Turning Pointer
— 移動操作だけで指示方向を制御できるポインタ —

古井 陽之助 (furui@trl.ibm.co.jp)
日本アイ・ビー・エム（株） 東京基礎研究所
〒242-8502 神奈川県大和市下鶴間1623-14

従来のリアルタイムグループウェアシステムで用いられるテレポインタは向きが固定されており、使用者は指示意図を表現するためにポインタの向きを用いることができない。これは、向きを制御する方法を提供することによって解決することができる。しかしながら、テレポインタはしばしばシステムポインタを兼ねて用いられるので、その制御方法はシステムポインタのためにすでに割り当てられている操作を再定義するものであるべきではない。本論文は、向きを制御するための機能を備えた新しいテレポインタ、 **Turning Pointer** について述べる。Turning Pointer の機能は、マウスなどのポインティングデバイスの動きによってのみ制御され、マウスボタンのクリックなどの操作を必要としないので、システムポインタを兼ねたテレポインタとして使用することができる。

Turning Pointer
— Pointing Direction Control Only According to Its Motion —

Younosuke Furui
IBM Research, Tokyo Research Laboratory

Since the telepointer of a conventional real-time groupware system has a fixed pointing direction, users cannot use the direction in order to express themselves. This shortcoming can be remedied by providing a method for changing the direction. However, since a telepointer is often also used as a system pointer, the new method should not redefine any operations already defined for system pointers. This paper introduces a new type of telepointer, *Turning Pointer*, which provides some functions for changing the pointing direction. Turning Pointer can also be used as a system pointer, because all the functions are controlled not by means of a separate operation such as clicking a mouse button, but according to the motion of the pointing device.

1 はじめに

今日、計算機システムのGUIにおいて使用されるマウスポインタもしくはシステムポインタは、非常に馴染み深いインタフェースになっている。システムポインタは、使用者がシステムに対して画面上の座標（**tracking point**）を指定するために用いられる。システムポインタは多くの場合矢印の形をしていて、先端は常に tracking point がどこにあるかを示しており、向きは固定されている。

一方、リアルタイム電子会議システムにおけるテレポインタは、複数の使用者がシステムを介して共有している仮想空間内のオブジェクトを指し示すために用いられるインタフェースである。テレポインタも多くの場合矢印のように向きを持つ形をしている。しかし、システムポインタの指示対象が点であるのと異なり、テレポインタの指示対象は広がりを持つオブジェクトである。そのため、使用者はテレポインタを動かすことによって指示意図を表現することがある。このような動作を**ジェスチャリング**と呼ぶ [3]。

従来のテレポインタは、上述のようなシステムポインタとの違いについての考察に基づいてはデザインされていない。矢印の向き、すなわちポインタの指示方向は固定であり、必ずしも指示対象物の方を向いていない。そこで著者は、指示方向を制御できるようなテレポインタをデザインすることを考えた。

一方、テレポインタが同時にシステムポインタとしても用いられる場合がしばしばある。したがって、テレポインタへの制御操作はシステムポインタとしての入力操作と衝突しないこと、つまりマウスクリックなどにテレポインタの制御機能を割り当てないことが望ましい。

著者の考案した **Turning pointer** （図1）は、指示方向を動きのみで制御するポインタである。本論文では、 Turning Pointer の基本設計や特長を

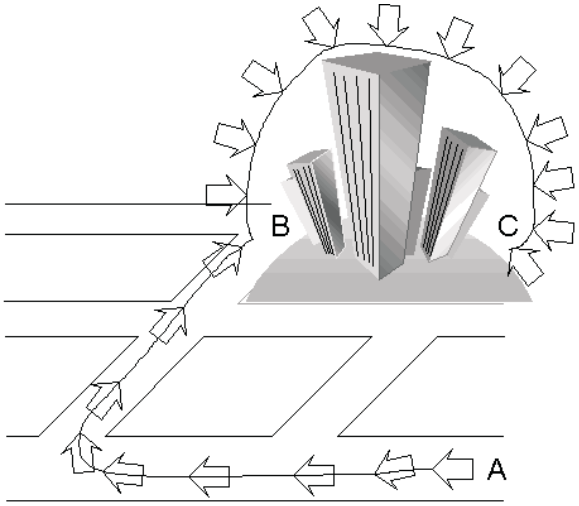


図 1: Turning Pointer

Turning Pointer の場合、移動方向と指示方向が、A からB までは一致、B からC までは直交している。

説明するとともに、著者の実装した評価用プロトタイプについて報告する。また、テレポインタ以外にもアプリケーションがあると考え、その一例として実装したシューティングゲームについても述べる。

本論文の構成は以下の通りである。2 節では、ジェスチャリングにおいて従来のテレポインタの持つ問題について議論する。3 節では、Turning Pointer の基本設計、利点、実装方法を説明する。4 節では、テレポインタ以外のアプリケーション実装例について述べる。5 節では、本論文のまとめと今後の課題を述べる。

2 ジェスチャリングにおける問題

従来のリアルタイム電子会議システムでは、1 節で述べたようにテレポインタの指示方向を制御することができない。本節では、ジェスチャリングにおいて従来のテレポインタの持つ問題について議論する。

(A) 指示方向の制御

まず、ポインタの指示方向が制御できればジェスチャリングが不要になる場合について説明する。図2のように方向を示す場合を考える。指示方向が固定されている従来のテレポインタを使用すると、もし示す方向がポインタの指示方向と異なるならば、示す方向に沿ってポインタを動かさなくてはならない（図2(a)）。しかし、もしポインタの指示方向で方向を示すことができれば、このジェスチャリングは必要無い（図2(b)）。

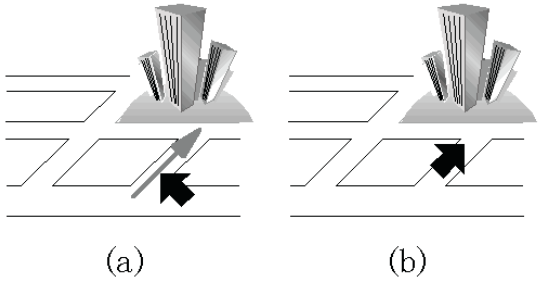


図 2: 方向を示す場合

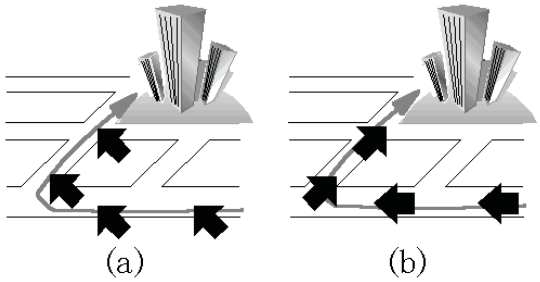


図 3: 経路を示す場合

次に、ポインタの指示方向が制御できればジェスチャリングがより適切に行える場合について説明する。図3のように経路を示す場合を考える。従来のテレポインタを使用する場合、経路はポインタの移動軌跡によってのみ示され、ポインタの指示方向は経路とは全く関係がない（図3(a)）。しかし、もしポインタの指示方向が制御できるなら、経路はポインタの移動軌跡と指示方向の両方によって強調される（図3(b)）。

図4は、ジェスチャリングにポインタの指示方向制御を組み合わせることのできるもう一つの例である。この図のようにオブジェクトを示す場合、使

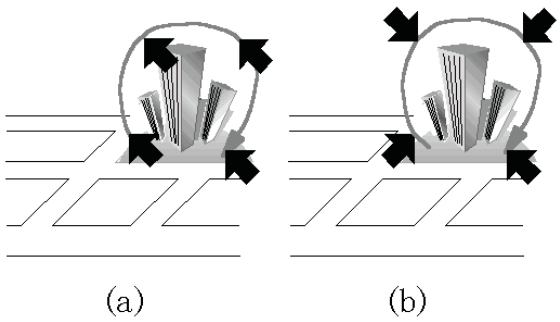


図 4: オブジェクトを示す場合

用者はオブジェクトの周りを囲むようにテレポインタを動かす（図4(a)）。しかし、従来のテレポインタを使用する場合、ポインタがオブジェクトを正しく指すのはある角度からのみである（図4(a)ではオブジェクトの右下）。もしポインタの指示方向が制御できるなら、ポインタは常に建物を指すようにできる（図4(b)）。

図2、図3、図4の例から、ポインタの指示方向を制御できることが有用であることが判る。また、図3の例ではポインタの指示方向と移動方向は一致するが、図4の例では一致しないことから、指示方向の制御方法は状況に応じて異なっていなければならないことが判る。したがって、ポインタの指示方向を制御する方法に加え、その制御方法を切り替える方法も必要である。

必ずポインタが指示対象物の方を指すようになっているシステムの例としては、 Flying Fingers [8] が挙げられる。Flying Fingers は、リアルタイム協業のためのポインティング方式である。この方式では、前提として指示対象物は三次元空間の中心に一つだけ存在する。人差し指を突き出した手の形をしたポインタの指示方向は、常に中心を向いている。またポインタは、指示対象物を中心とする球面座標上か、あるいは中心に向かって前後に動くことができる。Flying Fingers は、明らかに複数のオブジェクトを扱うには不向きであるが、ポインタの指示方向を、図4(b)の例を三次元に拡張したような形で活用しているという点で興味深い。

(B) システムポインタとの兼用

テレポインタはシステムポインタを兼ねることが多い。このような場合、1節で述べたように、テレポインタへの制御操作がシステムポインタとしての入力操作と衝突しないことが望ましい。例えば、マウスクリック操作にテレポインタを制御するための機能を割り当てると、テレポインタを使用している間はシステムポインタとしてマウスクリック操作を行うことができない。このような場合、テレポインタがシステムポインタを兼ねることができず、テレポインタとシステムポインタの切り替え操作を導入する必要がある。

テレポインタはマーカと共に多くのリアルタイム電子会議システムに採用されているが、マーカと比べたときのテレポインタの長所の一つはその操作の簡単さにある。上記のような切り替え操作の導入は、テレポインタの持つ長所である操作の簡単さを失わせることになる。

従来技術においてはポインタへの制御は次のような方法で行われていた。

(1) ポインタの位置
(2) モードキーなどによる一時切り替え
(3) メニュー選択などによるコマンド入力

(1) はポインタが画面上のどの位置にあるかに応じて動作モードを変える方法である。グラフィカルなWebブラウザにおいてリンクの上にポインタを移動させるとポインタの形状が手の形に変わるといったものがこの例である。(1) では、あらかじめ画面上に配置されている各表示要素や画面上の座標に関連付けてモードをプログラムしておく。これには、モード切り替えが自動的に起こるというメリットがあるが、使用者の任意によってモード切り替えを行う場合には (2) や (3) の方式を用いることになる。

(2) は、ドラッグ操作のようにマウスボタンの一つを押したままマウスを動かす場合や、 Microsoft Windows の File Manager または Explorer のファイル選択操作におけるコントロールキーやシフトキーを押したままのマウスクリックなどが例として挙げられる。

(3) は、例えば描画アプリケーションにおけるペン先の色や太さをメニューなどから選択するような場合に用いられる。(2) との違いは、モード切り替えの操作や元のモードへの復帰操作に要する操作の手数が多く、使用者の負担が大きい点である。これを軽減するために適用できる方法としては、以下のようなものが挙げられる。しかしいずれも、結局はシステムポインタの操作と衝突するか、デバイスを追加する必要がある。

Marking Menu [4] では、円を放射状に区切った pie menu [2] を非表示で起動し、その円の中心から外側へとポインタを動かす方向によってメニュー項目を選択する。pie menu の起動はクリックで行われる。

Tivoli [6] では、使用者はあらかじめ定義されたジェスチャをポインタで行ってコマンドを入力する。コマンド入力のためのジェスチャと通常の操作を区別するためにモード切り替えを必要とする。

Multifunctional Cursor [7] では、複数のモードを使用者の任意でポインタに割り当てておき (2) によって切り替える。この手法の身近な例としては、描画ソフトのペンモードにおいて左ボタンと右ボタンに異なる色を割り当てる操作が挙げられる。

see-through tools (click-through tools) [1] や T3 [5] では、あらかじめ機能を割り当てた透明な toolglass ごしに操作対象物をクリックすることによって、モード切り替え操作無しでクリックの役割を使い分ける。これを実際に効率よく操作する

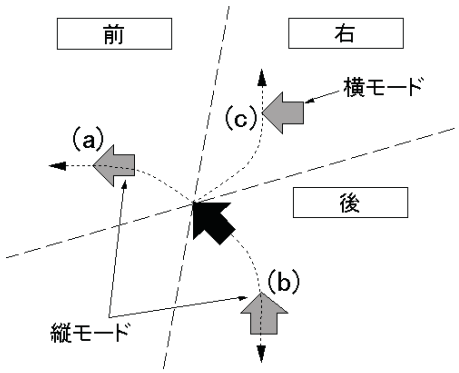


図 5: ジェスチャリングモードの切り替え

には、ポインタとは別に toolglass を操作するためのデバイスを用意して、両手で二つのデバイスを操作しなければならない。また、 T3 ではマウスの傾きを検出してオブジェクトの回転操作などを行うが、この機能を利用するためには、今日一般に普及しているのとは異なる、傾きを検出できるマウスを用いなければならない。

(C) 帯域幅が狭いときの表現力

リアルタイム電子会議システムは、ときに通信の帯域幅が小さいネットワークにおいて使用される。このような環境では、データ通信量を抑えるためにしばしばポインタの動きが間引きされ、結果として使用者の入力した動きが正確に再現されない。指示方向が固定された従来のポインタを使うと、図3や図4のような例では、動きの間引きが大きくなるほど使用者の指示意図を把握するのが困難になる。しかし、ポインタの動きのみならず指示方向も使用者が制御できれば、動きが間引きされても指示意図の把握は比較的容易になる。

3 Turning Pointer

3.1 基本設計

Turning Pointer の基本設計における要点は以下の 2 点である。

I. 状況に応じたやり方（モード）でポインタの指示方向を変える。 Turning Pointer は縦モード、横モード、停止モードの 3 つのジェスチャリングモードを持つ。縦モードでは指示方向は移動方向と一致する（図5(a),(b)）。横モードでは指示方向は移動方向と直交する（図5(c)）。停止モードではその直前の指示方向を保つ。

II. ポインタのモード切り替えを、ポインタそれ自体の動きによって行う。 Turning Pointer は現在の指示方向と移動方向を比較して、次のジェスチャリングモードを決定する（図5）。指示方向に対して前後いずれかに動くと縦モードになり、左右いずれかに動くと横モードになる。ある一定時間以上、移動距離がある一定範囲内に収まっている場合、停止モードになる。

3.2 ジェスチャリングにおける問題の解決

(A) 指示方向の制御

Turning Pointer によって使用者はポインタの指示方向を制御できる。3 つのジェスチャリングモードのうち、縦モードは図3(b) の動作に、横モードは図4(b) の動作に対応する。

(B) システムポインタとの兼用

Turning Pointer は動きによって制御され、制御操作がシステムポインタとしての操作と衝突しない。そのため、モード切り替えのための特別な操作方法や追加のポインティングデバイスを導入する必要が無い。

(C) 帯域幅が狭いときの表現力

指示方向を制御できるので、帯域幅が狭くデータ転送量が制限されてテレポインタの動きが間引きされるような場合でも、指示意図をより伝達しやすい。

3.3 実現方法

Turning Pointer の実現方法を以下に説明する。計算機システムは、使用者によるポインタの移動操作を離散的な座標データの列としてとらえる。以下の手順はこの座標データ列から、ポインタの (1) 移動を検出し、(2) 移動方向を判定し、(3) 前後左右を判定し、(4) 指示方向を決定するものである。

(1) 静止・移動判定

時間による判定 後述する (2) の移動方向判定や (3) の前後左右判定には、使用者がポインタを直前まで停止していたかどうかが影響する。そのため、使用者による入力の時間的連続性を判定する必要がある。新たに座標が入力されたとき、その時刻と、その前に座標が入力された時刻とを比較する。両者の差がある一定時間 t_s を越えていれば、いったん静止したポインタが移動したものと判定する。t_s はパラメータとして調整できる。

距離による判定 座標と座標の間の距離があまり近いと、後述の方法によって算出されるポインタの移動方向が荒い精度でしか得られず、使用者の意図を正しく反映しなくなる。そのため、ポインタが十分な距離を移動するまで、入力された座標をスキップする。具体的には、最後にポインタが移動した先の座標を**ピボット**とする。新たに座標が入力されたとき、これと

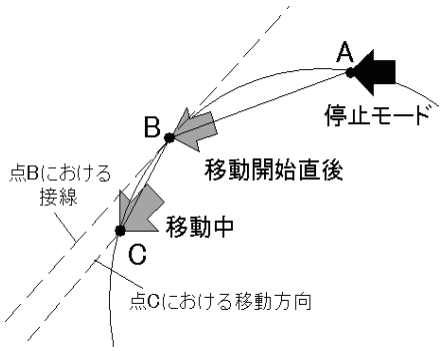


図 6: AからBを通ってCへ

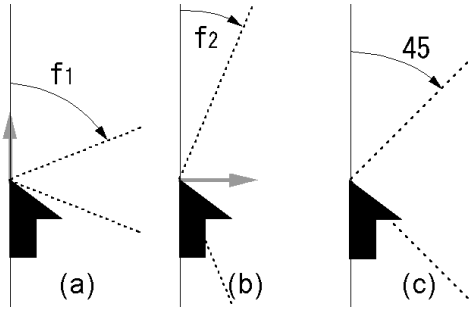


図 7: 前後左右判定における場合分け
(a) 縦モード (b) 横モード (c) 停止モード

ピボットとの距離を計算し、ある一定値（移動中は d_m、停止中は d_s）を 越えていれば、ポインタが新たなピボットに移動したものと判定する。d_m および d_s はパラメータとして調整できる。

(2) 移動方向の判定

入力された離散的な座標データ列は、ポインティングデバイスの特性や座標データの離散化過程のために、実際のユーザ操作の連続的な動きを正しく反映しないことがある。この問題を解決するために、離散的な座標データをもとに、ユーザ操作と同じと見なすことのできる滑らかな曲線を計算して、これからポインタの移動方向を判定する。

後述するプロトタイプでは、図6の例のようにして移動方向を判定する。この例では、点Aにおいて停止モードにあったポインタが、移動を開始して点Bを通って点Cに至っており、システムが捕らえたのは点A, B, Cの座標データである。このとき、点Bにおける移動方向は直線ABの方向とする。また点Cにおける移動方向は、点A, B, Cを結ぶ円弧の点Bを通る接線の方向とする。

(3) 移動方向の前後左右判定

ポインタのその時点における指示方向に対して、(2)で求めた移動方向が前後左右のいずれであるかを判定する。このとき、縦モードでは左右へのズレを、横モードでは前後へのズレを許容する（図7）。これは、使用者の意図しない操作のズレのために後述の(4)においてポインタの向きが変わってしまうと、操作性が低下するためである。$f1$ と $f2$ はパラメータとして調整できる。

縦モード 指示方向から左右 $f1$ 度を前、その対頂角を後ろ、残りを左右とする（図7(a)）。

横モード 指示方向から左右 $f2$ 度を前、その対頂角を後ろ、残りを左右とする（図7(b)）。

停止モード 指示方向から左右45度を前、その対頂角を後ろ、残りを左右とする（図7(c)）。

(4) モードと指示方向の決定

ポインタの指示方向は、図5に示されるようにして決定される。(3)において移動方向が前と判定されたら、ジェスチャリングモードを縦モードへ移行して、ポインタの指示方向を移動方向と同じとする。後ろと判定されたら、ジェスチャリングモードは縦モードへ移行するが、ポインタの指示方向は移動方向と反対とする。左右と判定されたら、横モードへ移行し、ポインタの指示方向は移動方向に対して垂直になるようにする。

3.4 プロトタイプ

上述の手順を実現するプロトタイプをJavaアプレット Pointer.class として実装した。このプロトタイプでは、背景画像の上で矢印の形をしたポインタを動かせる（図1）。また、キー入力によってポインタの軌跡を記録できる。ただし、ポインタの使用感を評価するためのものであるため、リアルタイム電子会議機能は提供せず、ローカルな計算機のみで動作する。

このプロトタイプ実装の初期段階においては、各パラメータは以下の通りであった。

$$t_s = 300 \quad [\text{ミリ秒}]$$
$$d_m = d_s = 7 \quad [\text{ドット}]$$
$$f1 = f2 = 45 \quad [\text{度}]$$

この段階のプロトタイプを著者自身で試したところ、いくつか問題が見られたが、各パラメータを調整することで解消できることも判った。以下ではこれらの問題と調整方法を述べる。

使用者がポインタをまっすぐ動かしているつもりでも、座標データのわずかなズレのために移動方向を変えたとシステムが判定し、結果として不

パラメータセット xxxxyz	$f1, f2$ xxxx	d_m, d_s y	t_s z
6030LL	$f1 = 60, f2 = 30$	$d_m = 7, d_s = 10$	$t_s = 300$
6030LS	$f1 = 60, f2 = 30$	$d_m = 7, d_s = 10$	$t_s = 100$
6030SL	$f1 = 60, f2 = 30$	$d_m = 3, d_s = 5$	$t_s = 300$
6030SS	$f1 = 60, f2 = 30$	$d_m = 3, d_s = 5$	$t_s = 100$
9000LL	$f1 = 90, f2 = 0$	$d_m = 7, d_s = 10$	$t_s = 300$
9000LS	$f1 = 90, f2 = 0$	$d_m = 7, d_s = 10$	$t_s = 100$
9000SL	$f1 = 90, f2 = 0$	$d_m = 3, d_s = 5$	$t_s = 300$
9000SS	$f1 = 90, f2 = 0$	$d_m = 3, d_s = 5$	$t_s = 100$

表 1: パラメータセットの一覧

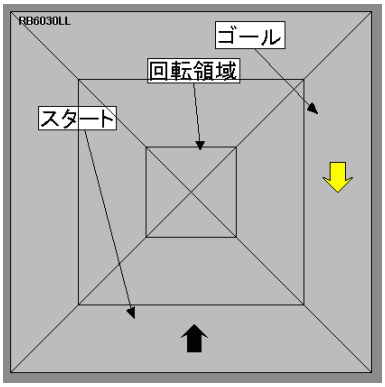
意にモードが切り替わることがあった。これは、縦モード時の前後判定しきい値 $f1$ を 45 度より大きく、横モード時の $f2$ を 45 度より小さくして、前後左右判定でズレを吸収することによって解消できることが判った。特に $f1 = 90, f2 = 0$ の場合、ポインタをいったん停止するまでモードは切り替わることがない（ただし、ポインティングデバイスとしてペンを使用する場合、ポインタを完全に停止させるのがマウスや TrackPoint-II に比べて難しいため、このような設定では使いづらいことも判った）。

上記の調整でも、ポインタが動き出す瞬間のズレは吸収できないので、移動方向の判定結果が使用者の意図と異なることがあった。これは、停止中の移動判定しきい値 d_s を移動中の d_m より大きくして、移動開始時の移動方向判定がより正確に行われるようにすれば解消できることが判った。

また、使用者はポインタを見ただけではその時点のジェスチャリングモードを知ることができない、つまり入力に対するフィードバックを得られないことも、誤操作の原因となった。これを解消するため、縦モードのときはポインタを青で、横モードのときは赤で、停止モードのときは黒でというように異なる色で描画して、使用者が視覚的にフィードバックを得られるようにした。

3.5 評価実験

上述のプロトタイプを用いて行った簡単な実験について述べる。この実験の目的は、パラメータの調整によって操作性が向上するか、様々なジェスチャリングを失敗なく実行できるようになるかどうかを確認することである。ただし、パラメータ特性は計算機システムの性能やデバイスの性質などによって異なることが想像され、多種多様な条件下でのパラメータ特性を明らかにするためには膨大な量の実験が必要になる。今回は一定の条件下におけるおおまかな調整にとどめた。実験に使った計算機環境は以下の表の通りである。



⇓ 操作を実行して

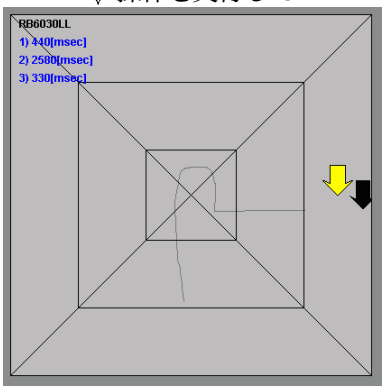


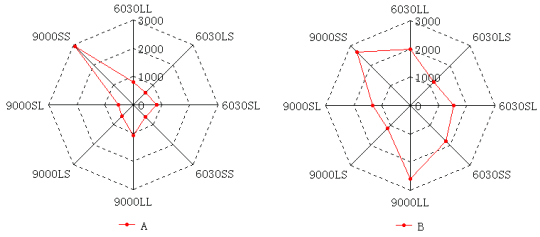
図 8: Prex.class の実行画面例

CPU	Pentium 100MHz
OS	Microsoft Windows 95
Web ブラウザ	Netscape Navigator 4.06[ja]

実験の概要は以下の通りである。まず、著者が自身の試行に基づいて代表的と考える 8 つのパラメータセットを作成した。これを表 1に挙げる。次に、様々なジェスチャリングの基本パターンとなると考える 16 の操作パターンを作成した。また、これらのパターンの実行環境として、 Pointer.class のサブクラスとして Java アプレット Prex.class （図 8）を実装した。いずれのパターンも Prex.class において以下のような手順で実行される。

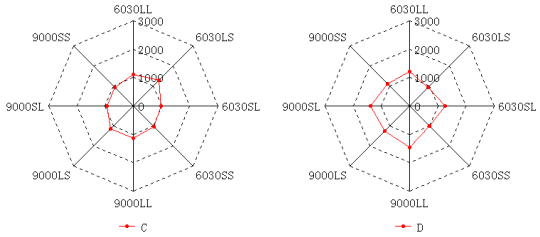
1) ポインタを下部のスタートから上向きにしたままで中央の回転領域に進入させ、
2) 回転領域内で、ゴールにある黄色い矢印の向きに合うよう方向転換し、
3) ゴールへと抜ける。

方向転換中に回転領域をはみだした場合は失敗とみなす。ゴールが上下左右のいずれの台形になるか、また黄色い矢印が上下左右のいずれを向くかで、操作パターンの数は 16 となる。8 つのパラメータセットのそれぞれについてこれを繰り返すので、被験者は上記操作を 128 回行うことになる。



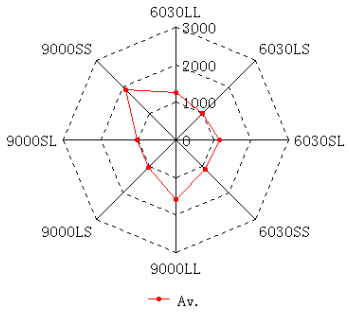
(a) 被験者 A の平均値

(b) 被験者 B の平均値



(c) 被験者 C の平均値

(d) 被験者 D の平均値



(e) 全員の平均値

図 9: パラメータセットごとの平均値

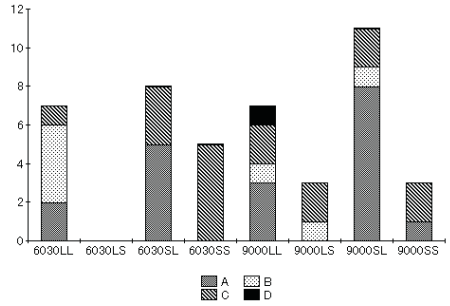


図 10: パラメータセットごとの失敗回数

ゴール・黄矢印	被験者 A	被験者 B	被験者 C	被験者 D
左・下	770	1590	3630	1210
左・左	280	660	1210	330
左・右	660	1480	1150	1260
左・上	440	820	1050	1040
上・下	720	1700	1320	1320
上・左	660	1590	1590	1370
上・右	660	2310	1100	880
上・上	50	600	550	170
右・下	1310	1370	1420	1210
右・左	610	1210	1210	1370
右・右	320	660	880	220
右・上	490	870	600	770
下・下	710	930	1860	770
下・左	610	1320	1100	1160
下・右	660	1260	1040	1100
下・上	490	660	610	940
平均	590	1189	1270	945

表 2: 6030LS における回転所要時間 [msec]

これを 4 人の被験者に実行してもらい、上記 2) の所要時間を操作結果のデータとした。被験者はいずれも東京基礎研究所の所員であり、日常業務においてマウスを使用している。Turning Pointer については予め知識はあるものの特に使用訓練などはしていなかった。実験前には、一通り Turning Pointer の特徴について説明を与えた。また、操作を繰り返すうちに慣れて実験結果が偏ることが予想されたので、これを緩和するために予め 16 パターンを簡単に練習してもらった。また被験者ごとに実行するパターンの順序を変えた。

この実験結果を図 9 と図 10 に挙げる。図 9(a), (b), (c), (d) は各被験者・各パラメータセットごとに 16 パターンの操作結果を平均しグラフにしたものである。図 9(e) は全被験者についての平均値をグラフにしたものである。これらの平均値は成功したもののみを対象に算出した。また、図 10 は失敗回数をグラフにしたものである。

図 9(e) および図 10 によると、8 つのパラメータセットの中で 6030LS が、所要時間の平均値が最も小さく（999msec）、また失敗回数が最も少ない（0 回）。このことから、8 つの中では 6030LS を最適なパラメータセットと考える。表 2 にこの 6030LS における各被験者の操作結果を挙げる。パターンごとにばらつきがあるが、これはパターン自体の難易度によるものと考えられる。例えばスタートからゴールまで一直線に動かすだけの「上・上」操作に比べ、図 8 の「右・下」操作は明らかに難しい。

結論として、パラメータ調整が操作性の向上に効果があること、またマウスを日頃使っている者なら一通りの説明とある程度の練習で Turning Pointer の方向転換操作ができるようになる（ようなパラメータセットが存在する）ことが判った。

4 テレポインタ以外のアプリケーション

4.1 シューティングゲーム

Pointer.class のサブクラスとしてシューティングゲーム Funnel.class を実装した（図 11）。ポインタの挙動に関する処理手順は Turning Pointer のものをそのまま用いている。相違は、ポインタの形状、クリック操作にビーム発射機能を割り当てたこと、

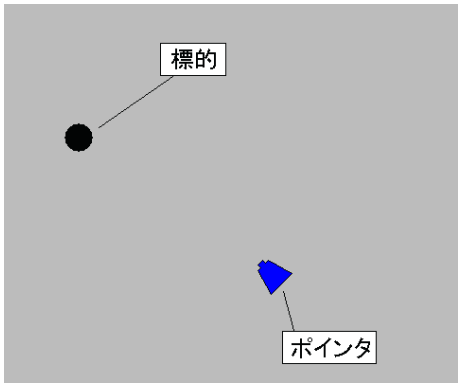


図 11: Funnel.class の実行画面例

そして標的が現われることである。ポインタはふいご状のオブジェクトとして、標的は黒い丸として描画される。標的は四方から現われる。使用者はポインタを操作して標的に照準を合わせ、クリックによってビームを放って標的に命中させる。この一連の操作において、Turning Pointer の仕組みによってポインタの動きに高い自由度が得られる。

実際に使ってみたところ、ビームを標的にうまく命中させるために、指示方向の制御にはテレポインタにおいてよりも高い精度が必要であることが判った。このようなアプリケーションによる相違は、操作性を向上させるために各パラメータを調整する上での相違となるものと予想される。

4.2 仮想美術館内の移動と視線制御（案のみ）

他にも例えば、仮想三次元空間における視線制御に応用することができるだろう。この場合、使用者の操作対象はポインタではなく、仮想三次元空間における自分の位置である。ポインタの指示方向は、自分の視線の方向に相当する。

具体的な例として、仮想三次元空間に構築された仮想美術館の中を歩く場合について考える。壁際などに展示してある美術品を眺めながら移動するときは、視線と移動方向は直角をなすことになる。また、展示品に近寄ったり遠ざかったりといった前後の移動を行うときは視線と移動方向は一致する。展示品を眺めながらではなく、求める展示品を探したりあてもなくさまようような移動もありうる。視線の向きをポインタの指示方向に置き換えれば、移動方向と異なる方向の視線も同じ方向の視線も Turning Pointer の方式で制御できる。

5 おわりに

本論文は、まずリアルタイム電子会議システムにおけるジェスチャリングを行う際に従来のポインタが持っていた問題 (A) 指示方向の制御ができないこと、(B) システムポインタを兼ねることがあるためテレポインタとしての操作と衝突するおそれがあること、(C) 帯域幅が狭く動きが間引きされるときに指示意図の表現力が低下すること、について述べ、これらを解決する方法として筆者が考案した Turning Pointer を説明した。Turning Pointer は、I. 指示方向を変えるためのジェスチャリングモードを複数持つ、II. ジェスチャリングモードの切り替えをポインタそれ自体の動きによって行う、という特徴を持つ。また、プロトタイプを実装して行った評価実験について述べた。今後も評価を進めていく予定である。

謝辞

中島周氏、坂入隆氏、曽谷俊男氏をはじめ日本アイビーエム（株）東京基礎研究所の皆様のご助言とご協力に感謝します。

参考文献

[1] Bier, E. A., Stone, M. C., Fishkin, K., Buxton, W., and Baudel, T., "A Taxonomy of See-through Tools," Proc. CHI '94, pp. 358-364, April, 1994.

[2] Callahan, J., Hopkins, D., Weiser, M., and Schneiderman, B., "An Empirical Comparison of Pie vs. Linear Menus," Proc. CHI '88, pp. 95-100, 1988.

[3] Greenberg, S., Gutwin, C., and Roseman, M., "Semantic Telepointers for Groupware," Proc. OzCHI '96, November, 1996.

[4] Kurtenbach, G., and Buxton, W., "Issues in Combining Marking and Direct Manipulation Techniques," Proc. UIST '91, pp. 137-144, November, 1991.

[5] Kurtenbach, G., Fitzmourice, G., Baudel, T., and Buxton, W., "The Design of a GUI Paradigm based on Tablets, Two-hands, and Transparency," Proc. CHI 97, March 1997.

[6] Moran, T. P., Chiu, P., and van Melle, W., "Pen-Based Interaction Techniques for Organizing Material on an Electronic Whiteboard," Proc. UIST '97, pp. 45-54, 1997.

[7] Muller, M. J., "Multifunctional Cursor for Direct Manipulation User Interface," Proc. CHI '88, pp. 89-94, April, 1988.

[8] Sakai, A., "Flying Fingers: A tool for three-dimensional shared workspace," Electronic Proc. CHI 96, April, 1996.
Available at `http://www.acm.org/sigchi/chi96/proceedings/shortpap.htm`