細長いターゲットのドラッグ開始を支援する手法とその評価

山中 祥太　宮下 芳明

ウィンドウの縁を移動してリサイズする操作のように，細長いターゲットをドラッグアンドドロップ (DnD) するには，正確な操作と十分な時間が必要である．本稿ではこういった操作を支援する手法 Cross-drag を提案し，実際の GUI 環境で利用するためのシステムを実装した．また既存のポインティング支援手法を利用した場合との比較議論によって，DnD 操作における Cross-drag の性能を考察した．さらに提案手法の有用性を検証するために，ターゲットの縦横サイズ，およびターゲット間の距離を変更した DnD タスクを行い，操作時間とエラー率を従来手法と比較した．実験により，多く条件下で Cross-drag の有用性が確認された．

Dragging and dropping narrow and long targets, such as resizing windows by moving their edges, requires accurate manipulation and takes a long time. To facilitate such tasks, we propose a new technique called Cross-drag. We developed a system to utilize this technique in a realistic GUI environment. Moreover, we discussed the performance of Cross-drag by comparing it with other pointing methods used to drag-and-drop. The experiment to evaluate our technique was conducted by dragging-and-dropping narrow targets with various width/height/gap sizes. The operation time and error rate results showed that Cross-drag outperformed the traditional pointing-based drag-and-drop under many conditions, thereby illustrating the effectiveness of Cross-drag.

1　はじめに

図 1 に例示するような，ウィンドウの外枠，あるいはレイアウトを変更するための境界線など，細長いターゲットをドラッグアンドドロップ (DnD) する場面では，正確な操作と時間が要求される．こういったリサイズやレイアウト変更などの操作は，それ自体に集中して取り組みたいものではなく，メインの作業 (ウェブブラウジングやプログラミングなど) に付随して必要になるものである．しかしながら，OS やソフトウェアによってはドラッグできるターゲットの幅が極めて細く設定されていることもあり，ユーザは慎重に操作しなければならない．こういった細長いターゲットを容易にドラッグできる手法を確立することで，WIMP 環境はさらに改善可能であると我々は考

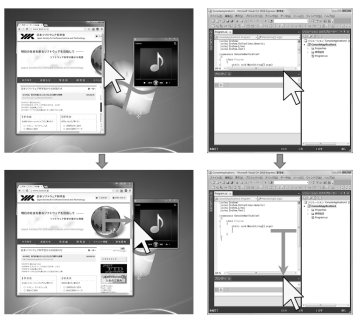


図 1　細長いターゲットを DnD する例．(左) ウィンドウのリサイズ，(右) ウィンドウ内のレイアウト変更．

A Support Technique to Begin Dragging Narrow Targets and its Evaluation.

Shota Yamanaka, 明治大学大学院理工学研究科，日本学術振興会特別研究員 DC2, Graduate School of Science and Technology, Meiji University, JSPS Research Fellow (DC2).

Homei Miyashita, 明治大学大学院理工学研究科，独立行政法人科学技術振興機構 CREST, Graduate School of Science and Technology, Meiji University, JST, CREST.

コンピュータソフトウェア, Vol.33, No.1 (2016), pp.111–125.
[研究論文] 2015 年 2 月 14 日受付.

図 2　ターゲットの幅 W と長さ H の方向.

えている．本研究の目的は，図 1 の例のように線分と直交する一軸方向に動くターゲットのドラッグ開始操作の支援である．以降は文献 [3] にならい，カーソルでドラッグするのと同軸方向のサイズを幅 W，垂直方向のサイズを長さ H とする (図 2).

従来のポインティングでのドラッグ開始操作を Point-drag と呼ぶことにする．Point-drag による DnD の手順を図 3(上) のように分解して考えたとき，ドラッグ開始に着目すると (図 3(上) の Step 2)，ターゲットの幅 W が小さいほどカーソルを乗せる操作は困難になる．一方でドロップ位置を決定する操作 (Step 4) の難易度は W に依存せず，ユーザがそのときどきの要求に応じて調整しなければならない.

小さいターゲットをポインティングする時間・精度を改善するために，従来から様々な手法が提案されてきた．例として，ターゲット上でカーソル速度を低下させる Sticky Icons [21] や，カーソルを広範囲にするエリアカーソル [9] などがあり，時間短縮やエラー率の低減に一定の効果を発揮している．こういったターゲット選択手法の 1 つであるクロッシング [2] は，線分の形状をしたターゲットをカーソルが通過することで選択を行う．ターゲットまでの距離や長さ H によっては，クロッシングはポインティングよりも高速であることが示されている [2]．このことから，クロッシングによって細長いターゲットを捕捉することで，カーソルを乗せる微細な操作が必要なくなり，Point-drag よりもドラッグ開始が容易になると考えた．この操作手法を Cross-drag と呼ぶことにする.

Cross-drag では図 3(下) のようにターゲットに衝突することでドラッグを開始する．カーソルが移動速度を維持したままターゲットを捕捉できるため，操作開始からドラッグ開始まで (図 3 の Step 3 まで：フェーズ 1 と呼ぶ) の所要時間のみならず，その後のドロップ終了まで (同じくフェーズ 2 と呼ぶ) の時間

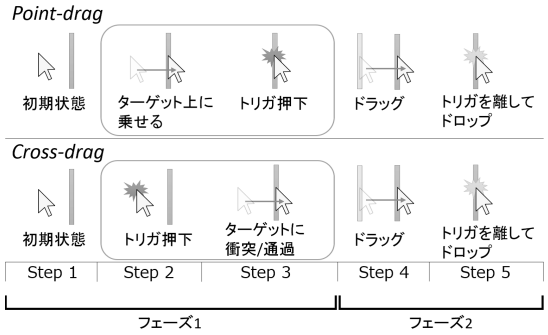


図 3　DnD 操作を手順ごとに分解したもの.

を短縮することも可能だと考えた.

本稿では Cross-drag の利点・制約を述べたうえで先行研究との比較議論を行い，実験によってタスクパフォーマンスを改善可能なターゲットの条件を求める．なお，本稿は過去の発表 [22] [23] を発展させたものである.

2　Cross-drag 手法

2.1　操作方法

図 3(下) のように，トリガ (マウスの左ボタンなど) を押している状態で，カーソルがターゲットの領域に侵入または通過した時点でドラッグ開始となり，トリガを離した時点でその場にターゲットを置く．カーソルがターゲットに乗っている状態でトリガを押下しても同様にドラッグ開始となる．従来の Point-drag との差異はトリガを押すタイミングであり，Cross-drag は先にトリガを押してからターゲットに近づくことができる．一方で Point-drag は，ターゲット上にカーソルを乗せてからトリガを押下するため，幅が細い (=W が小さい) と微細な制御が求められる．Cross-drag の利点は，トリガを押したままカーソルをターゲットに“当てる”ようにすればよく，ある程度の長さ H があれば微細な操作が必要ないことである.

ターゲットが一次元方向にのみ移動する DnD 操作では，カーソルがターゲット上から外れてしまってもドラッグを継続するように設計されることが多い．ウィンドウの枠やシークバーなどがよく知られた例である．Cross-drag でも同様に，一度ターゲットを捕

図 4 複数ターゲットへの **Cross-drag** を許す場合の挙動例．ウィンドウを左へ拡大中に，上の枠にも衝突して左上方向に拡大する．

捉するとトリガを離すまでドラッグが継続される．

2.2 複数のターゲットに衝突したときの挙動

従来手法では，Ctrl キーを押したまま再度マウスボタンを押すといった操作により，ドラッグするターゲットを追加できる．裏を返せば，コマンド操作をしない限りは，最初に捕捉したターゲットだけをドラッグする設計になっている．Cross-drag でも同様に，トリガの押下中に最初に衝突したターゲットのみをドラッグする設計にしている．

これとは異なる設計として，衝突した複数個のターゲットを同時にドラッグする方針も考えられる．こうすれば図 4 のように，ウィンドウの左枠を広げている最中に上枠にも衝突し，左上方向に拡大してく操作が可能になる．しかし，左方向にだけ拡大したいときには上下の枠に衝突しないように移動させる必要があるため，ウィンドウサイズによってはステアリング操作 (一定の幅から出ないように経路を通過する操作 [1]) が必要になり，従来は存在しなかった問題を生むことになってしまう．これを避けるためにも，既にターゲットをドラッグしている最中には Cross-drag が発生しない設計とした．複数のターゲットをドラッグ可能にする設計の利点・弊害も興味深い検討課題であるが，本稿では Cross-drag 手法の基本的な性能に焦点を当てて議論したいため，別途のコマンドを併用しない操作について考えることにする．

3 Cross-drag を利用するシステムの実装例

各種 OS にはウィンドウを自動でリサイズする機能が設けられており，簡易な操作で規定サイズに変更できる．たとえば Windows 8 に搭載されている Aero Snap では，ドラッグしているウィンドウを画面上端

図 5 ウィンドウのリサイズシステム．

図 6 ウィンドウ内のレイアウト変更システム．

に移動させることで最大化したり，左右端に移動させることで画面半分のサイズに変更できる．また Mac OS X Yosemite には，ウィンドウ上部をダブルクリックすることでコンテンツに応じてリサイズする機能がある．これらの機能で要求が満たされることもある一方で，手動でのサイズ調整に頼らざるをえない状況も存在する．前述の Mac の例では，たとえばウェブブラウザで動画投稿サイトを閲覧しているときにダブルクリックすると，推薦動画やコメント欄も含めたウェブページ全体の横幅にリサイズされる．しかし実用上は,「今は動画プレイヤの部分だけが見えるようにリサイズしたい」などという要求も生じるため，規定サイズへの自動調整では不十分な場合がある．さらにマルチウィンドウ環境で作業をしていると，他のウィンドウとの兼ね合いも考えてサイズを決定しなければならないこともあり，手動でのリサイズが求められる場面が増大する．

今回実装したウィンドウのリサイズシステム (図 5) では，Windows OS 上で起動する全てのウィンドウに対して Cross-drag によるリサイズが可能になる．聴覚的・視覚的フィードバックのために，ターゲットへの衝突時に効果音を鳴らし，カーソルを拳形状に変化させて移動方向を向くようにする．リサイズが起こるのは望ましくない状況もあるため，Cross-drag を発動しない操作パターンを事前に設定している．たとえばウィンドウ間でファイルを DnD できるようにするため，アイコンのドラッグ中にはリサイズしないようにしている．またウィンドウ内のレイアウトを変更

するシステム (図 6) も実装した．これはユーザがドラッグしたい境界線を選択してシステムに登録することで Cross-drag を利用できる．これらのシステムは Win32API を用いて実装した．

4 議論

4.1 ターゲットの配置間隔に関して

クロッシング操作の性質として，ターゲット同士が平行に並んでいるときには目的外のものを避けなければならず，ポインティング操作に近くなるという制約がある．実際の GUI 環境では，先行研究の実験条件である 256 ピクセル [2] [7] あるいは 128 ピクセル [20] などよりもターゲット同士の間隔が狭い場面もある．こういった状況下では，ターゲットから離れた位置でトリガを押してもよいという Cross-drag の利得が小さくなってしまう．極端な場合には，3 つ以上のターゲットが完全に密着しているとき，間に挟まれたターゲットを Cross-drag する操作は Point-drag と同じになる．原理的にはターゲット間に僅かでも隙間があれば Point-drag と同等以上の性能を示すはずであるが，実際の効果は評価実験で確かめる必要がある．そこで本稿では，先行研究よりも狭いターゲット間隔を設定することで，実際の GUI 環境に近い条件下での性能を検証する．

また図 4 のウィンドウを拡大していく例において，ドラッグしたい左枠の長さ H が小さい場合には，ウィンドウの上下にある横枠線同士が近接してしまう．すると最初に目的外のターゲットに衝突してしまう危険が高まり，難度が上昇すると考えられる．本稿のシステム例ではターゲットが縦横に近接した環境も含まれるが，こういった条件下での Cross-drag の有用性は未知であるため，本稿の実験で独自に検証する必要がある．

さらに，ターゲットの幅 W が大きいときには Point-drag でも問題なく操作できるはずである．すなわち，Cross-drag が有意な貢献をする W の範囲があると考えられる．よって実験では，平行に並んだターゲット間のギャップ (G とする)，長さ H，幅 W を変化させて有用性を検証する．

4.2 提案手法を使用するためのトリガに関して

システムによっては Cross-drag が行いづらい状況が生じることがある．第 3 章のウィンドウリサイズでは，ウィンドウ内の文字列を選択しているときに不用意に外枠を Cross-drag してしまうことがある．文字列選択中には Cross-drag しないように設定すると，今度は文字で満たされたウィンドウの枠を Cross-drag するのが困難になる．こういった影響を避けようと慎重に操作してしまうのはトリガを押下できる範囲が狭まることと同じであり，Cross-drag の利得が小さくなる．実用上は文字列の選択状態をキャンセルするのが容易なので例外登録しなくても問題ないが，Cross-drag 専用のトリガを設ければこのような競合が発生しないのは確かである．

トリガに関する制約の有無および影響の程度は，Cross-drag を搭載するシステムの性質に依存する問題である．本稿の実験では，特定のシステムではなく，ベースとなる手法の有用性検証を主眼とするため，Cross-drag 専用のトリガを設定しない条件で評価を行う．一連の作業中にトリガを替えて操作する条件での有用性については今後の検証課題としたい．

5 関連研究

Cross-drag はドラッグ開始操作をクロッシングで行う手法であるが，その他のポインティング支援手法を利用することも可能である．ここでは，本稿の議論対象である細長いターゲットの DnD について，Cross-drag とその他のポインティング手法との比較議論を行う．

5.1 カーソル速度変更とターゲット拡大

微細な制御をせずにポインティングを行う手法に，ターゲットとの距離に応じてカーソル速度を変更する Semantic Pointing [5] や，ターゲット上でカーソル速度を低下させる Sticky Icons [21] がある．これらを DnD に適用しても，フェーズ 1 のターゲット捕捉を容易にできると考えられる．またフェーズ 2 においても，フォルダアイコンなど特定のターゲットにドロップする操作が支援される．しかしこれらの手法は，ターゲットが多く配置されていると過度に速度減

少が起こってしまい，パフォーマンスが低下すると指摘されている [19]．また Mac OS の Dock のように，カーソルとの距離によってターゲットを拡大する手法は，近辺のターゲットが視認できなくなったり，目的のターゲットが初期位置から移動してしまうことで運動計画が崩れる問題がある [15] [24]．

上記の手法は，ポインティングデバイスの移動量に対して，カーソルがターゲットに乗っている時間を増大させることで位置制御を支援する．これはカーソルが高速なまま短時間でターゲットを通過する Cross-drag とは対極にある解決方法であるといえる．Cross-drag は，カーソルの速度およびターゲットの形状はシステム側から変更されないため，ここに挙げられている問題が生じることは原理上ない．

5.2 エリアカーソル

広範囲を指せるエリアカーソル [9] を用いることで，微細な制御を軽減するアプローチも存在する．これもフェーズ 1 のターゲット捕捉と，フェーズ 2 のドロップ位置調整を容易にすると考えられる．エリアカーソルを発展させた手法に，カーソルが直近のターゲットに乗った状態を保つ Bubble Cursor [8] や，選択操作が困難な高速移動中にエリアを拡大する DynaSpot [6] も提案されている．またスナッピングもカーソルのエリアの拡大，あるいはターゲットサイズの拡大であると解釈できる．すなわち，ターゲットまでの距離が一定以下のときにトリガを押すと，カーソルがターゲット上に移動する (またはターゲットがカーソルに吸着する) ことで，一方の実効範囲を拡大していることになる．

エリアカーソルの問題の 1 つに，ターゲットを捕捉するための最短経路が認識しづらいことが挙げられる．たとえば Bubble Cursor を使用しても，どこまで近づけばターゲットがエリア内に入るか分かりづらいため，遠くから捕捉できる機能を活かせないと指摘されている [11]．これに対し Cross-drag は一点を指すカーソルを利用しているため，上記のような認識のしづらさは生じない．ただしターゲットの配置によっては十分近づいてからトリガを押さなければならない状況があり，この制約はエリアカーソルと同様である．

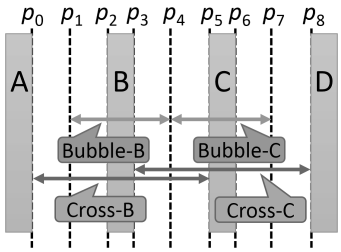


図 7 Bubble Cursor と Cross-drag で，ターゲット B と C をドラッグしたいときにトリガ押下が可能なカーソル範囲．「(手法)–(ターゲット)」の組み合わせで記載．

さらに詳細に議論するため，フェーズ 1 のドラッグ開始時におけるエリアカーソルと Cross-drag の利得を比較する．ターゲットの拡大手法やスナッピングで得られる利得もカーソル拡大と同様に考えられるため，ここでは代表して利得が最大になる Bubble Cursor と比較する．Bubble Cursor では，図 7 においてカーソルの中心が p_1〜p_4 の間にあるときにはターゲット B，p_4〜p_7 の間では C が選択される．一方 Cross-drag では，B をドラッグしたいときには p_0〜p_5 の範囲でトリガを押してからターゲットに衝突する．同様に C をドラッグしたければ p_3〜p_8 の間でトリガを押せばよい．Cross-drag ではターゲット B と C のトリガ押下可能範囲が p_3〜p_5 で共有されている．したがって，Cross-drag の方が微細な制御を要さず，この観点では利得が大きいといえる．

5.3 移動方向を考慮したターゲット選択

移動方向を考慮するエリアカーソルに，Fan Cursor [18] やアメーバ状カーソル [16] がある．これらはカーソルの移動方向の先にあるターゲットを優先的に選択する機能をもち，遠くのターゲットをより素早く選択できる．ターゲットとドロップしたい位置によっては Cross-drag と同等以上の高速な操作も可能である．たとえば Fan Cursor が図 7 の p_0 にあるとき，少しでも右方向へ移動すればターゲット B を捕捉でき，トリガ押下を許す範囲の左端が p_0 になるためである．それに加えて，ターゲットに衝突する前にドラッグを開始できるため，提案手法よりも早くターゲットの移動を開始できる．よってドロップしたい位

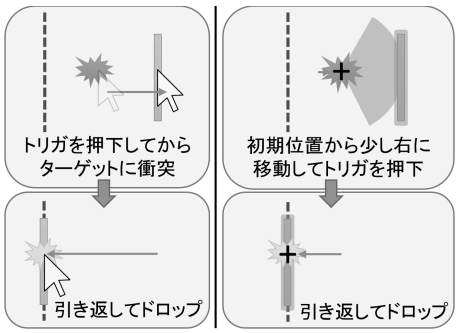


図 8　Cross-drag(左) と Fan Cursor(右) で，初期移動から引き返した方向にドロップする操作の比較．破線はドロップしたい目標位置．

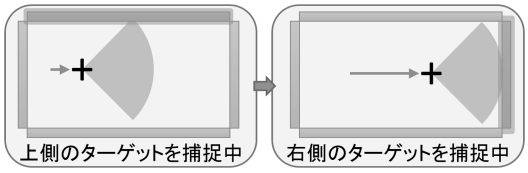


図 9　上下左右にターゲットが近接した状況．Fan Cursor は速度に応じて扇型を 90°～180°に広げるが，ここでは最も前方にまで選択できる 90°の例を示す．

置が図 8 のようにカーソルを引き返す方向にある場合には，Cross-drag よりも移動距離を短縮できる．逆に図 1(左) のように，カーソル→ターゲット→ドロップ位置，という順の配置では，結局カーソルをドロップ位置まで移動させる必要があり，Cross-drag と移動距離は変わらない．このことから，フェーズ 1 におけるターゲットの選択手法として有用なものであっても，フェーズ 1～2 を通した一連の DnD 操作に適用するとメリットが限定される状況があることがわかる．

ここまでの議論は，ウィンドウの枠のように長さ H が十分に大きく，図 7 のようにターゲットが 1 次元方向に並んでいることを想定していた．一方で図 9 のように上下のターゲットが近接している状況下では，移動方向を考慮したエリアカーソルの利得が小さくなる問題が生じる．たとえば Fan Cursor やアメーバ状カーソルでは，左右のターゲットにかなり接近しないと捕捉できず，それまでは上下のターゲットを選択できる状態になっているためである．同様に Cross-drag も，ステアリング操作を避けるためになるべくターゲット付近でトリガを押すようになり，利得が小さくなるおそれがある．よっていずれの手法も，2 次元方向に妨害刺激が近接している条件では有用性が低減するといえる．

スナッピングに移動方向を導入するアプローチも考えられる．仮に専用トリガを用いるのであれば，トリガを押してからカーソルの移動先にあるターゲットを吸着させてよいように思える．しかしここで懸念される問題にマイクロスリップがある．これはユーザがカーソル操作をするときに“迷い箸”に代表されるフラフラとした動きをしており，目的のターゲット方向へのみ移動するわけではない [10] という知見である．コントロールされた実験環境下では，実験参加者が目的のターゲット (多くの場合は 1 つだけ色分けされている) のみを目指してカーソルを移動させることも可能だと考えられるが，実際の GUI 環境では意図しないターゲットをドラッグしてしまう危険がある．あるいは熟慮してからでないとカーソルを動かすことができなくなってしまい [10]，ユーザの操作を束縛してしまうため，Cross-drag ではカーソルが衝突するまではターゲットを確定しない設計にした．

6　実験デザイン

Cross-drag が有効に機能する条件を求めるための実験を行う．細長いターゲットを DnD するタスクにおいて，Cross-drag と Point-drag の操作時間およびエラー率を比較する．以下に詳細を述べる．

6.1　タスク

図 10 のように，タスクが開始されると，画面内にスタート領域，ターゲット，障害物 (ダミーターゲット)，ドロップ領域が表示され，カーソルがスタート領域の中心に置かれる．実験参加者はスタート領域をクリックした後，ターゲットをドロップ領域に DnD する．スタートからゴールへの方向は上下左右の 4 パターン存在する．ターゲットはスタートとゴールを結ぶ 1 次元方向にのみ移動するが，カーソルは 2 次元方向に移動できる．操作時間およびエラーの計測は，スタート領域をクリックしてから，ターゲット全体が

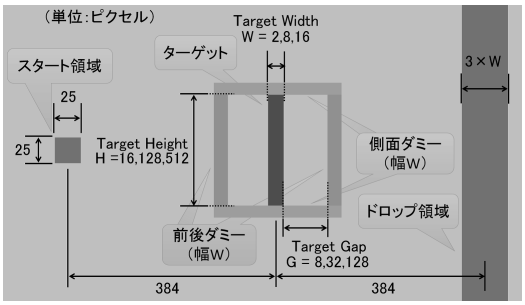


図 10 タスクの画面配置模式図．スタート位置が左の場合．

ドロップ領域に含まれるように移動してトリガを離すまでとする．なお図 10 において，G のみは中心同士の距離ではなくターゲットとダミーに挟まれた空白領域のサイズである．

4.1 節の議論を基に，ターゲットの前後左右にダミーのターゲットを設置する．移動方向の前後にあるダミーはギャップ G，側面のダミーはターゲットの長さ H に依存して提案手法の性能に影響を与える妨害刺激である．Cross-drag 操作で目的のターゲットよりも先にダミーに接触した場合は，意図しないドラッグ開始であると見なしてエラーとする．すなわち，実験参加者は図 10 の薄い色のダミーで囲まれた領域でトリガを押し，ダミーを避けて濃い色のターゲットに衝突することで Cross-drag する．

ターゲットを正しくドロップするとベル音が鳴って全画面が 1 秒間灰色で描画され，次のスタート領域・ターゲット・ダミー・ドロップ領域の配置 (画面配置と呼ぶ) が決定し，カーソルがスタート領域の中心に移動する．画面配置の選出順はランダムである．エラーが発生すると，その時点でビープ音を鳴らすとともに全画面が 1 秒間灰色で描画され，試行が中断される．その後に同一の画面配置でスタート領域のクリックからタスクをやり直す．

6.2 エラー

以下の操作をエラーに計上する．

- トリガを押した状態でダミーに衝突 (Cross-drag)
- ターゲット・ダミーのいずれにも衝突せずにトリガを離す (Cross-drag)
- ターゲットの領域外でトリガを押す (Point-drag)
- ターゲット全体がドロップ領域に入っていない状態でトリガを離す (両手法)

文献 [20] におけるクロッシングのエラーは，トリガを押したまま線分の延長線上を通過した場合と定義されている．しかし 4.1 節の議論にあるように，本実験ではターゲット同士が縦横に近接した環境での有用性も検証したい．よって先行研究よりもエラーの解釈を拡大している．

6.3 実験パラメータ

ここでは各種 OS の数値を参考に実験パラメータを設定した．参考にした数値は付録 A を参照されたい．

6.3.1 平行なターゲット同士のギャップ G

ギャップ G は 8, 32, 128 ピクセルである．これらの値は，各種 OS におけるウィンドウのカスケードのギャップを参考にしつつ，ターゲット同士がある程度近接している状況から離れている状況までを含むように設定した．

6.3.2 ターゲットの幅 W

W は 2, 8, 16 ピクセルとした．まず，「Point-drag にとってはドラッグ開始が困難なほど細いターゲットであっても，Cross-drag であれば素早くできる」という仮説を検証するために，下限値を 2 ピクセルとした．また各種 OS のウィンドウ枠のドラッグ領域サイズを包含するように，8, 16 ピクセルを採用した．

6.3.3 ターゲットの長さ H

H は 16, 128, 512 ピクセルとした．図 10 の側面ダミー同士の距離，すなわちターゲットの長さ H が小さい条件として 16，128 ピクセルを採用した．また，「ターゲットが十分に長ければ微細な制御を要さない」ことを検証するために，最大値を 512 ピクセルとした．これは，「フル HD の画面で，中程度のサイズのウィンドウ (960 × 540 ピクセル) をリサイズする」操作を想定して設定した．

6.4 実験機器等

マウスは光学式で 1000dpi のものを C-D ゲイン 1 で用いる．ディスプレイは 21.5 インチ，1920 × 1080 ピクセルの LCD を用い，Windows 7 で実験システムを全画面表示する．カーソルは長さ 25 ピクセル，

幅 1 ピクセルの線分を交差させた黒色十字を用いる．練習時に椅子の高さやマウスパッドの位置，ディスプレイの位置・角度を調節させ，各実験参加者が実験に集中できるよう配慮した．なお，実験に使用するポインティングデバイスの選定理由については付録 B を参照されたい．

6.5 実験参加者

実験参加者は情報系の大学生及び大学院生の 10 名 (男性 9 名，女性 1 名，平均 22.9 歳) であり，全員がマウス操作に習熟している．9 名が右利き，1 名が左利きであった．普段の PC 作業時にマウスを利用しているのは 9 名であり，全実験参加者が利き手で操作した．左利きの実験参加者 1 名は普段からマウスの左ボタンでクリック操作をしていたため，使用するトリガは全実験参加者とも左ボタンである．

6.6 実験手順

Cross-drag の画面配置は合計で 3(W) × 3(H) × 3(G) × 4(スタート位置)=108 パターンである．Point-drag ではダミーの有無がタスクの難易度に影響しないが，視覚刺激量を同程度にするために G を中距離の 32 ピクセルに固定して描画する．したがって Point-drag の画面配置は 3(W) × 3(H) × 1(G) × 4(スタート位置)=36 パターンとなる．

各手法での本番前に 18 試行ずつの練習を設け，タスク内容の理解および使用機器への慣れを促した．記録されるデータは (Point-drag 36 パターン+Cross-drag 108 パターン) × 10 名=1440 回分となる．

実験参加者にはタスク内容やエラーとなる操作を教示したうえで，可能な限り素早くかつエラーをせずに操作するよう伝えた．実験参加者の半数が Cross-drag を先に使用し，残りの半数が Point-drag を先に使用する．練習と本番の間，および Cross-drag 本番の 36 回ごとに 30 秒間の休憩を設ける．全試行の後，主観評価を行うためのアンケートを実施した．実験に要する時間は，事前のインストラクションから全試行終了まで 25 分程度である．

6.7 その他の実験条件の検討

タスクパフォーマンスに影響を与えるその他の要因として，スタート・ターゲット・ゴールの位置関係やサイズの比率，ポインティングデバイスの C-D ゲインやソフトウェア加速の有無など様々なものが考えられる．これらの条件によって結果が変わる可能性は残されているが，本実験では 4.1 節で議論したように，今回特に重視したいターゲットの幅，長さ，ギャップの 3 条件に絞って観察する．

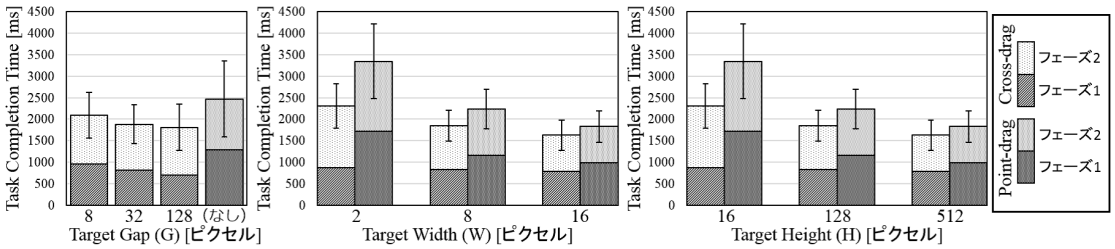
ここでは第 3 章のシステム例に含まれるような視覚的・聴覚的支援をするためのフィードバックを排除した実験デザインを採用する．すなわち，カーソルは拳型ではなく常に十字型であり，Cross-drag によってドラッグを開始したときの効果音は鳴らない．操作手法の設計にフィードバック方法を含める考え方もあるが，第 3 章のフィードバック例が Cross-drag にとって望ましいか否かは新たな議論が必要になる．本実験ではシステム例のベースとなる操作方法の有用性を検証したいと考えているため，フィードバック方法の良し悪しと操作手法の性能を切り分けて検証するべく，フィードバックは行わないこととした．同様に，Point-drag で一般的に行われるフィードバックについても搭載せず，Cross-drag と同一条件にする．

7 実験結果

7.1 操作時間

7.1.1 タスク完了時間

図 11 にパラメータとフェーズごとの操作時間を示す．実験全体で 1527 回の試行があり，その内 87 回 (5.70%) がエラーであった．エラーの無かった試行 1440 回のタスク完了時間 (フェーズ 1+2 の合計時間) に関して，ターゲットの幅 W，長さ H，手法を要因とした対応あり 3 要因分散分析 (反復測定) を行った．手法の分析は G の要因を含み，以下では手法を Cross-8，Cross-32，Cross-128，Point の 4 種類で表記する (Cross の数値は G の値)．多重比較には Bonferroni の手法を用いた．分析の結果，W ($F_{2,18} = 774.84$, $p < .001$)，H ($F_{2,18} = 56.36$, $p < .001$)，手法 ($F_{3,27} = 69.04$, $p < .001$) による主効果が見られた．多重比較では，全ての W の間に $p < .001$,



(a) ダミーターゲットのギャップG (b) ターゲットの幅W (c) ターゲットの長さH

図 11 実験パラメータごとの平均タスク完了時間.

H=16 と 128 の間に $p < .001$，H=16 と 512 の間に $p < .001$ の有意差が見られ，それぞれ値が大きいほどタスク完了時間が短縮された．H=128 と 512 の間には有意差が見られなかった．手法に関して，Cross-32 と Cross-128 の間以外で有意差が見られた．Point と Cross-8 の間に $p < .01$ の有意差が見られた以外は，全て $p < .001$ の有意差が見られた．分析の結果，G の値によらず Cross-drag は Point-drag よりも高速であり，Cross-drag の中では G が大きいほど高速であることがわかった．

手法 × H の間に交互作用が見られ ($F_{6,54} = 2.73$, $p < .05$)，Cross-8 ではいずれの H の間にも有意差がなかった．Cross-32 では H=16 と 128 の間 ($p < .001$)，H=16 と 128 の間 ($p < .001$)，H=16 と 512 の間 ($p < .01$) に有意差が見られた．Cross-128 では，H=16 と 128 の間および H=16 と 512 の間に $p < .001$ の有意差が見られた．Point は，H=16 と 128 の間および H=16 と 512 の間に $p < .001$ の有意差が見られた．また，いずれの H においても Cross-drag の方が高速であった ($p < .001$)．以上のいずれも G と H が大きいほど時間が短縮された．

また手法× W の間に交互作用が見られた ($F_{6,54} = 29.10$, $p < .001$)．手法間に有意差が見られなかったのは，W=8，16 のときに Cross-8 を使用した場合である．つまり幅 W が大きくギャップ G が狭いときには，Cross-drag は操作時間を短縮しないことがわかった．Cross-drag の中で比較すると，有意差が見られなかったは W=128 のときの Cross-32 と Cross-128 の間だけであった．つまりギャップ G が大きいほど時間は短縮されるが，幅 W が十分に大きい時には G $\geq$ 32 であれば十分高速化できていることがわかった．これら以外の組み合わせでは少なくとも $p < .05$ の有意差が見られ，G と W が大きいほど時間が短縮されていた．

7.1.2 フェーズ 1 の所要時間

フェーズ 1 の所要時間に関して同様に分析したところ，W ($F_{2,18} = 86.87$, $p < .001$)，H ($F_{2,18} = 91.22$, $p < .001$)，手法 ($F_{3,27} = 101.09$, $p < .001$) による主効果が見られた．多重比較では，全ての W の間に $p < .001$，H=16 と 128 の間および H=16 と 512 の間に $p < .001$ の有意差が見られ，それぞれ値が大きいほど短時間になった．H=128 と 512 の間には有意差が見られなかった．また Point と Cross-8 の間，および Cross-8 と Cross-32 の間に $p < .01$，その他の全ての手法の間に $p < .01$ の有意差が見られた．G の値によらず Cross-drag は Point-drag よりも高速であり，Cross-drag は G が大きいほど高速であることがわかった．

手法× H の間に交互作用が見られた ($F_{6,54} = 3.50$, $p < .01$)．H=32 と H=128 のときには，全ての手法の間に少なくとも $p < .01$ の有意差が見られ，G と H の値が大きいほど高速であった．H=16 のときには，Point と Cross-8 の間に $p < .01$，Point と Cross-32，Cross-128 の間に $p < .001$ の有意差が見られた．また H=16 のときの Cross-drag は，Cross-8 と Cross-128 の間に $p < .01$ の有意差が見られたが，Cross-32 と Cross-128 の間には有意差がなかった．すなわち，Cross-drag はいずれの G の値においても Point-drag より高速であり，G が大きくなるほど時間が短縮されることがわかった．

また手法× W の間に交互作用が見られた ($F_{6,54} = 40.23$, $p < .001$)．Cross-8 のときには，W=2 と 8 の間 ($p < .01$)，W=8 と 16 の間 ($p < .01$)，W=2 と 16 の間 ($p < .001$) の有意差が見られた．また Cross-32 のときには，W=2 と 16 の間 ($p < .05$)，W=8 と 16 の間 ($p < .05$) に有意差が見られた．また Point では，全ての W の間に $p < .001$ の有意差が見られ，W が大きいほど時間が短縮された．Cross-drag の視点では，W=16，Cross-8 のときに Point に対して有意差が見られなかったのを除き，全ての W と G の組み合わせにおいて少なくとも $p < .05$ で Point-drag よりも高速であった．

7.1.3 フェーズ 2 の所要時間

フェーズ 2 の所要時間に関して同様に分析したところ，W ($F_{2,18} = 88.12$, $p < .001$)，手法 ($F_{3,27} = 5.35$, $p < .01$) による主効果が見られた．H による主効果は見られなかった．多重比較では，全ての W の間に $p < .001$ の有意差が見られ，値が大きいほど時間が短縮された．手法については Cross-8 と Cross-32 の間に $p < .05$ の有意差が見られた．G の値に関わらず Cross-drag は Point-drag より高速であったが，有意差は見られなかった．

交互作用が見られたのは手法× W の間のみであった ($F_{6,54} = 3.16$, $p < .05$)．W=8 のとき，いずれの手法の間にも有意差は見られなかった．また W=16 のときに有意差が見られたのは Cross-8 と Cross-32 の間 ($p < .05$) のみであった．したがって，W が十分大きければ Cross-drag による時間短縮は起こらないことが分かった．最後に W=2 のときに有意差が見られたのは Point と Cross-32 の間 ($p < .05$) のみであった．よってフェーズ 2 では，Cross-drag が時間を短縮するのは限られた条件下であることがわかる．

7.2 エラー率

Cross-drag と Point-drag ではエラーを認める操作とフェーズが異なるため，各手法の合計のエラー率のみを比較する．Cross-drag の 5.26%(60/1140)，Point-drag の 6.97%(27/387) のエラー率に関して，操作時間と同様の分散分析では W のみに主効果が認められた ($F_{2,18} = 11.26$, $p < .01$)．多重比較では W=2 と 8 の間 ($p < .05$)，W=2 と 16 の間 ($p < .05$) の間に有意差が見られた．

手法× H のみ交互作用が見られ ($F_{6,54} = 3.06$, $p < .05$)，Cross-8 のときに H=128 と 512 の間 ($p < .05$) に有意差が見られた．Point では H による有意差は見られなかった．また，いずれの H に関しても Cross-drag と Point-drag の間には有意差は見られなかった．

7.3 主観評価

事後アンケートでは，各手法でのターゲット捕捉の難易度について 1:容易〜7:困難 の 7 段階で尋ねた．その結果，Point-drag が平均 4.8 (SD=0.74)，Cross-drag が平均 2.6 (SD=0.80) となり，10 名全員が Cross-drag の方が容易であると回答した．

また自由記述によって Cross-drag 操作の所感を尋ねたところ,「ギャップ G が十分に大きい場合は，多少の減速程度でスムーズにドラッグに移れた」,「G が狭い場合は Point-drag とほぼ同じ感覚で操作した．空白部分でボタンを押しても問題ない分，少しだけ楽に感じた」といった意見が得られ，Cross-drag の支援によって操作が容易になったと感じられたとの意見が見られた．一方で,「G が小さいとき，Point-drag のように [ターゲット上でトリガを押す] 操作ができても，あえてギャップの部分でボタンを押す操作をしてしまった」,「ターゲットから離れた位置からボタンを押し始めるとミスを誘発するので，G によらずターゲットの近くからボタンを押し始めるよう心がけた」といった指摘があり，Cross-drag の利点を活かす使われ方をしていなかった事例も確認された．

8 考察

8.1 ターゲット間のギャップ G

Cross-drag のギャップ G に関して，タスク完了時間は Cross-8 と Cross-32 の間，および Cross-8 と Cross-128 の間では有意差があったが ($p < .001$)，Cross-32 と Cross-128 の間には差がなく，G は 32 ピクセル以上あれば十分であると考えられる．また G の値に関わらず Cross-drag は Point-drag よりタスク完了時間が短く，ターゲット間に僅かでもギャップ

があれば Cross-drag が有効であることが確認された．

フェーズ 1 では G が大きいほど所要時間が短縮され，Cross-drag のドラッグ開始操作は G が大きいほど容易に行えることが確認された．一方でフェーズ 2 では，Cross-8 と Cross-32 の間にのみ有意差が見られた．よって「高速なままドラッグを開始できるため，フェーズ 2 の所要時間は G が大きいほど短縮される」という性質は限定的なものであった．この理由として，高速なドラッグ開始で得られる短縮時間に対して，フェーズ 2 の終盤におけるドロップ操作に要する時間の割合が大きく，Cross-drag による利得の影響が弱まった可能性が考えられる．

8.2 ターゲットの幅 W

フェーズ 1 では全ての手法× W の組み合わせにおいて Cross-drag の方が高速であり，フェーズ 2 では Cross-8，W=16 以外の組み合わせにおいて Cross-drag の方が高速であった．また手法間でタスク完了時間に有意差が無かったのは Cross-8 で W=8，および Cross-8 で W=16 の場合であった．これは W に関して Point-drag が好条件，かつ G に関して Cross-drag が最も悪条件の場合であり，Cross-drag の方が短時間であったものの有意差は見られなかった．したがって，一般的な GUI における細長いターゲットの幅 W，ギャップ G においては，いずれの W，G の組み合わせにおいても Cross-drag は Point-drag と同等以上の性能を発揮できることが確認された．

8.3 ターゲットの長さ H

ターゲットの長さ H に関して，両手法ともターゲットの長さ H が 128 ピクセルあれば，それ以上の場合と遜色なく操作できるといえる．特に Cross-drag は H が大きいほど操作が容易になるが，その場合は 128 ピクセルで十分であることが分かった．ターゲットが短い (H=16) 場合には慎重さを要するが，いずれの H に対しても Point-drag より有意に ($p < .001$) 高速化されており，H によらずドラッグ開始操作が支援されていることが確認された．フェーズ 2 の所要時間は H による差が見られなかったが，これはドロップ操作が H に依存しないタスクであるため予想通りの結果であった．したがって Cross-drag のタスク完了時間が H によらず Point-drag より短縮されていたのは，フェーズ 1 のドラッグ開始操作が支援されていたことに起因するといえる．

8.4 ドロップ領域の幅

今回の実験ではドロップ領域の幅を 6，24，48 ピクセルに設定しており，フェーズ 2 の操作もある程度慎重に行う必要があったと考えられる．これに対して，ドロップ位置を細かく指定しないようなタスク，たとえば「コンテンツ全体が見られるようにウィンドウを拡大できればよい」などという要求に対しては，ターゲットを一定距離以上ドラッグした後はどこでドロップしてもよいことになる．このような条件下では，Cross-drag はドラッグ開始からドロップまで全く減速しないことも可能であり，フェーズ 2 の所要時間が今回の実験よりも短縮されると考えられる．一方で Point-drag はドラッグ開始後に加速し始める必要があるため，本実験結果よりもフェーズ 2 の所要時間差が大きくなる可能性がある．

ドロップ位置が限定されないタスクでは，手法の性能を一般化して議論するのが困難であるため，本実験ではドロップ位置調整にもある程度慎重な操作を要するタスクを設定した．ただし実験目的はあくまでターゲットのパラメータによる影響の検証であるため，ドロップ領域の幅は「慎重さを要しつつも，ターゲット幅 W と同じにするほど微細な操作を求めなくて良い」と考え，3 × W とした．ドロップ領域の幅によって，特にフェーズ 2 の所要時間が変わることが予想されるため，今後の検証課題としてターゲット以外のパラメータを変更しての有用性評価が残されている．

8.5 エラー率

エラー率は手法間に有意差が見られず，Cross-drag によってエラー率を改善することはないことが分かった．一般的なポインティングタスクではエラー率が 4%程度 [3] とされているが，DnD タスクでは 2 回の選択操作があるのに加え，何もないところでトリガを離してしまうエラーも生じるため，マウス操作ではエラー率が 10%以上になる [14] との報告がある．

これに対し本実験の Point-drag のエラー率は低めの 6.97%であった．この原因としては，文献[14] ではターゲットの幅 W が 8〜64 ピクセルであったのに対し，本稿の実験では W が 2〜16 ピクセルと全体的に小さい条件だったために，実験参加者が速度よりも精度を優先していた可能性が考えられる．

またマウスによるクロッシング操作は 3%程度のエラー率になる[20] とされているが，本実験の Cross-drag のエラー率は 5.26%であった．Cross-drag はターゲットの捕捉がクロッシング，ターゲットのドロップ位置の決定はポインティングで行うため，文献[20] の 3%と文献[3] の 4%を加算した 7%程度のエラー率だと予想されたが，こちらも Point-drag と同様の理由でやや低めになったと考えられる．Cross-drag ではカーソルが高速なままターゲットに衝突することを許すため，実験参加者の戦略によってはダミーへの衝突が多く発生し，Point-drag よりもエラー率が高くなることも予想された．しかし G と H の値に関わらず Point-drag より悪化することはなかった．また Point-drag にとって最も好条件の W=16 の場合においても Cross-drag のエラー率は悪化しないことが確認された．

8.6 Cross-drag が効果を発揮する条件

Cross-drag の性能はターゲットの条件ごとに次のようになる．

- G: ターゲット同士が密着している場合には Point-drag と同一の操作になるが，わずか (8 ピクセル) でもターゲット間にギャップがあれば Point-drag より操作時間が改善される．G が 32 ピクセルあれば，それ以上ギャップがあるときと操作時間に差はない．
- W: 幅に関わらず Point-drag と同等以上に高速である．W が大きく (8，16 ピクセル)，かつギャップ G が小さい (8 ピクセル) 場合には Point-drag と操作時間に差がない．
- H: 長さに関わらず Point-drag より操作時間が改善される．H が 16 ピクセルのときより 128 ピクセルの方が操作時間が短縮され，128 ピクセル以上の長さでは差はない．

実験結果と考察をまとめると，Cross-drag は Point-drag よりもエラー率を悪化させることなく，多くの条件下で操作時間のみを改善する手法であるといえる．先行研究で提案されている手法では，条件によって従来手法より性能が悪化する例もあり，間接的な比較ながらこの点においても Cross-drag の有用性が認められると考える．たとえばターゲットが密集した環境で Bubble Cursor[8] を使うと，一点を指すカーソルよりも操作時間および主観的な操作性が悪化することが分かっている[17]．またターゲットを拡大する手法[15][24] は，ターゲット間の空白を利用した DnD タスクで操作時間が増大すると報告されている[19]．既存の GUI に新たな操作手法を導入するとき，従来手法よりパフォーマンスを悪化させないことは最低条件であると考える． Cross-drag はこの要件をクリアしつつ，Point-drag より操作時間を短縮することが多かったため，ドラッグ開始操作の支援に有用であることを確認した．

8.7 制約

今回はマウスを利用したが，たとえば指での直接タッチでは G=8 だと Cross-drag するのに不十分であり，操作時間が改善されない可能性もある．逆に，他のデバイスでは Point-drag がより困難になり，Cross-drag を使うことで改善度合いが大きくなることも考えられる．また専用トリガを必須とする適用先では，指やペンでの直接タッチのみでは Cross-drag を利用できないため，キーボードなど他のデバイスを使用するか，何らかの操作でモードを切り替える必要がある．こういった複数デバイスの連携やモード切り替えの手間を含めた Cross-drag の性能は今回の実験結果からは議論できず，将来の検討課題として残されている．したがって今回の考察および得られた知見は，ポインティングデバイス群の中から特にマウスを使用した実験の結果から導かれたものであるという制約がある．

8.8 フィードバック方法

第 3 章で述べたシステムでは，視覚的フィードバックとして拳型のカーソルを表示し，またドラッグ開

始時に音を鳴らして聴覚的フィードバックを加えていた．本実験ではこれらのフィードバックをせずに性能を評価したが，今後は適切なフィードバック方法を検討したいと考えている．たとえば聴覚的フィードバックについて，当初はドラッグ中に効果音を継続して鳴らす実装案もあったが，これは煩わしく感じられるであろうと考え，衝突時に一瞬だけ鳴るように変更した経緯がある．効果音を加えるタイミングや音の種類によっても Cross-drag の操作性・快適さが変わってくると考えられるため，より良いデザインを検証していきたい．

また視覚的フィードバックについて，第 3 章で述べた拳型のカーソルは，線分のターゲットを押し込んでいくイメージを伝えるために採用したデザインである．そのため線分が拳の先端に表示されるようにしたが (図 5，6)，線分が拳の中央に来るようなデザインにすれば，あたかもターゲットを握ってドラッグするような外観になる．実際に Adobe Reader などでは画面を掴んでスクロールするために拳型カーソルが採用されており，ユーザの理解が促される可能性がある．拳型以外にも，親指と人差し指でピンチしているデザインによって，細いものを摘んでいることを表現することも可能であると考える．また，ドラッグされているターゲット側に視覚効果を加える方法や，それらを組み合わせるなど，様々なバリエーションがある．本実験では操作手法の性能評価に焦点を絞ったが，フィードバックを加えた場合の性能や，操作感がどのように変化するかを定性的に評価することで，適切なフィードバック方法を検討する必要があると考える．

9 おわりに

本稿では，細長いターゲットの DnD においてドラッグ開始操作を支援する手法 Cross-drag を提案し，実際の GUI 環境で Cross-drag を利用するためのシステムを複数実装した．また，ターゲットの配置間隔やトリガの必要性の議論を通して，提案手法の利点と制約について議論した．先行研究のターゲット選択手法に関して，DnD 操作に適用した場合の利点や，移動距離の利得が得られないターゲット条件などを考察した．実験では各種 OS の設定値を参考にした DnD タスクを課し，支援なしの場合と比較した性能評価によって有用性を確認した．

謝辞 本研究は JSPS 科研費 15J11634 の助成を受けたものです．

参 考 文 献

[1] Accot, J. and Zhai, S. : Performance Evaluation of Input Devices in Trajectory-Based Tasks: An Application of the Steering Law, in *Proc. of CHI '99*, 1999, pp.466–472.
[2] Accot, J. and Zhai, S. : More than Dotting the i's — Foundations for Crossing-Based Interfaces, in *Proc. of CHI '02*, 2002, pp.73–80.
[3] Accot, J. and Zhai, S. : Refining Fitts' Law Models for Bivariate Pointing, in *Proc. of CHI '03*, 2003, pp.193–200.
[4] Bi, X., Li, Y. and Zhai, S. : FFitts Law: Modeling Finger Touch with Fitts' Law, in *Proc. of CHI 2013*, 2013, pp.1363–1372.
[5] Blanch, R., Guiard, Y. and Beaudouin-Lafon, M. : Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation, in *Proc. of CHI '04*, 2004, pp.519–526.
[6] Chapuis, O., Labrune, J. -B. and Pietriga, E. : DynaSpot: Speed-Dependent Area Cursor, in *Proc. of CHI '09*, 2009, pp.1391-1400.
[7] Forlines, C. and Balakrishnan, R. : Evaluating Tactile Feedback and Direct vs. Indirect Stylus Input in Pointing and Crossing Selection Tasks, in *Proc. of CHI '08*, 2008, pp.1563–1572.
[8] Grossman, T. and Balakrishnan, R. : The Bubble Cursor: Enhancing Target Acquisition by Dynamic Resizing of the Cursor's Activation Area, in *Proc. of CHI '05*, 2005, pp.281–290.
[9] Kabbash, P. and Buxton, W. A. S. : The "Prince" Technique: Fitts' Law and Selection Using Area Cursors, in *Proc. of CHI '95*, 1995, pp.273–279.
[10] 栗原一貴：マイクロスリップの Human Computer Interaction 研究への応用，生態心理学研究，Vol.4, No.1(2009), pp.7–13.
[11] 桑原智大, 山本景子, 倉本到, 辻野嘉宏, 水口充：ゴーストハンティング: 疑似オブジェクト提示によるオブジェクト選択最適化手法, 情処研報 2011-HCI-144 (12), 2011, pp.1–8.
[12] Luo, Y. and Vogel, D. : Crossing-Based Selection with Direct Touch Input, in *Proc. of CHI '14*, 2014, pp.2627–2636.
[13] MacKenzie, I. S., Kauppinen, T. and Silfverberg, M. : Accuracy Measures for Evaluating Computer Pointing Devices, in *Proc. of CHI '01*, 2001, pp.9–16.
[14] MacKenzie, I. S., Sellen, A. and Buxton, W. A.

S. : A Comparison of Input Devices in Element Pointing and Dragging Tasks, in *Proc. of CHI '91*, 1991, pp.161–166.
[15] McGuffin, M. and Balakrishnan, R. : Acquisition of Expanding Targets, in *Proc. of CHI '02*, 2002, pp.57–64.
[16] 重森晴樹, 入江健一, 倉本到, 渋谷雄, 辻野嘉宏 : バブルカーソルの GUI 環境への適用と拡張, インタラクション 2006 論文集, 2006, pp.21–22.
[17] 重森晴樹, 入江健一, 倉本到, 渋谷雄, 辻野嘉宏 : GUI 環境でのバブルカーソルの実用的評価, 情報処理学会論文誌, Vol.48, No.12(2007), pp.4076–4079.
[18] Su, X., Au, O. K. -C. and Lau., R. W. H. : The Implicit Fan Cursor: A Velocity Dependent Area Cursor, in *Proc. of CHI '14*, 2014, pp.753–762.
[19] 築谷喬之, 高嶋和毅, 朝日元生, 伊藤雄一, 北村喜文, 岸野文郎 : Birdlime Icon: 動的にターゲットを変形するポインティング支援手法, コンピュータソフトウェア, Vol.28, No.2(2011), pp.140–152.
[20] Wobbrock, J. O. and Gajos, K. Z. : A Comparison of Area Pointing and Goal Crossing for People with and without Motor Impairments, in *Proc. of ASSETS '07*, 2007, pp.3–10.
[21] Worden, A., Walker, N., Bharat, K. and Hudson, S. : Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons, in *Proc. of CHI '97*, 1997, pp.266–271.
[22] Yamanaka, S. and Miyashita, H. : The Nudging Technique: Input Method without Fine-Grained Pointing by Pushing a Segment, in *Adjunct Proc. of UIST '13*, 2013, pp.3–4.
[23] 山中祥太, 宮下芳明 : Cross-drag: 細長いターゲットのドラッグを容易にする操作手法, WISS 2014 論文集, 2014, pp.97–102.
[24] Zhai, S., Conversy, S., Beaudouin-Lafon, M. and Guiard, Y. : Human On-line Response to Target Expansion, in *Proc. of CHI '03*, 2003, pp.177–184.

A 付録: 各種 OS，アプリケーションにおける細長いターゲットの設定値

第 6 章の実験パラメータは，広く普及している OS の設定値を参考にする．OS は調査時点で最新版である Windows 8.1，Mac OS X Yosemite 10.10.2，Ubuntu 14.04.1，Debian 7.8.0 である．

ウィンドウの設定値を表 1 に示す．リサイズ用のドラッグ幅は，各種 OS がデフォルトで搭載しているファイルエクスプローラを対象に調査した．またウィンドウ間のギャップは，ファイルエクスプローラを連続起動してカスケード表示されたときに，ウィンドウ間でカーソルがドラッグ可能状態にならない領域である．枠の位置によって差がある場合には値の範囲を記載している．Ubuntu でエクスプローラを起動すると，画面内の四隅に順に密着して起動するためデータなしとした．また，Point-drag での操作に極めて長時間を要するため，幅 W=1 の例 (Debian のウィンドウの左右の枠) のみ本稿の実験パラメータの範囲から除外した．

表 1 OS ごとのウィンドウの設定値 (ピクセル)

OS	枠のドラッグ幅	ギャップ
Windows	7	17〜18
Mac	7	13
Ubuntu	10	データなし
Debian	1〜3	32

B 付録: 実験に用いるポインティングデバイスの検討

クロッシングの有用性を検証したオリジナルの文献 [2] では実験にスタイラスが用いられており，机上の入力面への操作で画面上のカーソルを操作する間接制御タイプであった．その後，スタイラス [7] や指 [12] での直接制御，マウスおよびトラックボール [20] などでもクロッシングの有用性が確かめられている．今回の実験はデバイス間の比較を目的としないため，これらの中から 1 種類のデバイスを用い，ターゲットのパラメータによって手法の性能がどう変化するかに焦点を当てることにした．

そこで本実験で用いるデバイスを検討するため，事前に 10 名の実験参加者に対し，普段 PC 操作をするときのポインティング方法を複数回答可で尋ねた．利用者が 1 名以上いた項目の結果を表 2 に示す．この中から，実験参加者がデバイスの学習に要する時間を軽減するために，人数の多かったマウス，タッチパッド，指でのタッチスクリーン操作を候補とした．まずタッチスクリーンについて，指での直接タッチ操作では十分に時間をかけても一点に正確に触れることはできないことが分かっている [4] ため，デバイスの影響が大きくなることで操作手法の効果を観察しづらくなると考えて除外した．次に第一著者がマウスとタッチパッドで実験を試行したところ，両手法ともにマ

表 2 普段 PC を操作するときのポインティング手段 (人)

マウス	9
タッチパッド	7
指で直接タッチスクリーン操作	6
スタイラスで直接タッチスクリーン操作	3
トラックポイント	1

ウスの方が短時間 (タッチパッドの 50〜60%程度) であった．この結果は，マウスはタッチパッドに比べてポインティングタスクが 60%程度の時間で済む [13]，ステアリングタスクは 50%程度の時間で済む [1] などのデバイス比較をした知見とも合致する．この結果を考慮し，微細で高速な制御を要求する本実験はマウスを用いるのがよいと判断した．

山 中 祥 太

2013 年明治大学大学院理工学研究科博士前期課程修了，同年より同研究科博士後期課程に在籍．2013 年より明治大学理工学部助手，2015 年度より日本学術振興会特別研究員 DC2，現在に至る．ユーザインタフェース研究，特にポインティング手法の研究に興味を持つ．情報処理学会 HCI 研究会学生奨励賞，同研究会貢献賞，第 15 回ヒューマンインタフェース学会論文賞を受賞．

宮 下 芳 明

千葉大学工学部卒業 (画像工学)，富山大学大学院で音楽教育を専攻，北陸先端科学技術大学院大学にて博士号 (知識科学) 取得，優秀修了者賞．2007 年度より明治大学理工学部に着任．2009 年度より准教授．2013 年より同大学総合数理学部先端メディアサイエンス学科所属．2014 年より教授，現在に至る．日本ソフトウェア科学会，VR 学会，ヒューマンインターフェース学会，情報処理学会，ACM 各会員．