インタラクション 2003 論文集，pp.123-130 (2003.2)

ぱらぱらウィンドウ：ウィンドウの切り替えを容易にするインタフェース

加藤 直樹†，小國 健‡

東京農工大学工学部†，(株)NTT データ‡

概要：本稿では，オーバラップマルチウィンドウシステムにおけるウィンドウの切り替えが面倒であることを解決するために，一次元の変移を入力すると，最上位のウィンドウから，入力した変移に比例した枚数分のウィンドウが非表示または半透明化されるユーザインタフェースを提案する．評価実験からは，このインタフェースを用いると，下に位置するウィンドウを最上位にするタスクや，二つのウィンドウ間でアイコンを移動するタスクの操作時間が短くなることが示された．

PARA-PARA Window: An Easier Window Switching Interface

Naoki Kato†, Tsuyoshi Oguni‡

Tokyo University of Agriculture and Technology†, NTT Data Corporation‡

Abstruct：This paper proposes a new user interface to solve irritating trouble when switching windows on overlapped multi-window system. By using a 1-dimensional input device, a user can directs thickness of unnecessary windows from the top, and makes them hidden or translucent. As a result of the evaluation experiments, it is shown that the operation times to locate back windows frontward and to move icons from one window to another could be shortened when using the new interface.

１．はじめに

マルチウィンドウシステムの出現によって，一つの物理画面上に複数の仮想画面を表示することが可能になり，たとえば，複数のアプリケーションソフトウェアが表示する情報を同時に閲覧することができるようになった．複数のウィンドウを表示する方式としては，すべてのウィンドウに重なりがないように表示するタイルウィンドウ方式と，重なりを許すオーバーラップウィンドウ方式がある．タイルウィンドウ方式は，多くのウィンドウを表示すると，一つ一つのウィンドウの大きさが小さくなってしまう問題点があり，現在主流なのはオーバーラップウィンドウ方式である．

ところが，このオーバーラップウィンドウ方式にも，ウィンドウを重ねて表示することを許していることから，多くのウィンドウを表示すると，その一部または全体が見えないウィンドウが出てくるという問題点がある．奥行き方向において下側に位置するウィンドウ全体を見えるようにするためには，ウィンドウの一部をポインティングデバイスでクリックするのが一般的であるが，ウィンドウ全体が見えていない場合には，それを隠しているウィンドウを移動するなどの操作をして，一部を見えるようにする必要がある．

文章書きなど一つのウィンドウだけを長時間利用している場合は問題ないが，複数のアプリケーションソフトウェアを駆使した作業やファイルの整理を行うときには，ウィンドウの切り替えを頻繁に行わなければならない．しかし，現状では，直観的に，容易にウィンドウの切り替えを行うための操作方法が確立しているとは言えず，ユーザ

は日々の作業でストレスを溜めている．

そこで，本稿では，既存のウィンドウの切り替え手法が直観的でない理由を明確にし，そして，この問題を解決する入力デバイスおよび操作方法を提案する[1]．具体的には，1 次元の変移を入力することができる入力デバイスから変移を入力すると，最上位のウィンドウから変移に比例した枚数のウィンドウが非表示または半透明化され，下に位置するウィンドウの閲覧と操作ができるようになるユーザインタフェースを提案する．また，予備的評価実験を通して，その有効性を検証する．

２．ウィンドウ切り替えインタフェースの現状

2.1 基本インタフェースの問題点

はじめににも記したように，既存のウィンドウシステムで，ウィンドウを最上位にして，ウィンドウ全体が見えるようにするには，ウィンドウの一部をマウスなどのポインティングデバイスでクリックする．しかし，この操作方法だけでは，次のようなことが起こる．

- ウィンドウの一部や全体が隠れているウィンドウを一時的に参照したい場合，上に位置するウィンドウを一時的に動かしたり，最小（アイコン）化したりしなければならない．さらに元の作業に戻るときには，移動や最小化したウィンドウを元に戻す操作が必要となる．
- 二つのウィンドウを同時に閲覧したり，二つのウィンドウ間でドラッグ＆ドロップなどを行ったりしたい場合，両方のウィンドウが見えるように，両方のウィンドウの大きさや位置を調整しなければならない．

多くのウィンドウシステムでは，ウィンドウの一覧を表示し，そこから選択したウィンドウを最上位に表示する機能が用意されている．たとえば，米国マイクロソフト社の Microsoft Windows では，最小化したウィンドウがボタンとして格納されているタスクバーが常に表示されていて，そのボタンを押すことで対応するウィンドウを最上位に表示することができる．また，Alt キーを押しながら Tab キーを押すことで，すべてのウィンドウがアイコンとして表示されたダイアログが開き，そのまま Tab キーを繰り返し押すことで最上位に表示するウィンドウを選ぶことができる．しかし，この機能では，どのボタンやアイコンが目的とするウィンドウに対応しているのかわからず，手当たり次第にウィンドウを表示しなければならないことがある．

2.2 既存の解決ツール

こうしたウィンドウ切り替えに伴う問題を解決しようと，数々のツールがフリーソフトなどの形態で流通している．次にその例を示す．

- 下に位置し隠れているウィンドウを見えるようにするために，ウィンドウ上部のタイトルバーを右クリックすると，そのウィンドウを最下位に送り込むツール（たとえば[2]）
- 仮想的にデスクトップを増やしてウィンドウの切り替えの負担を減らそうとするツール（たとえば[3]）
- デスクトップに簡単にアクセスできるようにするため，画面左端に置かれたバーを動かすと，動かした部分にはデスクトップが表示されるツール[4]
- 使用していないウィンドウが占める表示領域を小さくするために，ウィンドウを立体的に表示するツール[5]

こうしたツールが便利な場合もあるが，必ずしも使い勝手が良いとは言えない．操作体系・画面表示が複雑化し，屋上屋を重ねた感は否めない．

３．新インタフェースの提案

3.1 次元のミスマッチ

オーバーラップウィンドウの世界は，平板なウィンドウが複数枚積み重なった三次元空間であるとみなせる．一方，現在もっとも普及しているポインティングデバイスであるマウスは，二次元上の点を指し示すために考案されたデバイスである．

これまで，ウィンドウ切り替えの問題を解決しようと，さまざまなアイディアが考案・実装されてきたが，満足のいく結果が出ていないのは，この点が明確に意識されないまま，試行錯誤が行われてきたからではないかと考えられる．つまり，次元のミスマッチが発生していたのである．たとえば，画面上に表示したボタンを二次元の位置を示すためのマウスでクリックすることによって，ウィンドウを最上位に表示するなどの三次元目の操作を行わせていた．

3.2 ぱらぱらウィンドウ

そこで我々は，三次元目を操作するための入力デバイスとして，一次元の変移を入力することが

できる入力デバイスを用い，その入力デバイスから変移を入力すると，最上位のウィンドウから順番に非表示，または半透明になり，下に位置するウィンドウが見えるように，そして，操作できるようになるユーザインタフェース（ぱらぱらウィンドウ）を提案する．

具体的な例を示すと，図１のように，スライドボリューム型のデバイスのつまみを押し込むと，押し込んだ量に応じて上位のウィンドウから非表示になるユーザインタフェースである．

これで，次元のミスマッチが解消でき，また，平面はマウス，奥行きは新デバイスと，役割分担を明確にすることで混乱が避けられる．三次元入力デバイスで同様の操作を行えるようにする方法も考えられるが，この場合，三次元同時に入力しなければならなく，また，三次元空間の動きは不安定となりがちで，操作が難しい．これに対して提案する方法では，従来の二次元の入力とは独立して三次元目を入力でき，かつ，単純な動きだけで入力できるため，操作が簡単である．

3.3　基本設計

次に，提案するユーザインタフェースの基本設計を示す．

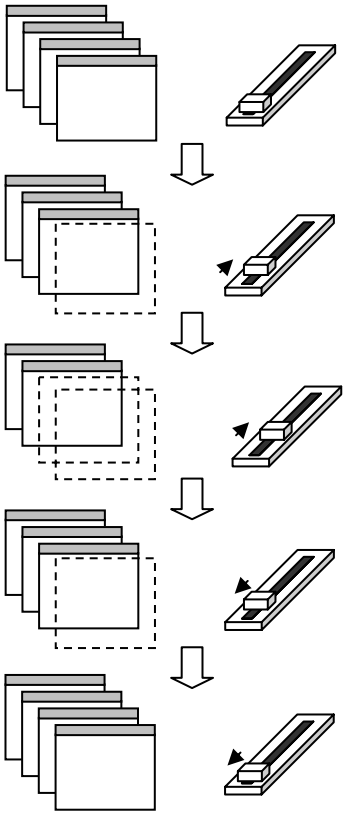

図 1 ぱらぱらウィンドウの基本動作

(1) 入力デバイスはセルフリターン式を基本とする

たとえば，図１のスライドボリュームであれば，つまみを押し込んだ指を離したり，力をぬいたりすると，自動的に初期位置につまみが戻るものとする．このようにすることで，非表示にしたウィンドウを再表示して，元の状態に戻したいときには，指を入力デバイスから離すだけで済む．

ただし，セルフリターン式の場合，いくつかのウィンドウを非表示にした状態で留めたい場合，つまみを押さえ続けなければならない．セルフリターン式と非セルフリターン式のどちらが使いやすいかはユーザの好みや，使い方によって異なると考えられる．そこで，非セルフリターン式も考慮することとする．

(2) 従来の操作を妨げない

提案する操作方法でウィンドウを非表示にしている間も，他の操作は普通に行えるようにする．このようにすることで，次節(2)(3)項に示すような操作が行えるようになる．

3.4　操作方法の例

本ユーザインタフェースを用いたウィンドウ操作の例を示す．なお，ここではウィンドウシステムとして Microsoft Windows（以下 Windows と記す）を例とし，また，入力デバイスとしては，前出のスライドボリューム型のものを用いて説明する．

(1) 基本操作：ウィンドウのブラウズ

前記したように，スライドボリュームのつまみを押していくと最上位のウィンドウから消えていき，つまみを押す力を抜いてつまみを戻すと消えたウィンドウが順に表示されていく（図１）．

このインタフェースを用いることで，目的のウィンドウを探し出すために，作業中のウィンドウを無意味に移動したり最小化したりする必要がなくなる．また，つまみを押し続けると，最終的にすべてのウィンドウが消えてデスクトップが表示されるため，デスクトップへのアクセスも容易となる．さらに，セルフリターン式であれば，つまみを放すだけで元の状態に簡単に戻せる．

(2) ウィンドウの切り替え

あるウィンドウを最上位にして作業を行いたい場合（Windows ではアクティブウィンドウにすると言う）には，(1)項の操作に従ってウィンドウをブラウズし，目的のウィンドウが見つかり，ウィンドウ上部のタイトルバーが操作できるよう

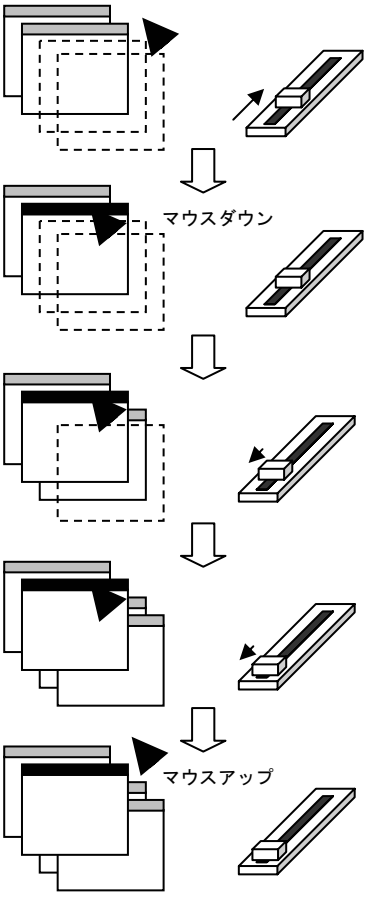

図 2 ウィンドウの切り替え

になったら，タイトルバーにマウスダウンする．続いて，つまみを放してすべてのウィンドウを表示した後に，マウスをアップすればよい（図２）．この操作方法を応用することで，任意のウィンドウを任意の奥行きに配置することも可能である．

(3) ウィンドウ間のドラッグ＆ドロップ

ウィンドウ間でアイコンをドラッグ＆ドロップするときも，(1)項の操作で移動元のウィンドウを表示させ，アイコン上でマウスダウンし，続いて移動先のウィンドウを見えるようにし，そのウィンドウの中でマウスアップすればよい（図３）．同様の手順で，文字列などのコピー&ペーストなども行える．ドラッグ&ドロップのために 2 つのウィンドウの位置を整える，という無駄な作業が必要なくなる．

４．実装

提案したユーザインタフェースの実現可能性・有用性を確認するために，Windows 上で利用できるようにするソフトウェアを試作した．次に，実装方法について述べる．

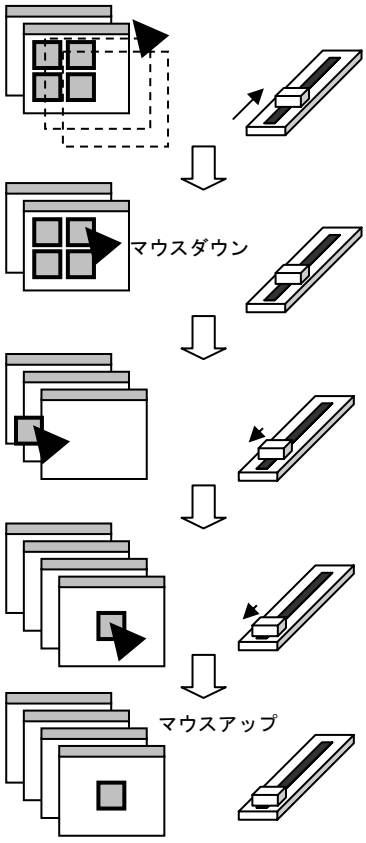

図 3 ドラッグ＆ドロップの方法

4.1 入力デバイス

入力デバイスとしては DirectX 対応の市販デバイスを対象とする．DirectX では 3 方向への変移，3 軸の回転，2 個のスライダ変移の 8 種類の入力を取得できるので，これらすべての入力に対応させる．また，セルフリターン式ではないが，ほとんどのユーザが所有するホイールマウスのホイール部でも操作できるようにする．

4.2 出力インタフェース

どのくらいのウィンドウが非表示されているのかを確認できるようにするために，本来表示されるべきウィンドウの枚数に対して，実際に表示されているウィンドウの割合を棒グラフで提示する（図４a）．この情報は，見たいときには常に見えている必要がある．そこで，この情報を表示するウィンドウ（コントロールウィンドウ）は，本ソフトウェアによる非表示ウィンドウの対象からはずし，また，常に最上位に表示する．なお，コントロールウィンドウを必要としないユーザもいると考えられるので，非表示とすることも可能とする．

4.3 構成

本ソフトウェアは主に，デバイス監視部，ウィンドウ非表示部，ウィンドウ再表示部，システム監視部から構成される（図５）．次に各部の動作を述べる．

(1) デバイス監視部

DirectX を通して入力デバイスの変移を監視し，隠し枚数＝（（変移量－バイアス）÷規定値）を計算する．ここでバイアスは，デバイスに触っていないときも若干の変移が検出されることが多いため，これを無効化する定数である．また，規定値はデバイスの変移量に対して非表示するウィンドウの枚数の割合を決める係数で，コントロールウィンドウを介して変更することができる（図４b）．

また，ホイールマウスのホイールが回転された場合は，その方向に応じて隠し枚数を増減させる．通常 Windows 上のアプリケーションでマウスの動作を監視する場合は，ウィンドウシステムが通知するイベントを用いるが，この方法では，あるウィンドウが監視できるマウスの動作は，そのウィンドウへの入力だけに限られてしまう．本ソフトウェアでは常にホイールの動きを知る必要があるため，マウスの動きも DirectX 経由で取得する．

(2) ウィンドウ非表示部

隠し枚数が増加した場合，その時点で最上位に表示されているウィンドウから増加分の枚数だけ非表示にする．ただし，Windows では，見かけ上一つのウィンドウでも内部的には複数のウィンドウから構成されている．見かけ上のウィンドウ単位で非表示を行うために，タイトルバー（ウィンドウ上部のタイトル名が表示される部分）があること，システムメニュー（ウィンドウ右上のボタン）があることを満たすウィンドウを非表示の対象とする．また，3.4 節(2)項に述べた操作を可能とするために，ドラッグ中のウィンドウは非表示にする対象とはしない．

(3) ウィンドウ再表示部

隠し枚数が減少した場合，最近非表示にしたウィンドウから減少分の枚数だけ再表示を行う．

ここで，Windows2000/XP などでは，単に再表示の命令を送るだけでは，最上位に表示されるとは限らない．これはマルチタスクを意識しているため，あるウィンドウで作業をしているときに，突然他のウィンドウが最上位にならないようにしているからである．しかしこのままでは，デバイスから変移を入力すると最上位からウィンドウが消え，変移を戻すとウィンドウが再表示されて元に戻っていくというユーザインタフェースにならない．そこで，再表示するウィンドウが最上位になるまで，最上位にする命令を発行し続ける．ただし，ドラッグしているウィンドウがある場合にこの処理を行ってしまうと，ドラッグしているウィンドウが再表示したウィンドウより下位になってしまい，4.2(2)項の機能が実現できなくなる．そこで，そのようなときには，この処理は行わない．

(4) システム監視部

ウィンドウ非表示部と再表示部では，ドラッグ中のウィンドウを特別扱いすることを述べた．このために，ウィンドウがドラッグ中であるかどうかの監視をする必要がある．

また，あるウィンドウが非表示にされた後に，そのウィンドウを持ったアプリケーションソフトウェアが終了されるなど，ウィンドウ自体が存在しなくなることが考えられる．このようなウィンドウをウィンドウ再表示部で再表示するわけにはいかない．そこで，ウィンドウの破棄も監視する必要がある．

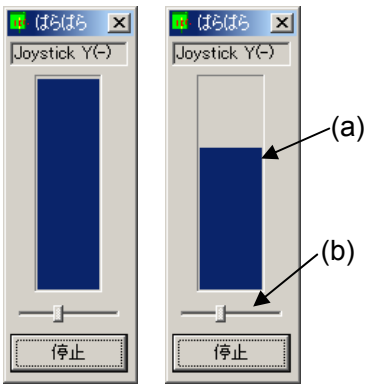

図 4 コントロールウィンドウの外観

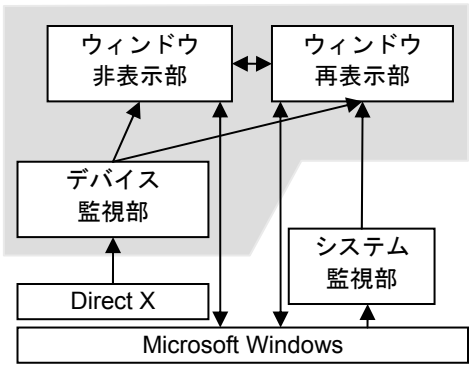

図 5 ソフトウェア構成

さらに，コントロールウィンドウに常に正確な情報を表示するためには，ウィンドウが生成されたことも監視する必要がある．

Windows では，これらの現象が発生すると，そのウィンドウや親子関係にあるウィンドウに対してイベントが送られる．しかし，マウスのイベントと同様，関係のないウィンドウがそのイベントを受けることはできない．そこで，システム監視部ではウィンドウシステムに対するイベントフックと呼ばれる技術を利用し，イベントを覗き見する．なお，すべてのウィンドウのイベントをフックするための条件に従って，システム監視部は DLL（Dynamic Linking Library）として実装する．

５．予備評価

提案したユーザインタフェースの有効性を確かめるために，単純なタスクを，マウスを用いて通常の操作方法で行うときと，提案したユーザインタフェースで行うときの操作時間を比較する予備評価実験を行った．次にこの実験について述べる．

5.1 実験方法

操作時間を比較するためのタスクは，下に位置するウィンドウは上に位置するウィンドウに完全に隠されるように重ねて置かれた 4 枚のウィンドウのうち，最下位のウィンドウを最上位にするタスク（実験 1）と，最下位のウィンドウ内のアイコンをドラッグし，上から 2 枚目のウィンドウへドロップするタスク（実験 2）とした．

このタスクをマウスで行う場合には，ウィンドウの移動またはサイズ変更だけを用いて行うこととした．実験 1 のタスクの場合，2.1 節に書いたタスクバーのボタンを押す方法などを用いると非常に少ない手数でタスクを完了できる．実験 2 の場合も Alt キーと Tab キーを押す方法を用いると少ない手数で行える．しかし，多くのウィンドウが開かれていて，どのボタンが対応するウィンドウであるかが分からないときを想定して，あくまでウィンドウの移動とサイズ変更だけで行うこととした．

ぱらぱらウィンドウを用いる場合の入力デバイスとしては，前にも例として示した指で操作するスライドボリューム型のデバイスに近いものとして，サンワサプライ社製コントローラ Real Feedback のラダーコントロール部（図 6）を用いた．このデバイスはマウスを持たない手で操作してもらうこととした．ここで，手はキーボードの操作にも用いるため，デバイスの持ち替えの煩雑さが増えることが予想される．そこでコンピュータを操作する際に用いることが少ない足で操作するデバイスとして，Logitech 社製コントローラ GT Force のフットペダル部（図 7）を用いた実験も行った．

被験者は Windows の操作に慣れた大学生 6 名で，各タスクについてマウスとペダルとラダーそれぞれについて 10 回行ってもらった．使うデバイスの順番が実験結果に与える影響を減らすために，被験者によって使用するデバイスの順番を変化させた．計測する時間は，開始を合図する音からタスクが完了するまでの時間とした．なお，タスク開始時にはマウスカーソルを画面中央に戻させ，また，できる限り正確に早く操作するように指示した．

5.2 実験結果

計測した操作時間の平均値を表 1 に示す．また被験者ごとの平均値を図 8，図 9 に示す．図内の棒は平均値を，線分は平均値±標準偏差を表している．なお，この平均値は，被験者 1 人につき 10 回計測した操作時間のうち，最長および最短操作時間を除いた 8 回から算出した．

5.3 考察

両方の実験において，提案したユーザインタフェースを用いた場合の操作時間は，マウスを用い

図 6 ラダーコントロール

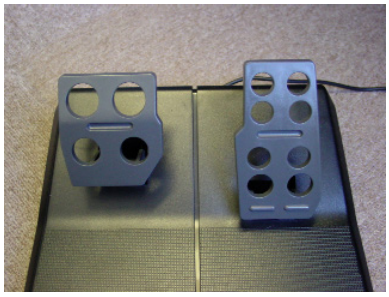

図 7 フットペダル

て通常の操作方法で行う場合の操作時間より短いことが示された．マウスとペダル，マウスとラダーとの操作時間に差異がない仮説は危険率 5%の t-検定で棄却された．

どの方法（デバイス）が操作しやすいかを口頭で尋ねたところ，ペダル（3 人），ラダー（2 人），マウスを含めどれとも言えない（1 人）に分かれ，提案したユーザインタフェースが肯定的に受け入れられたことがわかる．一方，ラダーが小さく操作しづらい，ペダルはある位置で止めるのが難しい，ウィンドウの非表示，再表示に合わせてクリック感があるとわかりやすい，との意見が得られた．デバイスの形状やフィードバックがユーザインタフェース自体の使いやすさに影響することがわかった．

表 1 平均操作時間（秒）

	実験 1	実験 2
ペダル	2.16	4.38
ラダー	2.07	3.95
マウス	4.02	5.59

６．関連研究

ウィンドウの切り替えを扱っている研究として，タスクの切り替えを容易にするユーザインタフェースの研究がある．Rooms[6]は複数の仮想的なデスクトップを切り替えられるようにすることで，一度に多くのウィンドウをデスクトップ上に表示せずに済むようにした．Data Mountain[7]では使用していないウィンドウをアイコンではなく縮小（サムネイル）表示することで，Forager[8]，Task Gallary[9]は仮想的な三次元空間に配置することで，それらのウィンドウの一覧性と検索性を高めた．しかしこれらは通常表示のウィンドウの切り替え方法は扱っていない．

通常表示のウィンドウ切り替え問題を解決する研究としては，久納らの研究があげられる．久納らは，首を傾げてウィンドウの後ろを覗き込む動作を行うと，後ろ側のウィンドウが前に出てくるというユーザインタフェースを提案している[10]．首の動作だけで目的のウィンドウを前に出すことができるが，ウィンドウの配置を予めある程度決めておく必要があるという問題点がある．これに対して提案したユーザインタフェースでは，今ま

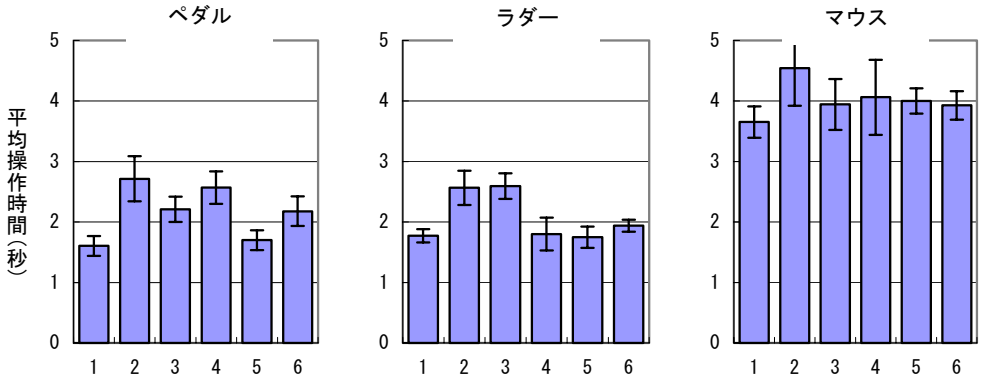

図 8　実験 1 被験者別平均操作時間

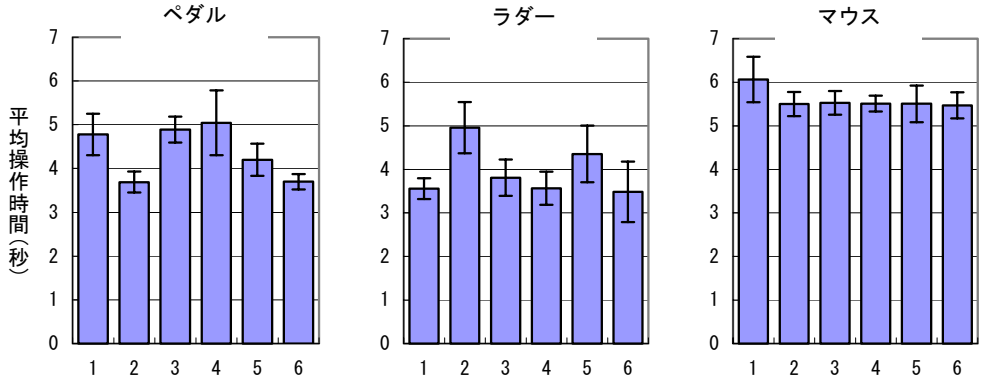

図 9　実験 2 被験者別平均操作時間

でのウィンドウの配置の方法を変える（ウィンドウの配置に気を配る）必要はない．

また，本論文で提案したユーザインタフェースは，作業を効率よく行うために複数のデバイスで操作を可能としたものと捉えることができる．Buxton らは，本来一つのデバイスで独立して行う操作を，二つのデバイスで同時に行う方法について操作性を調べ，二つのデバイスを用いることの有効性を報告している[11]．中村らは，二つのマウスを利用すると，いくつかの操作では効率が向上するとの報告を行っている[12]．二次元 GUI の操作において，複数デバイスを同時に操ることで操作できるシステムの提案としては，Toolglass[13]，Bricks[14]，T3[15]などがある．しかし，ウィンドウの切り替えに応用した事例はない．

7．おわりに

本稿ではウィンドウの切り替えに伴う操作性の問題点を解決するためのデバイスおよび操作方法を提案した．このユーザインタフェースを利用することで，複数のウィンドウが重なっている状態でも，意図するウィンドウを容易に表示させ，操作することができる．予備評価実験からは，マウスを用いる通常の操作方法に比べ，このユーザインタフェースを用いることで，下に位置するウィンドウを最上位にするタスクや，二つのウィンドウ間でアイコンを移動するタスクの操作時間が短くなることが示された．

今回提案したユーザインタフェースは，三次元目（奥行き）の操作を直感的に操作できる特徴を持ち，ウィンドウの操作以外にも，奥行きの指示が面倒であった場面，たとえば作図ソフトにおける図形オブジェクトの選択など，を解決する可能性を持っている．

今回の評価実験はあくまで決められたタスクの操作時間を比較したものであり，提案したユーザインタフェースが実際どのくらい利用される可能性あるのか，どのくらい便利なのか，および，タスクバーなどを用いるウィンドウ切り替え方法に比べて使いやすいかどうかなどは，長期的に常用してもらわなければわからない．そこで，実際に長期間に渡って利用してもらうことを通した現実的評価が今後の課題である．また，最適なデバイスの形状や構造の探求，ウィンドウ操作以外への応用を行っていきたいと考えている．

参考文献

[1] 小國健，加藤直樹：ウィンドウの切り替えを容易にするインタフェースの提案，情報技術レターズ, Vol.1, pp.201-202 (2002)

[2] 枠ピタ：http://member.nifty.ne.jp/yamazaki/WakuPita/index.html

[3] スイッチ XP：http://www.geocities.co.jp/SiliconValley-PaloAlto/8654/switch/index.html

[4] RWiper: http://www.kit.hi-ho.ne.jp/modified-r32/SoftGallery/softgal.html

[5] 窓立て：http://www.ksky.ne.jp/~seahorse/mtate2/

[6] Stuart K. Card and Austin Henderson, Jr.: A multiple, virtualworkspace interface to support user task switching, Proceedings of the CHI+GI, pp. 53-59 (1987)

[7] George Robertson, Mary Czerwinski, Kevin Larson, Daniel C. Robbins, David Thiel and Maarten van Dantzich: Data Mountain:Using Spatial Memory for Document Management, Proceedings of UIST ’98, pp.153-162 (1998).

[8] Stuart K. Card, George G. Robertson and William York: The WebBook and the Web Forager: An information workspace for the World-Wide Web, Proceedings of CHI ’96, pp. 111-117 (1996)

[9] George Robertson, Maarten van Dantzich, Daniel Robbins, Mary Czerwinski, Ken Hinckley, Kirsten Risden, David Thiel and Vadim Gorokhovsky: The Task Gallery:A 3D Window Manager, Proceedings of CHI2000, pp.494-501 (2000)

[10] 久納章寛，岡本壮平，武藤直美，中島誠，伊藤哲郎：層構造の作業環境におけるユーザ意図の把握，情報技術レターズ，Vol.1，pp.199-200 (2002)

[11] William Buxton and Brad A. Myers: A Study in Two-Handed Input, Porceengs of CHI ’86, pp.321-326 (1986)

[12] 中村聡史，塚本昌彦，西尾章治郎：ウィンドウ環境のためのダブルマウスシステムの実現について，ヒューマンインタフェースシンポジウム’99 論文集, pp.555-560 (1999)

[13] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton and Tony D. DeRose: Toolglass and magic lenses: the see-through interface, Proceedings of SIGGRAPH ’93, pp.73-80 (1993)

[14] George W. Fitzmaurice, Hiroshi Ishii and William A. S. Buxton: Bricks: laying the foundations for graspable user interfaces, proceedings of CHI ’95, pp.442-449 (1995)

[15] Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel and Bill Buxton: The design of a GUI paradigm based on tablets, two-hands, and transparency, Proceedings of CHI ’97, pp.35-42 (1997)