
savedfm.h

実行時パラメータファイル
parameter.dat　　刺激のパラメータ設定
hwsetup.dat　　ハードウェアのパラメータ設定

実行時データファイル
mask.dat　動く平面の形状データ
position.dat　　ランダムドットの位置データ
intencityL.dat　　プロジェクタの輝度データ左用
intencityR.dat　　プロジェクタの輝度データ右用

メイクファイル（参考：環境によって適宜変更してください）
makefile

画像作成シェルスクリプト（参考：cshell 用）
mkimg1
makeimage

-- generator.c ----------

```
/************************************************
 *        Ver. 1.0   made by Kasahara      *
 *        Ver. 2.0   made by Uwa                    *
 *        1997/10/17 changed by Uwa                 *
 *        1997/11/xx changed by Yamato     *
 *        1998/02/09 changed by Uwa                 *
 *        1998/03/16 changed by Uwa                 *
 *        1998/03/20 changed by Uwa                 *
 *        1998/07/09 changed by Uwa for H.K         *
 *        2000/01/06                                *
 *        Ver. 3.0   made by Uwa                    *
 ************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define    ___PRIVATE___
#include "generator.h"
#include "vsd.h"
#include "savedfm.h"
#include "brightness.h"

#define    FIXINTENCITY         48
#define    DEBUG      0                    /* 0 : OFF , 1 : ON */

int main(int argc,char **argv)
{
          PARAM      para,*paraptr;
          BYTE       *left,*right,*mask;
          FILE       *fp;
          int        i;

          if( argc < 4 ){
                    fprintf(stderr,"Stimulus Generator for Body Sway Exp. on DFM Ver.3.0¥n");
                    fprintf(stderr,"      (2000/01/06)¥n");
                  fprintf(stderr,"Usage: %s  [-m{w|b}] [-n num_of_dots] [-o lroffset]",argv[0]);
                    fprintf(stderr," [-p parameter_file] [-h hardware_setup] [-f fixation_depth]");
                    fprintf(stderr," [-v]");
                    fprintf(stderr," shape_file image_name depth¥n");
                    fprintf(stderr,"      m: marker  w:white or b:black¥n");
                    fprintf(stderr,"      n: a number of dots (default(max.)=5000)¥n");
                    fprintf(stderr,"      o: offset between left image and right image¥n");
                    fprintf(stderr,"      p: parameter file name (default=¥"parameter.dat¥")¥n");
                    fprintf(stderr,"         h: hardware setup file name (default=¥"hwsetup.dat¥")¥n");
                    fprintf(stderr,"         f: depth of fixation point (effected for moving fixation only)¥n");
                    fprintf(stderr,"         v: verbose mode¥n");
                    return 0;
          }
```

```
        /* parameter analysis */
        if( (paraptr = AnalizeArguments(argc,argv,&para)) == NULL ){
                fprintf(stderr,"*** Argument error ***¥n");
                return 1;
        }
        /* Load Intencity Table */
        Lefttable = LoadIntensityProfile("intencityL.dat");
        Righttable = LoadIntensityProfile("intencityR.dat");
        /* create image */
        left = right = mask = NULL;
        if( (left = InitializeImage(LEFT)) == NULL ){
                fprintf(stderr,"*** memory allocatoin error ***¥n");
                return 2;
        }
        if( (right = InitializeImage(RIGHT)) == NULL ){
                fprintf(stderr,"*** memory allocatoin error ***¥n");
                DestroyData(left,right,mask);
                return 2;
        }
        if( paraptr->verbose ){
                printf("Intencity Profile¥n");
                printf("Input Right Left¥n");
                for(i=0;i<256;i++){
                        printf("%3d,  %3d,  %3d¥n",i,RightInverted(i,0),LeftInverted(i,0));
                }
        }
        if( (mask = LoadShapeData(paraptr)) == NULL ){
                fprintf(stderr,"*** cannot make shape data ***¥n");
                DestroyData(left,right,mask);
                return 3;
        }
        DrawImage(left,right,mask,paraptr);
        SaveImage(left,right,paraptr);
        DestroyData(left,right,mask);
        return 0;
}

BYTE* InitializeImage(int left_right)
{
        BYTE    *ptr,*img;
        int     i;
        BYTE    value;
        if( (ptr = (BYTE*)malloc(sizeof(BYTE)*HDWIDTH*HDHEIGHT)) == NULL )
                return NULL;
        img = ptr;

        if (LEFT == left_right)
            value = LeftInverted(0,0);
        else
            value = RightInverted(0,0);

        for(i=0;i<HDWIDTH*HDHEIGHT;i++)
                *ptr++ = value;
        return img;
}

void DestroyData(BYTE* left,BYTE* right,BYTE* mask)
{
        free(left);
        free(right);
        free(mask);
}

PARAM* AnalizeArguments(int argc,char **argv,PARAM* para)
{
        int     a,j,i;
        char    *b;
        char    parafile[64],hwfile[64];

        a = 1;
      b = (char*)&a;
      if( *b )      EndianLong = LittleEndianLong;
      else          EndianLong = BigEndianLong;
        /* Initial value */
        para->lroffset = 0.0;
        para->marker = 0;
        para->dotnum = 5000;
        para->fixdepth = 0.0;
        para->verbose = 0;
        para->imagetype = 0; /* 0:DFM 1:PPM */
        strcpy(parafile,"parameter.dat");
        strcpy(hwfile,"hwsetup.dat");
        j = 0;
        for(i=1;i<argc;i++){
                if( *(argv[i]) == '-' ){
                        switch(*(argv[i]+1)){
```

```
                case 'm':  /* marker */
                        if( *(argv[i]+2) == 'w' ){
                                para->marker = 1;     /* white marker */
                        }
                        else if( *(argv[i]+2) == 'b' ){
                                para->marker = 2;     /* black marker */
                        }
                        else if( *(argv[i]+2) == '¥0' ){
                                i++;
                                if( *(argv[i]) == 'w' ){
                                        para->marker = 1; /* white marker */
                                }
                                else if( *(argv[i]) == 'b' ){
                                        para->marker = 2; /* black marker */
                                }
                        }
                        break;
                case 'n':  /* a number of dots */
                        if( *(argv[i]+2) == '¥0' ){
                                i++;
                                para->dotnum = atoi(argv[i]);
                        }
                        else{
                                para->dotnum = atoi(argv[i]+2);
                        }
                        break;
                case 'o':  /* distance between L and R */
                        if( *(argv[i]+2) == '¥0' ){
                                i++;
                                para->lroffset = atof(argv[i]);
                        }
                        else{
                                para->lroffset = atof(argv[i]+2);
                        }
                        break;
                case 'p':
                        if( *(argv[i]+2) == '¥0' ){
                                i++;
                                strcpy(parafile,argv[i]);
                        }
                        else{
                                if( strcmp((argv[i]+2),"pm") == 0 ){
                                        printf("ppm¥n");
                                        para->imagetype = 1;
                                }
                                else{
                                        strcpy(parafile,(argv[i]+2));
                                }
                        }
                        break;
                case 'h':
                        if( *(argv[i]+2) == '¥0' ){
                                i++;
                                strcpy(hwfile,argv[i]);
                        }
                        else{
                                strcpy(hwfile,(argv[i]+2));
                        }
                        break;
                case 'f':  /* depth for fixation point */
                        if( *(argv[i]+2) == '¥0' ){
                                i++;
                                para->fixdepth = atof(argv[i]);
                        }
                        else{
                                para->fixdepth = atof(argv[i]+2);
                        }
                        break;
                case 'v':  /* verbose mode */
                        para->verbose = 1;
                        break;
                default:
                        fprintf(stderr,"*** invalid argument -%c ***¥n",
                                *(argv[i]+1));
                        j++;
                }
        }
        else{
                switch(j){
                case 0:
                        strcpy(para->maskfilename,argv[i]);
                        break;
                case 1:
                        strcpy(para->imagename,argv[i]);
                        break;
                case 2:
```

```
                                        para->depth = atof(argv[i]);
                                        break;
                                }
                                j++;
                        }
                }
                LoadParameterFile(parafile,para);
                LoadSetupFile(hwfile,para);
                return para;
}

void LoadParameterFile(char *filename,PARAM *para)
{
        FILE    *fp;
        int     i,j,flag;
        char    *ptr,*pp,ebuf[128],buf[256];

        if( (fp = fopen(filename,"r")) == NULL ){
                fprintf(stderr,"*** file not found: parameter file ***¥n");
                return;
        }
        /* Initialize */
        para->reference = ON;
        para->variance = BUNSAN;
        /* --- */
        while( fgets(buf,255,fp) != NULL ){
                ptr = buf;
                while((*ptr == ' ') || (*ptr == '¥t'))
                        ptr++;
                if( *ptr == '#' )       continue;
                i = 0;
                while( Pterm[i] != NULL ){
                        if( !strncmp(ptr,Pterm[i],strlen(Pterm[i])) ){
                                ptr += strlen(Pterm[i]);
                                while((*ptr == ' ') || (*ptr == '¥t'))
                                        ptr++;
                                pp = ptr;
                                while( *pp != '¥0' ){/* LF to NULL */
                                        if( *pp == '¥n' ){
                                                *pp = '¥0';
                                                break;
                                        }
                                        pp++;
                                }
                                switch(i){
                                case 0:     /* POSITION FILE */
                                        strcpy(para->posfilename,ptr);
                                        break;
                                case 1:     /* DISTANCE EYE */
                                        para->eye = atof(ptr);
                                        break;
                                case 2:     /* REFERENCE */
                                        para->reference = !strcmp("ON",ptr);
                                        break;
                                case 3:     /* DOT SIZE */
                                        para->sdotsize = atoi(ptr);
                                        break;
                                case 4:     /* DISPARITY */
                                        pp = ptr;
                                        para->disparity = 0;
                                        do{
                                                j = 0;
                                                while( *pp != '|' && *pp != '¥0' ){
                                                        if( *pp != ' ' && *pp != '¥t' ){
                                                                ebuf[j++] = *pp;
                                                        }
                                                        pp++;
                                                }
                                                ebuf[j] = '¥0';
                                                flag = 0x01;
                                                j = 0;
                                                while( Dterm[j] != NULL ){
                                                        if( !strcmp(ebuf,Dterm[j]) ){
                                                                para->disparity |= flag;
                                                        }
                                                        flag <<= 1;
                                                        j++;
                                                }
                                                if( *pp == '|' )        pp++;
                                        }while( *pp != '¥0' );
                                        break;
                                case 5:     /* CHANGING SIZE */
                                        para->valsize = !strcmp("ON",ptr);
                                        break;
                                case 6:     /* FIXATION */
                                        if( !strcmp("M",ptr) )
```

```
                                        para->fixation = 'm';
                                else if( !strcmp("F",ptr) )
                                        para->fixation = 'f';
                                else    para->fixation = 0;
                                break;
                        case 7:     /* BUNSAN */
                                para->variance = atof(ptr);
                                break;
                        }
                    }
                    i++;
                }
        }
        fclose(fp);
        Variance = para->variance;
}

void LoadSetupFile(char *filename,PARAM *para)
{
        FILE    *fp;
        int     i,j;
        char    *ptr,*pp,ebuf[128],buf[256];

        if( (fp = fopen(filename,"r")) == NULL ){
                fprintf(stderr,"*** file not found: hardware setup file ***¥n");
                return;
        }
        while( fgets(buf,255,fp) != NULL ){
                ptr = buf;
                while((*ptr == ' ') || (*ptr == '¥t'))
                        ptr++;
                if( *ptr == '#' )    continue;
                i = 0;
                while( Hterm[i] != NULL ){
                        if( !strncmp(ptr,Hterm[i],strlen(Hterm[i])) ){
                                ptr += strlen(Hterm[i]);
                                while((*ptr == ' ') || (*ptr == '¥t'))
                                        ptr++;
                                pp = ptr;
                                while( *pp != '¥0' ){/* LF to NULL */
                                        if( *pp == '¥n' ){
                                                *pp = '¥0';
                                                break;
                                        }
                                        pp++;
                                }
                                switch(i){
                                case 0:     /* SCREEN WIDTH */
                                        para->swidth = atof(ptr);
                                        break;
                                case 1:     /* SCREEN HEIGHT */
                                        para->sheight = atof(ptr);
                                        break;
                                case 2:     /* SCREEN W PIXELS */
                                        para->swpix = atoi(ptr);
                                        break;
                                case 3:     /* SCREEN H PIXELS */
                                        para->shpix = atoi(ptr);
                                        break;
                                case 4:     /* OFFSET X */
                                        para->offsetx = atoi(ptr);
                                        break;
                                case 5:     /* OFFSET Y */
                                        para->offsety = atoi(ptr);
                                        break;
                                case 6:     /* VIEWING DISTANCE */
                                        para->distance = atof(ptr);
                                        break;
                                }
                        }
                        i++;
                }
        }
        fclose(fp);
}

BYTE* LoadShapeData(PARAM* paraptr)
{
        BYTE    *ptr,*p;
        FILE    *fp;
        char    check,buf[256];
        int     x,y,xs,ys,width,height;

        if( (fp = fopen(paraptr->maskfilename,"rb")) == NULL ){
                fprintf(stderr,"*** file not found ***:%s¥n",paraptr->maskfilename);
                return NULL;
```

```
        }
        if( (ptr = (BYTE*)malloc(sizeof(BYTE)*HDWIDTH*HDHEIGHT)) == NULL ){
                fclose(fp);
                return NULL;
        }
        check = fgetc(fp);
        if( check == '#' ){
                fgets(buf,255,fp);
                width = atoi(buf);
                fgets(buf,255,fp);
                height = atoi(buf);
                p = ptr;
                xs = (HDWIDTH-width) / 2;
                ys = (HDHEIGHT-height) / 2;
                for(y=0;y<HDHEIGHT;y++){
                        for(x=0;x<HDWIDTH;x++){
                                if( y>=ys && y<ys+height && x>=xs && x<xs+width )
                                        *p++ = ON;
                                else    *p++ = OFF;
                        }
                }
        }
        else{
                *ptr = check;
                fread(ptr+1,HDWIDTH*sizeof(BYTE)*HDHEIGHT-1,1,fp);
        }
        fclose(fp);
        return ptr;
}

void DrawImage(BYTE* left,BYTE* right,BYTE* mask,PARAM* paraptr)
{
        double  depth,x,y,rx,ry,lx,ly,dx,dy;
        int     i,ix,iy,dnum,number,val,xi,yi;
        long    *posptr;
        BYTE    *dottable;

        if( paraptr->verbose )
                printf("disparity type = %d¥n",paraptr->disparity);
        Parallax = DisparityFunc[paraptr->disparity];
        depth = paraptr->depth;
        posptr = LoadPositionData(paraptr->posfilename,&number);
        dottable = LoadDotTable(paraptr);
        dnum = 0;
        if( paraptr->verbose )
                printf("number = %d¥n",number);
        for(i=0;i<number;i++){
                x = ((double)(*EndianLong)(*posptr++)) / 100.0;
                y = ((double)(*EndianLong)(*posptr++)) / 100.0;
                dx = x - (paraptr->swpix) / 2.0 + (paraptr->offsetx);
                dy = y - (paraptr->shpix) / 2.0 + (paraptr->offsety);
                if( dx*dx+dy*dy < 1800 )        continue;
                ix = (int)x;
                iy = (int)y;
                if( paraptr->valsize == ON ){
                        (paraptr->cdotsize)[0] = CalculateDotSize(paraptr,depth);
                        (paraptr->cdotsize)[1] = (paraptr->cdotsize)[0];
                }
                else{
                        (paraptr->cdotsize)[0] = paraptr->sdotsize;
                        (paraptr->cdotsize)[1] = (paraptr->cdotsize)[0];
                }
                if( mask[iy*HDWIDTH+ix] ){
                        (*Parallax)(x,y,depth,&rx,&ry,&lx,&ly,paraptr);
                        lx -= (paraptr->lroffset / 2.0);
                        rx += (paraptr->lroffset / 2.0);
                        if( lx >= 0.0 && rx < HDWIDTH && ly >= 0.0
                         && ry >= 0.0 && ly < HDHEIGHT && ry < HDHEIGHT ){
                                DrawPoint(left,lx,ly,dottable,(paraptr->cdotsize)[0],paraptr,LeftInverted);
                                DrawPoint(right,rx,ry,dottable,(paraptr->cdotsize)[1],paraptr,RightInverted);
                                dnum++;    /* count a number of dot */
                                if( dnum >= paraptr->dotnum )   break;
                        }
                }
                else if( paraptr->reference == ON ){
                        lx = x - (paraptr->lroffset / 2.0);
                        rx = x + (paraptr->lroffset / 2.0);
                        if( lx >= 0.0 && rx < HDWIDTH ){
                                DrawPoint(left,lx,y,dottable,paraptr->sdotsize,paraptr,LeftInverted);
                                DrawPoint(right,rx,y,dottable,paraptr->sdotsize,paraptr,RightInverted);
                        }
                }
        }
        if( paraptr->fixation ){
                DrawFixation(paraptr,depth,left,right);
        }
```

```
        if( paraptr->marker ){
                if( paraptr->marker == 1 )      val = 255;
                else                                    val = 0;
                for(yi=0;yi<35;yi++){
                        for(xi=0;xi<40;xi++){
                                left[yi*HDWIDTH+xi] = val;
                                right[yi*HDWIDTH+xi] = val;
                        }
                }
        }
}

void DrawFixation(PARAM* paraptr,double depth,BYTE* left,BYTE* right)
{
        double  x,y,rx,ry,lx,ly;
        int     i,w,h;

        x = (int)((paraptr->swpix) / 2.0 + (paraptr->offsetx));
        y = (int)((paraptr->shpix) / 2.0 + (paraptr->offsety));
        if( paraptr->fixation == 'm' ){
                if( paraptr->verbose ){
                        printf("moved fixation point %lf¥n",depth);
                        printf("fixation depth = %lf¥n",paraptr->fixdepth);
                }
                paraptr->disparity = 0x0B;
                if( paraptr->fixdepth == 0.0 ){
                        HorizontalParallax(x,y,depth,&rx,&ry,&lx,&ly,paraptr);
                }
                else{
                        HorizontalParallax(x,y,paraptr->fixdepth,&rx,&ry,&lx,&ly,paraptr);
                }
                if( paraptr->verbose )          printf("%lf %lf %lf %lf¥n",rx,ry,lx,ly);
        }
        else if( paraptr->fixation == 'f' ){
                if( paraptr->verbose )          printf("fixed center point¥n");
                ry = ly = y;
                rx = lx = x;
        }
        w = 15;
        h = 20;
#if DEBUG
        printf("lx = %lf,  rx = %lf¥n",lx,rx);
#endif
        for(i=-w;i<=w;i++){
                DrawPixel(lx+(double)i,ly,FIXINTENCITY/2,left,LeftInverted);
                DrawPixel(rx+(double)i,ry,FIXINTENCITY/2,right,RightInverted);
                DrawPixel(lx+(double)i,ly+1,FIXINTENCITY/4,left,LeftInverted);
                DrawPixel(rx+(double)i,ry+1,FIXINTENCITY/4,right,RightInverted);
                DrawPixel(lx+(double)i,ly-1,FIXINTENCITY/4,left,LeftInverted);
                DrawPixel(rx+(double)i,ry-1,FIXINTENCITY/4,right,RightInverted);
        }
        for(i=-h;i<=h;i++){
#if DEBUG
                printf("left¥n");
#endif
                DrawSpreadPixel(lx-w,ly+(double)i,FIXINTENCITY/2,left,LeftInverted);
                DrawSpreadPixel(lx+w,ly+(double)i,FIXINTENCITY/2,left,LeftInverted);
#if DEBUG
                printf("right¥n");
#endif
                DrawSpreadPixel(rx-w,ry+(double)i,FIXINTENCITY/2,right,RightInverted);
                DrawSpreadPixel(rx+w,ry+(double)i,FIXINTENCITY/2,right,RightInverted);
                if( i < -5 ){
                        DrawSpreadPixel(lx,ly+(double)i,FIXINTENCITY,left,LeftInverted);
                }
                else if( i > 5 ){
                        DrawSpreadPixel(rx,ry+(double)i,FIXINTENCITY,right,RightInverted);
                }
                if( i > -4 && i < 4 ){
                        DrawSpreadPixel(lx,ly+(double)i,FIXINTENCITY/2,left,LeftInverted);
                        DrawSpreadPixel(rx,ry+(double)i,FIXINTENCITY/2,right,RightInverted);
                }
        }
}

void DrawSpreadPixel(double x,double y,int val,BYTE *img,int (*func)(int,int))
{
        DrawPixel(x,y,val,img,func);
        DrawPixel(x-1,y,val/1.5,img,func);
        DrawPixel(x+1,y,val/1.5,img,func);
        DrawPixel(x-2,y,val/3,img,func);
        DrawPixel(x+2,y,val/3,img,func);
}

void DrawPixel(double x,double y,int val,BYTE *img,int (*func)(int,int))
{
```

```
        int         ix, iy, val1, val2, val3, val4;
        double      dx, dy, dx2, dy2, r1, r2, r3, r4;

        ix = (int)x;
        iy = (int)y;
        dx = x - (double)ix;
        dy = y - (double)iy;
        dx2 = 1.0 - dx;
        dy2 = 1.0 - dy;
        r1 = dx2 * dy2;
        r2 = dx * dy2;
        r3 = dx2 * dy;
        r4 = dx * dy;
#if DEBUG
        printf("%lf %lf %lf %lf¥n", r1, r2, r3, r4);
#endif
        val1 = (int)(r1 * (double)val + 0.5);
        val2 = (int)(r2 * (double)val + 0.5);
        val3 = (int)(r3 * (double)val + 0.5);
        val4 = (int)(r4 * (double)val + 0.5);
#if DEBUG
        printf("%d %d %d %d : %d %d %d %d¥n", val1, val2, val3, val4, (*func)(val1, 0), (*func)(val2, 0), (*func)(val3, 0), (*func)(
val4, 0));
#endif
        PutValue(ix  , iy  , val1, img, func);
        PutValue(ix+1, iy  , val2, img, func);
        PutValue(ix  , iy+1, val3, img, func);
        PutValue(ix+1, iy+1, val4, img, func);
}

long* LoadPositionData(char *filename, int *number)
{
        char    *buffer;
        long    num;
        FILE    *fp;

        *number = 0;
        if( (fp = fopen(filename, "rb")) == NULL ){
                fprintf(stderr, "*** file not found ***:%s¥n", filename);
                return NULL;
        }
        if( fread((char*)&num, sizeof(long), 1, fp) == 0 ){
                fprintf(stderr, "*** file read error ***¥n");
                fclose(fp);
                return NULL;
        }
        num = (*EndianLong)(num);
        if( num == 0 ){
                fprintf(stderr, "*** invalid data ***¥n");
                fclose(fp);
                return NULL;
        }
        if( (buffer = (char*)malloc(sizeof(long)*num*2)) == NULL ){
                fprintf(stderr, "*** memory allocation error:LoadPositionData ***¥n");
                fclose(fp);
                return NULL;
        }
        if( fread(buffer, sizeof(long), num*2, fp) == 0 ){
                fprintf(stderr, "*** file read error ***¥n");
                free(buffer);
                fclose(fp);
                return NULL;
        }
        fclose(fp);
        *number = num;
        return (long*)buffer;
}

BYTE* LoadDotTable(PARAM *paraptr)
{
        long    size;
        BYTE    *data;

        size = (paraptr->sdotsize)*SAMPLERATE;
        if( (data = (BYTE*)malloc(size*sizeof(BYTE))) == NULL ){
                fprintf(stderr, "*** memory allocation error:LoadDotTable ***¥n");
                return NULL;
        }
        CreateDotTable(data, paraptr->sdotsize);
        return data;
}

void CreateDotTable(BYTE *dot, int dotsize)
{
        double  z;
        int     x, intency;
```

```
        for(x=0;x<(dotsize*SAMPLERATE)/2;x++){
                /* z = 7.5 * (x/(double)(dotsize*SAMPLERATE)); 7.5???*/
                z = ((double)dotsize/2.0) * (x/(double)(dotsize*SAMPLERATE));
                intency = (int)(q_nor(z) * 256.0);
                if( intency > 255 )   intency = 255;
                if( intency < 0 )     intency = 0;
                *dot++ = intency;
        }
}

double q_nor(double z)
{
    return exp(-z * z / (2*Variance));
}

double p_nor(double z)
{
    return q_nor(z);
}

double CalculateDotSize(PARAM* paraptr,double depth)
{
        return (double)(paraptr->sdotsize)/depth*(paraptr->distance);
}

void HorizontalParallax(double x,double y,double z,
            double *rx,double *ry,double *lx,double *ly,PARAM* paraptr)
{
        double     a,r,hswp,hshp;

        hswp = (paraptr->swpix)/2.0;
        hshp = (paraptr->shpix)/2.0;
        if( paraptr->disparity & 0x01 == 0 ){
                if( paraptr->valsize == ON ){
                        x -= paraptr->offsetx;
                        y -= paraptr->offsety;
                        x = (x-hswp) * (paraptr->swidth) / (paraptr->swpix);
                        y = (y-hshp) * (paraptr->sheight) / (paraptr->shpix);
                        *rx = (paraptr->distance) / z * x;
                        *lx = *rx = *rx * (paraptr->swpix)/(paraptr->swidth) + hswp + paraptr->offsetx;
                        *ry = (paraptr->distance) / z * y;
                        *ry = *ry * (paraptr->shpix)/(paraptr->sheight) + hshp + paraptr->offsety;
                        *ly = *ry;
                }
                else{
                        *ly = *ry = y;
                        *lx = *rx = x;
                }
        }
        else{
                if( paraptr->valsize == ON ){
                        x -= paraptr->offsetx;
                        y -= paraptr->offsety;
                        x = (x-hswp) * (paraptr->swidth) / (paraptr->swpix);
                        y = (y-hshp) * (paraptr->sheight) / (paraptr->shpix);
                        *rx = (paraptr->distance) / z * (x-(paraptr->eye)/2.0) + (paraptr->eye)/2.0;
                        *rx = *rx * (paraptr->swpix)/(paraptr->swidth) + hswp + paraptr->offsetx;
                        *lx = (paraptr->distance) / z * (x+(paraptr->eye)/2.0) - (paraptr->eye)/2.0;
                        *lx = *lx * (paraptr->swpix)/(paraptr->swidth) + hswp + paraptr->offsetx;
                        *ry = (paraptr->distance) / z * y;
                        *ry = *ry * (paraptr->shpix)/(paraptr->sheight) + hshp + paraptr->offsety;
                        *ly = *ry;
                }
                else{
                        x -= paraptr->offsetx;
                        x = (x-hswp) * (paraptr->swidth) / (paraptr->swpix);
                        r = z / (paraptr->distance);
                        a = (1-r)/r  * paraptr->eye;    /* a < 0 far a > near */
                        *ly = *ry = y;
                        a = a / 2.0;
                        *rx = x - a;
                        *lx = x + a;
                        *rx = *rx * (paraptr->swpix)/(paraptr->swidth) + hswp + paraptr->offsetx;
                        *lx = *lx * (paraptr->swpix)/(paraptr->swidth) + hswp + paraptr->offsetx;
                }
        }
}

void CalcHorizontalShearDisparity(double x,double y,double angle,
            double *rx,double *ry,double *lx,double *ly,PARAM* paraptr)
{
        double     x0,y0;

        x0 = (x - paraptr->offsetx)-(paraptr->swpix)/2.0;
        y0 = (y - paraptr->offsety)-(paraptr->shpix)/2.0;
```

```
        x0 = x0 * (paraptr->swidth) / (paraptr->swpix);
        y0 = y0 * (paraptr->sheight) / (paraptr->shpix);
        *rx = x0 + y0 * tan(angle*M_PI/180.0);
        *lx = x0 - y0 * tan(angle*M_PI/180.0);
        *ry = *ly = y0;
        *lx = *lx * ((paraptr->swpix)/(paraptr->swidth));
        *lx = *lx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ly = *ly * ((paraptr->shpix)/(paraptr->sheight));
        *ly = *ly + paraptr->offsety + (paraptr->shpix)/2.0;
        *rx = *rx * ((paraptr->swpix)/(paraptr->swidth));
        *rx = *rx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ry = *ry * ((paraptr->shpix)/(paraptr->sheight));
        *ry = *ry + paraptr->offsety + (paraptr->shpix)/2.0;
}

void CalcVerticalShearDisparity(double x,double y,double angle,
                double *rx,double *ry,double *lx,double *ly,PARAM* paraptr)
{
        double    x0,y0;

        x0 = (x - paraptr->offsetx)-(paraptr->swpix)/2.0;
        y0 = (y - paraptr->offsety)-(paraptr->shpix)/2.0;
        x0 = x0 * (paraptr->swidth) / (paraptr->swpix);
        y0 = y0 * (paraptr->sheight) / (paraptr->shpix);
        *rx = *lx = x0;
        *ry = y0 - x0 * tan(angle*M_PI/180.0);
        *ly = y0 + x0 * tan(angle*M_PI/180.0);
        *lx = *lx * ((paraptr->swpix)/(paraptr->swidth));
        *lx = *lx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ly = *ly * ((paraptr->shpix)/(paraptr->sheight));
        *ly = *ly + paraptr->offsety + (paraptr->shpix)/2.0;
        *rx = *rx * ((paraptr->swpix)/(paraptr->swidth));
        *rx = *rx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ry = *ry * ((paraptr->shpix)/(paraptr->sheight));
        *ry = *ry + paraptr->offsety + (paraptr->shpix)/2.0;
}

void CalcVandHShearDisparity(double x,double y,double angle,
                double *rx,double *ry,double *lx,double *ly,PARAM* paraptr)
{
        double    x0,y0;

        x0 = (x - paraptr->offsetx)-(paraptr->swpix)/2.0;
        y0 = (y - paraptr->offsety)-(paraptr->shpix)/2.0;
        x0 = x0 * (paraptr->swidth) / (paraptr->swpix);
        y0 = y0 * (paraptr->sheight) / (paraptr->shpix);
        *rx = x0 + y0 * tan(angle*M_PI/180.0);
        *lx = x0 - y0 * tan(angle*M_PI/180.0);
        *ry = y0 - x0 * tan(angle*M_PI/180.0);
        *ly = y0 + x0 * tan(angle*M_PI/180.0);
        *lx = *lx * ((paraptr->swpix)/(paraptr->swidth));
        *lx = *lx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ly = *ly * ((paraptr->shpix)/(paraptr->sheight));
        *ly = *ly + paraptr->offsety + (paraptr->shpix)/2.0;
        *rx = *rx * ((paraptr->swpix)/(paraptr->swidth));
        *rx = *rx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ry = *ry * ((paraptr->shpix)/(paraptr->sheight));
        *ry = *ry + paraptr->offsety + (paraptr->shpix)/2.0;
}

void CalcVergence(double x,double y,double z,double *rx,double *ry,double *lx,double *ly,PARAM* paraptr)
{
        double    x0,y0,z0,x1,z1,r,a,angle;

        /* normalize */
        x0 = (x - paraptr->offsetx)-(paraptr->swpix)/2.0;
        y0 = (y - paraptr->offsety)-(paraptr->shpix)/2.0;
        x0 = x0 * (paraptr->swidth) / (paraptr->swpix);
        y0 = y0 * (paraptr->sheight) / (paraptr->shpix);
        r = z / (paraptr->distance);
        a = (1-r)/r  * paraptr->eye;
        a /= 2.0;
        angle = atan2(paraptr->distance,paraptr->eye/2.0) - atan2((paraptr->distance),(paraptr->eye/2.0)+a);
        x1 = x0 * cos(-angle);
        z0 = -x0 * sin(-angle);
        z1 = paraptr->distance - z0;
        (paraptr->cdotsize)[0] = (paraptr->cdotsize)[0] * (paraptr->distance) / z1;
        *lx = (paraptr->distance) / z1 * x1 + a;
        *lx = *lx * ((paraptr->swpix)/(paraptr->swidth));
        *lx = *lx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ly = (paraptr->distance) / z1 * y0;
        *ly = *ly * ((paraptr->shpix)/(paraptr->sheight));
        *ly = *ly + paraptr->offsety + (paraptr->shpix)/2.0;
        x1 = x0 * cos(angle);
        z0 = -x0 * sin(angle);
        z1 = paraptr->distance - z0;
```

```
        (paraptr->cdotsize)[1] = (paraptr->cdotsize)[1] * (paraptr->distance) / z1;
        *rx = (paraptr->distance) / z1 * x1 - a;
        *rx = *rx * ((paraptr->swpix)/(paraptr->swidth));
        *rx = *rx + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ry = (paraptr->distance) / z1 * y0;
        *ry = *ry * ((paraptr->shpix)/(paraptr->sheight));
        *ry = *ry + paraptr->offsety + (paraptr->shpix)/2.0;
}

void AfinTransfer(double xi,double yi,double zi,double *xo,double *yo,PARAM* paraptr,int flag)
{
        double    angle,x,z;
        static double       angle2 = 0;

        xi = xi * (paraptr->swidth) / (paraptr->swpix);
        yi = yi * (paraptr->sheight) / (paraptr->shpix);
        angle = atan2(paraptr->distance,paraptr->eye/2.0) - atan2((paraptr->distance)-zi,paraptr->eye/2.0);
        x = xi * cos(angle*flag); /* + (zi) * sin(angle); */
        z = -xi * sin(angle*flag); /* + (zi) * cos(angle); */
        z = paraptr->distance - z;
        *xo = (paraptr->distance) / z * x;
        *yo = (paraptr->distance) / z * yi;
        angle2 = angle;
        if( flag == 1 )
                (paraptr->cdotsize)[0] = (paraptr->cdotsize)[0] * (paraptr->distance) / z;
        else    (paraptr->cdotsize)[1] = (paraptr->cdotsize)[1] * (paraptr->distance) / z;
        *xo = (*xo) * ((paraptr->swpix)/(paraptr->swidth));
        *yo *= ((paraptr->shpix)/(paraptr->sheight));
}

void CalcDisparityField(double x,double y,double z,
                double *rx,double *ry,double *lx,double *ly,PARAM* paraptr)
{
        double    x0,y0,x1,xr,y1,yr;

        z = paraptr->distance - z;
        x0 = (x - paraptr->offsetx)-(paraptr->swpix)/2.0;
        y0 = (y - paraptr->offsety)-(paraptr->shpix)/2.0;
        AfinTransfer(x0,y0,z,&x1,&y1,paraptr,1);
        *lx = x1 + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ly = y1 + paraptr->offsety + (paraptr->shpix)/2.0;
        AfinTransfer(x0,y0,z,&x1,&y1,paraptr,-1);
        *rx = x1 + paraptr->offsetx + (paraptr->swpix)/2.0;
        *ry = y1 + paraptr->offsety + (paraptr->shpix)/2.0;
}

void DrawPoint(BYTE *img,double dx,double dy,BYTE *dottable,
                double dotsize,PARAM* paraptr,int (*func)(int,int))
{
        BYTE      val;
        int       ix,iy,six,eix,siy,eiy;
        double    distance,temp;

        six = (int)(dx - dotsize / 2.0);
        siy = (int)(dy - dotsize / 2.0);
        eix = (int)(dx + dotsize / 2.0) + 1;
        eiy = (int)(dy + dotsize / 2.0) + 1;
        for(iy=siy;iy<=eiy;iy++){
                if( iy < 0 )                    continue;
                if( iy >= HDHEIGHT ) break;
                temp = ((double)iy-dy)*((double)iy-dy);
                for(ix=six;ix<=eix;ix++){
                        if( ix < 0 )                    continue;
                        if( ix >= HDWIDTH )   break;
                        distance = sqrt(((double)ix-dx)*((double)ix-dx) + temp);
                        if( distance <= (dotsize/2.0) ){
                                distance = distance / dotsize * paraptr->sdotsize;
                                val = GetIntencity(distance,dottable);
                                PutValue(ix,iy,val,img,func);
                        }
                }
        }
}

BYTE GetIntencity(double distance,BYTE* dottable)
{
        return dottable[(int)(distance*SAMPLERATE)];
}

void PutValue(int x,int y,BYTE val,BYTE* img,int (*func)(int,int))
{
        int       pval,dval;

        pval = img[y*HDWIDTH+x];
        dval = (*func)(pval,1) + val;
        dval = (*func)(dval,0);
```

```
        if( dval > 255 )      dval = 255;
        img[y*HDWIDTH+x] = (BYTE)dval;
}

int RightInverted(int color,int flag)      /* intencity to digital data */
{
        int       i;

        if( flag == 0 ){
                return Righttable[color];
        }
        else{
                for(i=0;i<256;i++){
                        if( color == Righttable[i] ){
                                return i;
                        }
                }
                return 0;
        }
}

int LeftInverted(int color,int flag)
{
        int       i;

        if( flag == 0 ){
                return Lefttable[color];
        }
        else{
                for(i=0;i<256;i++){
                        if( color == Lefttable[i] ){
                                return i;
                        }
                }
                return 0;
        }
}

long LittleEndianLong(long value)
{
        unsigned char       *temp;
        long                newvalue;

        temp = (unsigned char*)&value;
        newvalue = (temp[0]<<24) | (temp[1]<<16) | (temp[2]<<8) | temp[3];
        return newvalue;
}

long BigEndianLong(long value)
{
        return value;
}
```

-- generator.h ----------

```
#ifndef   ___dfmgenerator___
#define   ___dfmgenerator___

#define   FAIL              0
#define   TRUE              !FAIL
#define   HDWIDTH           1920
#define   HDHEIGHT  1035
#define   SAMPLERATE 200
#define   OFF               0
#define   ON                !OFF
#define LEFT          0
#define RIGHT         1

#define           BUNSAN    1.2       /* default */
#if 0
#         define   RightInverted(C)    Righttable[C]
#         define   LeftInverted(C)             Lefttable[C]
#endif
/*
#define   Righttable right_table
#define   Lefttable  left_table
*/
typedef   unsigned char       BYTE;
typedef struct{
                  double    swidth;           /* parameter.dat */
                  double    sheight;
                  int       swpix;
                  int       shpix;
                  int       offsetx;
```

```
                int     offsety;
                double  distance;
                int     fixation;
                double  fixdepth;
                double  variance;
                int     verbose;
                char    posfilename[64];  /* hwsetup.dat */
                double  eye;
                int     reference;
                int     sdotsize;
                double  cdotsize[2];
                int     disparity;
                int     valsize;
                char    maskfilename[64]; /* arguments */
                char    imagename[64];
                double  depth;
                int     marker;              /* options */
                int     dotnum;
                double  lroffset; /* for H.K */
                int     imagetype;
        }       PARAM;

typedef char*   TERM;

#ifdef  ___PRIVATE___
#undef  ___PRIVATE___

#include "vsd.h"

TERM    Pterm[9] = {
                "POSITION_FILE",
                "DISTANCE_EYE",
                "REFERENCE",
                "DOT_SIZE",
                "DISPARITY",
                "VALIABLE_SIZE",
                "FIXATION",
                "VARIANCE",
                NULL
        };

TERM    Hterm[8] = {
                "SCREEN_WIDTH",
                "SCREEN_HEIGHT",
                "SCREEN_W_PIXEL",
                "SCREEN_H_PIXEL",
                "OFFSET_X",
                "OFFSET_Y",
                "DISTANCE_SCREEN",
                NULL
        };

TERM    Dterm[5] = {
                "HORIZONTAL",           /* 0x01 */
                "VERTICAL",             /* 0x02 */
                "SHEAR",    /* 0x04 */
                "SIMULATED",            /* 0x08 */
                NULL
        };

BYTE    *Lefttable,*Righttable;
void    (*Parallax)(double,double,double,double*,double*,double*,double*,PARAM*);
long    (*EndianLong)(long);
double  Variance;

void    LoadParameterFile(char *filename,PARAM *para);
void    LoadSetupFile(char *filename,PARAM *para);
BYTE    GetIntencity(double,BYTE*);
void    PutValue(int,int,BYTE,BYTE*,int (*)(int,int));
void    DrawPoint(BYTE*,double,double,BYTE*,double,PARAM*,int (*)(int,int));
void    HorizontalParallax(double,double,double,double*,double*,double*,double*,PARAM*);
void    CalcVandHShearDisparity(double,double,double,double*,double*,double*,double*,PARAM*);
void    CalcVerticalShearDisparity(double,double,double,double*,double*,double*,double*,PARAM*);
void    CalcHorizontalShearDisparity(double,double,double,double*,double*,double*,double*,PARAM*);
void    AfinTransfer(double,double,double,double*,double*,PARAM*,int);
void    CalcDisparityField(double,double,double,double*,double*,double*,double*,PARAM*);
void    CalcVergence(double,double,double,double*,double*,double*,double*,PARAM*);
double  CalculateDotSize(PARAM*,double);
BYTE*   LoadDotTable(PARAM*);
double  q_nor(double);
double  p_nor(double);
void    CreateDotTable(BYTE*,int);
long*   LoadPositionData(char*,int*);
void    DrawImage(BYTE*,BYTE*,BYTE*,PARAM*);
void    DrawFixation(PARAM*,double,BYTE*,BYTE*);
void    DrawSpreadPixel(double,double,int,BYTE*,int (*)(int,int));
```

```
void        DrawPixel(double,double,int,BYTE*,int (*)(int,int));
BYTE*       LoadShapeData(PARAM*);
BYTE*       InitializeImage(int left_right);
PARAM*      AnalizeArguments(int,char**,PARAM*);
void        DrawPoint(BYTE*,double,double,BYTE*,double,PARAM*,int (*)(int,int));
BYTE        GetIntencity(double,BYTE*);
void        PutValue(int,int,BYTE,BYTE*,int (*)(int,int));
void        DestroyData(BYTE*,BYTE*,BYTE*);
int         LeftInverted(int,int);
int         RightInverted(int,int);
long        LittleEndianLong(long);
long        BigEndianLong(long);

void        (*DisparityFunc[16])(double,double,double,double*,double*,double*,double*,PARAM*) = {
                    NULL,
                    NULL,
                    CalculateVerticalDisparity,     /* 0010 */
                    NULL,
                    NULL,
                    CalcHorizontalShearDisparity,   /* 0101 */
                    CalcVerticalShearDisparity,     /* 0110 */
                    CalcVandHShearDisparity,        /* 0111 */
                    HorizontalParallax,             /* 1000  changing size */
                    CalcVergence,                                /* 1001 */
                    CalcDisparityField,             /* 1010 */
                    HorizontalParallax,             /* 1011 */
                    NULL,
                    NULL,
                    NULL,
                    NULL
          };


#else     /* not ___PRIVATE___ */

#endif    /* ___PRIVATE___ */

#endif
```

-- brightness.c ----------

```
#define    ___PRIVATE___
#include "brightness.h"

BYTE* LoadIntensityProfile(char *filename)
{
          /* data , a number of data , first value(x,y) , last value(x,y) */
          int       pos,line,num;
          BYTE      *profile;
          double    *datax,*datay;
          FILE      *fp;
          char      buf[256];

          if( (fp = fopen(filename,"r")) == NULL ){
                    fprintf(stderr,"*** file not found:%s ***¥n",filename);
                    return NULL;
          }
          line = 0;
          while( fgets(buf,255,fp) != NULL ){
                    if( line == 0 ){
                              num = (int)GetNumber(buf,&pos);
                              if( (datax = (double*)malloc(sizeof(double)*num)) == NULL ){
                                        fprintf(stderr,"*** memory allocation error in LoadIntencityProfile ***¥n");
                                        fclose(fp);
                                        return NULL;
                              }
                              if( (datay = (double*)malloc(sizeof(double)*num)) == NULL ){
                                        fprintf(stderr,"*** memory allocation error ***¥n");
                                        fclose(fp);
                                        free(datax);
                                        return NULL;
                              }
                    }
                    else{
                              datax[line-1] = GetNumber(buf,&pos);
                              datay[line-1] = GetNumber(buf+pos,&pos);
                    }
                    if( line == num )     break;
                    line++;
          }
          if( (profile = CreateFunction(datay,datax,num,datay[0],datay[num-1])) == NULL
                    fprintf(stderr,"*** memory allocation error ***¥n");
                    return NULL;
          }
```

```
	fclose(fp);
	free(datax);
	free(datay);
	return profile;
}

BYTE* CreateFunction(double *x,double *y,int num,double min,double max)
{
	double	*z,step,val;
	int	i;
	BYTE	*tbl;

	if( (z=(double*)malloc(sizeof(double)*num)) == NULL ){
		return NULL;
	}
	MakeTable(x,y,z,num);
	if( (tbl=(BYTE*)malloc(sizeof(BYTE)*256)) == NULL ){
		free(z);
		return NULL;
	}
	step = (max - min) / 256;
	val = min;
	for(i=0;i<256;i++){
		tbl[i] = (BYTE)(Spline(val,x,y,z,num) + 0.5);
		val += step;
	}
	free(z);
	return tbl;
}

void MakeTable(double *x,double *y,double *z,int num)
{
	int	i;
	double	t;

	z[0] = 0.0;
	z[num-1] = 0.0;
	for(i=0;i<num-1;i++){
		H[i] = x[i+1] - x[i];
		D[i+1] = (y[i+1] - y[i]) / H[i];
	}
	z[1] = D[2] - D[1] - H[0] * z[0];
	D[1] = 2 * (x[2] - x[0]);
	for(i=1;i<num-2;i++){
		t = H[i] / D[i];
		z[i+1] = D[i+2] - D[i+1] - z[i] * t;
		D[i+1] = 2 * (x[i+2] - x[i]) - H[i] * t;
	}
	z[num-2] -= (H[num-2] * z[num-1]);
	for(i=num-2;i>0;i--){
		z[i] = (z[i] - H[i] * z[i+1]) / D[i];
	}
}

double Spline(double t,double *x,double *y,double *z,int num)
{
	int	i,j,k;
	double	d,h;

	i = 0;
	j = num - 1;
	while( i < j ){
		k = ( i + j ) / 2;
		if( x[k] < t ){
			i = k + 1;
		}
		else{
			j = k;
		}
	}
	if( i > 0 )		i--;
	h = x[i+1] - x[i];
	d = t - x[i];
	return (((z[i+1] - z[i]) * d / h + z[i] * 3) * d + ((y[i+1] - y[i]) / h
		- (z[i] * 2 + z[i+1]) * h)) * d + y[i];
}

double GetNumber(char *buf,int *pos)
{
	char	nbuf[256],*ptr;
	int	i,state;
	double	num;

	i = 0;
	ptr = nbuf;
	state = 0;
```

```
        while( buf[i] != '¥n' || buf[i] != '¥0' ){
                if( strchr("0123456789.",buf[i]) != NULL ){
                        if( state == 0 )        state++;
                        *ptr++ = buf[i];
                }
                else{
                        if( state == 1 ){
                                state++;
                        }
                }
                if( state == 2 ){
                        break;
                }
                i++;
        }
        *pos = i;
        *ptr = '¥0';
        num = atof(nbuf);
        return num;
}
```

-- brightness.h ----------

```
#ifndef   ___brightness___
#define   ___brightness___

#ifdef  ___PRIVATE___
#undef  ___PRIVATE___

#include <stdio.h>
#include <strings.h>
#include <stdlib.h>
typedef unsigned char   BYTE;
double    H[256],D[256];

BYTE* LoadIntensityProfile(char *filename);
double Spline(double t,double *x,double *y,double *z,int num);
void MakeTable(double *x,double *y,double *z,int num);
BYTE* CreateFunction(double *y,double *x,int num,double min,double max);
double GetNumber(char *buf,int *pos);

#else   /* not ___PRIVATE___ */

extern BYTE* LoadIntensityProfile(char *filename);

#endif  /* ___PRIVATE___ */

#endif
```

-- vsd.c ----------

```
#include <math.h>
#include <stdio.h>
#define   PRIVATE
#include "vsd.h"

#define   HorizontalShear               0
#define   VerticalShear                 1
#define   OverallShear                  2
#define   HorizontalSize        3
#define   VerticalSize                  4
#define   OverallSize           5
#define   VerticalSizeModulation        6
#define   CenterX                       961
#define   CenterY                       520
#define   LeftEnd                       42.0
#define   TopEnd                        9.0
#define   RightEnd              1880.0
#define   BottomEnd             1031.0
#define   Hshear                0.02
#define   Vshear                0.02
#define   HSize                 0.04
#define   VSize                 0.04
#define   ScreenLength          2.43
#define   ScreenHeight          1.40
#define   Xppm                          ((RightEnd - LeftEnd) / ScreenLength)
#define   Yppm                          ((BottomEnd - TopEnd) / ScreenHeight)
#define   Xmpp                          (1 / Xppm)
#define   Ympp                          (1 / Yppm)

static double VAmp[4]={50, 50, 30, 30};
```

```
static double VFreqX[4]={0.0125, 0.0125, 0.0, 0.0};
static double VFreqY[4]={0.0125, 0.0125, 0.025, 0.025};
static double VPhaseX[4]={90, -90, 90, -90};
static double VPhaseY[4]={0, 0, 0, 0};

void CalculateVerticalDisparity(double dotX,double dotY,double disptype,double *rx,double *ry,double *lx,double *ly,PARAM*
paraptr)
{
        double sinHX, sinHY, sinVX, sinVY;
        double x[2], y[2], Hsize[2], Vsize[2];
        double Vamp, VfreqX, VfreqY, VphaseX, VphaseY;
        int i,j,sign;

        if( disptype >= 6.0 ){
                j = (int)(disptype - 6.0);
                disptype = 6.0;
        }
        Hsize[0] = 1.0;
        Hsize[1] = 1.0 - HSize;
        Vsize[0] = 1.0;
        Vsize[1] = 1.0 - VSize;
        x[0] = x[1] = dotX;  /* init x and y */
        y[0] = y[1] = dotY;
        for( i=0; i<2; i++){
                if( i==0 ) sign = 1;
                else       sign = -1;
                switch( (int)disptype
                case HorizontalShear:
                        x[i] -= (dotY-CenterY)*Hshear*sign;
                        y[i] = dotY;
                        break;
                case VerticalShear:
                        x[i] = dotX;
                        y[i] += (dotX-CenterX)*Vshear*sign;
                        break;
                case OverallShear:
                        x[i] -= (dotY-CenterY)*Hshear*sign;
                        y[i] += (dotX-CenterX)*Vshear*sign;
                        break;
                case HorizontalSize:
                        x[i] = (dotX-CenterX) * Hsize[i] + CenterX;
                        y[i] = dotY;
                        break;
                case VerticalSize:
                        x[i] = dotX;
                        y[i] = (dotY-CenterY) * Vsize[i] + CenterY;
                        break;
                case OverallSize:
                        x[i] = (dotX-CenterX) * Hsize[i] + CenterX;
                        y[i] = (dotY-CenterY) * Vsize[i] + CenterY;
                        break;
                case VerticalSizeModulation:
                        Vamp = tan(VAmp[j]/120/180*M_PI)*Yppm;
                        VfreqX  = VFreqX[j]/(2*tan(M_PI/360))*Xmpp;
                        VfreqY  = VFreqY[j]/(2*tan(M_PI/360))*Ympp;
                        VphaseX = VPhaseX[j]/180*M_PI;
                        VphaseY = VPhaseY[j]/180*M_PI;
                        sinVX = sin( 2*M_PI*VfreqX*(dotX-CenterX)+VphaseX);
                        sinVY = sin( 2*M_PI*VfreqY*(dotY-CenterY)+VphaseY);
                        x[i] = dotX;
                        y[i] -= Vamp*sinVX*sinVY*sign;
                        break;
                defalut:
                        x[i] = dotX;
                        y[i] = dotY;
                }
                if( (x[i] < LeftEnd)||(RightEnd < x[i]) || (y[i] < TopEnd) ||(BottomEnd < y[i]) ){
                        x[i] = y[i] = 0.0;
                }
        }
        /* printf("vsd check %lf %lf %lf %lf¥n",x[0],y[0],x[1],y[1]); */
        *rx = x[1];
        *ry = y[1];
        *lx = x[0];
        *ly = y[0];
}
```

-- vsd.h ----------

```
#ifndef ___VSD___
#define ___VSD___

#ifdef  PRIVATE
#undef  PRIVATE
```

```
#include "generator.h"

#else
extern void CalculateVerticalDisparity(double, double, double, double*, double*, double*, double*, PARAM*);
#endif
#endif
```

-- savedfm.c ----------

```
/*
gcc -O -c savedfm.c -I/export/gassan1/dfm/include -DDFM_E
*/

#define    __PRIVATE__
#include "savedfm.h"
#include <string.h>

union{
        struct{
                char    dfm[4];
                int     xsize;
                int     ysize;
                int     bit_mode;
                int     color;
                int     n_frame;
                int     frame_num;
                int     reserve;
        }       st;
        char    data[32];
}       whd;

void SaveImage(BYTE* left,BYTE* right,PARAM* paraptr)
{
        switch(paraptr->imagetype ){
        case 0:
                SaveDfmImage(left,right,paraptr);
                break;
        case 1:
                SavePpmImage(left,right,paraptr);
                break;
        }
}

static void SaveDfmImage(BYTE* left,BYTE* right,PARAM* paraptr)
{
        FILE                    *fpo;
        int                     j;
        char                    *gptr,*rptr;

        strcpy(whd.st.dfm,"DFM");
        whd.st.xsize = HDWIDTH;
        whd.st.ysize = HDHEIGHT;
        whd.st.bit_mode = 8; /* 10 or 8 bit */
        whd.st.color    = 1; /* 1: Color 0: Gray Scale */
        whd.st.n_frame  = 1;
        whd.st.frame_num    = 0;
        whd.st.reserve = 0;
        if( paraptr->verbose )
                printf("%s :DFM¥n",paraptr->imagename);
        if( (fpo = fopen(paraptr->imagename,"wb")) == NULL ){
                fprintf(stderr,"*** file not create:%s ***¥n",paraptr->imagename);
                return;
        }
        for(j=0;j<32;j++){
                fputc((whd.data)[j],fpo);
        }
        gptr = left;
        rptr = right;
        for(j=0;j<HDWIDTH*HDHEIGHT;j++){
                fputc(*gptr++,fpo);
                fputc(0,fpo);
                fputc(*rptr++,fpo);
        }
        fclose(fpo);
}

static void SavePpmImage(BYTE* left,BYTE* right,PARAM* paraptr)
{
        FILE                    *fpo;
        int                     j;
        char                    *gptr,*rptr;

        if( paraptr->verbose )
```

```
                printf("%s :PPM¥n",paraptr->imagename);
        if( (fpo = fopen(paraptr->imagename,"wb")) == NULL ){
                fprintf(stderr,"*** file not create:%s ***¥n",paraptr->imagename);
                return;
        }
        fprintf(fpo,"P6¥n%d %d¥n255¥n",HDWIDTH,HDHEIGHT);
        gptr = left;
        rptr = right;
        for(j=0;j<HDWIDTH*HDHEIGHT;j++){
                fputc(*rptr++,fpo);
                fputc(*gptr++,fpo);
                fputc(0,fpo);
        }
        fclose(fpo);
}
```

-- savedfm.h ----------

```
#ifndef ___VSD___
#define ___VSD___

#ifdef  PRIVATE
#undef  PRIVATE

#include "generator.h"

#else
extern void CalculateVerticalDisparity(double,double,double,double*,double*,double*,double*,PARAM*);
#endif
#endif
```

-- parameter.dat ----------

```
# parameter file
POSITION_FILE   position.dat
DISTANCE_EYE    60.0
REFERENCE  OFF
DOT_SIZE        15
# SIMULATED SHEAR HORIZONTAL VERTICAL
DISPARITY  SIMULATED|HORIZONTAL
VALIABLE_SIZE   OFF
# FIXATION F(fix) or M(moving)
FIXATION   M
VARIANCE   1.0
```

-- hwsetup.dat ----------

```
#Hardware Setup
SCREEN_WIDTH    2496.0
SCREEN_HEIGHT   1345.5
SCREEN_W_PIXEL  1920
SCREEN_H_PIXEL  1035
OFFSET_X        0
OFFSET_Y        0
DISTANCE_SCREEN 1000.0
```

-- mask.dat ----------

```
#1920
1035
```

-- position.dat ----------

バイナリファイルのため添付なし

-- intencityL.dat ----------

```
32.0
  7.0, -0.04957
 15.0, -0.03956
 23.0,  0.000
 31.0,  0.1187
 39.0,  0.3165
 47.0,  0.6331
```

```
 55.0,  1.068
 63.0,  1.345
 71.0,  1.662
 79.0,  2.057
 87.0,  2.493
 95.0,  3.007
103.0,  3.759
111.0,  4.392
119.0,  5.381
127.0,  6.212
135.0,  7.676
143.0,  9.100
151.0, 10.49
159.0, 12.11
167.0, 14.01
175.0, 16.18
183.0, 18.64
191.0, 20.50
199.0, 22.51
207.0, 25.40
215.0, 27.62
223.0, 29.67
231.0, 31.46
239.0, 33.24
247.0, 34.54
255.0, 35.57
```

-- intencityR.dat ----------

```
32.0
  7.0,  0.1381
 15.0,  0.1482
 23.0,  0.1583
 31.0,  0.1684
 39.0,  0.1978
 47.0,  0.2079
 55.0,  0.2770
 63.0,  0.4748
 71.0,  0.8309
 79.0,  1.227
 87.0,  1.622
 95.0,  2.018
103.0,  2.493
111.0,  2.928
119.0,  3.601
127.0,  4.431
135.0,  5.421
143.0,  6.449
151.0,  7.518
159.0,  9.417
167.0, 10.92
175.0, 13.10
183.0, 15.19
191.0, 17.77
199.0, 20.10
207.0, 22.91
215.0, 25.01
223.0, 27.82
231.0, 30.15
239.0, 32.64
247.0, 34.22
255.0, 35.77
```

-- makefile ----------

```
#
generator: generator.o vsd.o savedfm.o brightness.o
	cc -o generator generator.o vsd.o savedfm.o brightness.o -lm

vsd.o:	vsd.c generator.h vsd.h
	cc -0 -c vsd.c -D__STDC__=0

savedfm.o: savedfm.c generator.h savedfm.h
	cc -0 -c savedfm.c -D__STDC__=0

generator.o:		generator.c generator.h vsd.h savedfm.h brightness.h
	cc -0 -c generator.c -D__STDC__=0

brightness.o:		brightness.c brightness.h
	cc -0 -c brightness.c -D__STDC__=0
```

-- mkimg1 ----------

```
#!/bin/csh
cp mvparameter.dat parameter.dat
makeimage 400 mv ; mv mv*.dfm Images/WithoutRef/MoveVergence/400/
makeimage 800 mv ; mv mv*.dfm Images/WithoutRef/MoveVergence/800/
makeimage 1000 mv ; mv mv*.dfm Images/WithoutRef/MoveVergence/1000/
makeimage 1920 mv ; mv mv*.dfm Images/WithoutRef/MoveVergence/1920/
```

-- makeimage ----------

```
#!/bin/csh
@ i=85
echo generator mask$1.dat $2$1'_0'$i.dfm $i'0.0' -mw
generator mask$1.dat $2$1'_0'$i.dfm $i'0.0' -mw
@ i++
while( $i < 100 )
        echo generator mask$1.dat $2$1'_0'$i.dfm $i'0.0' -mb
        dfmgenerator mask$1.dat $2$1'_0'$i.dfm $i'0.0' -mb
        @ i++
end
#
while( $i < 116 )
        echo generator mask$1.dat $2$1'_'$i.dfm $i'0.0' -mb
        generator mask$1.dat $2$1'_'$i.dfm $i'0.0' -mb
        @ i++
end
```