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

山中 祥太　　宮下 芳明*

概要.　表計算ソフトのセルをリサイズする操作のように，GUI で細長いターゲットをドラッグアンドドロップする場面では，操作時間と精度が要求される．ときには操作ミスによって生じた状態の回復に労力を割き，本来の作業から注意がそれることもある．従来からボタンやアイコンの選択を支援するためのポインティング手法は多く提案されているが，本稿ではその中から Crossing によってドラッグを行う操作方法 Cross-drag を提案する．そしてエリアカーソルやターゲットを拡大する手法，カーソル速度を変更する手法などとの比較議論をしたうえで，提案手法の新規性や優位性について考察する．さらに提案手法を実際の GUI 環境で利用するために，ウィンドウのリサイズ，表計算ソフトのセルのリサイズおよび位置調整，ウィンドウ内のレイアウト変更操作に適用した 3 つのアプリケーションを実装した．

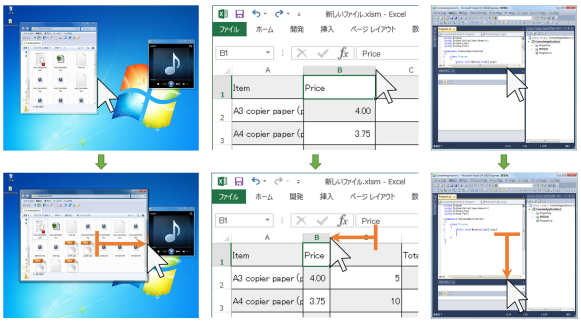

図 1. 細長いターゲットを DnD する例.（左）ウィンドウのリサイズ,（中）表計算ソフトのセルのリサイズ・位置調整,（右）ウィンドウ内のレイアウト変更.

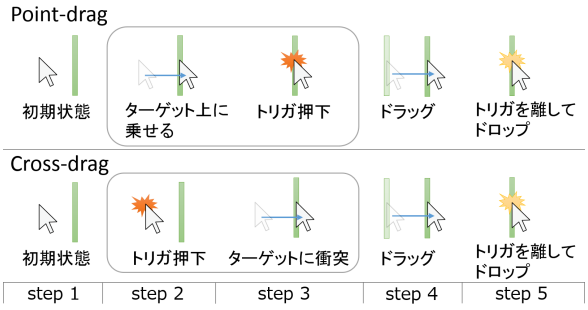

図 2. DnD 操作を手順ごとに分解したもの.

1　はじめに

細長いターゲットをポインティングするには操作精度と時間を要する [1] が，ドラッグアンドドロップ (DnD) でも同様に，ターゲットが細長いために操作が煩雑な場面がある．図 1 のように，ウィンドウの外枠をドラッグしてリサイズする操作や，表計算ソフトのセルのサイズ・位置を調整する操作，ウィンドウ内にある境界線を動かしてレイアウトを変更する操作などである．もしもターゲットが非常に細く，究極的には幅が 1px しかなくとも快適に DnD できる手法があれば，本来の作業である表計算やタイピングにより集中できることだろう．

本稿では文献 [1] に従い，細長いターゲットのサイズが大きい方向を長さ W，他方を太さ H と呼ぶ（すなわち $W \gg H$ である）．従来のポインティングによる DnD（Point-drag と呼ぶことにする）の一連の操作を図 2（上）のように分解して考えたとき，ドロップ位置の調整 (step 4) の難度はターゲットの太さ H に依存せず，ユーザがそのときどきの要求に応じて微調整しなければない．一方でドラッグ開始に着目すると，カーソルをターゲットに乗せる操作 (図 2（上）の step 2) の難度はターゲットの太さ H に依存する．つまり「ターゲットが細いほど DnD は難しい」と感じるのは，ドラッグ開始時の影響が大きい．

解決手段として考えられるのは，カーソルが接近したときにターゲットを拡大する手法 [2] や，カーソルを低速にする手法 [3][4]，広範囲を選択できるエリアカーソル [5][6][7] を用いる方法，ターゲットが通過することで選択したと見なす手法 [8] などである．本稿ではこれらのうち，最後の Crossing 操作 [8] をドラッグ開始に利用した Cross-drag 手法を提案する．Cross-drag では図 2（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．Crossing は GUI 上でのボタン押下やメニュー選択などを対象操作として提案されており，たとえばマウスでのクリックやペンタブレットでのタップなどといった，

* Shota Yamanaka, 明治大学大学院 理工学研究科 新領域創造専攻 ディジタルコンテンツ系, Homei Miyashita, 明治大学大学院 理工学研究科 新領域創造専攻 ディジタルコンテンツ系, 独立行政法人科学技術振興機構 CREST

確定操作を置き換える手法として論じられている．文献 [8] の実験では，同じ難度のタスクであれば操作時間がポインティングと同等か短くなることが分かっており，我々はこの有用性を DnD でも発揮できると考えた．Crossing 手法と他のポインティング支援手法の比較は関連研究の章で詳しく行う．特に，DnD を支援するにはどのポインティング手法を採用するのがよいか，という比較議論がこれまでは不十分だと考えており，この観点でも考察する．

2 Cross-drag 手法

2.1 操作方法

図 2（下）のようにドラッグ操作のトリガを押下している状態で，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．また，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．ここでいう「トリガを押下」とは，マウスの左ボタンを押したり，スタイラスをタッチスクリーンに接触させるなどといった，ドラッグを開始するための行為を指す．図 2 で比較しているように，従来の Point-drag との差異はトリガを押下するタイミングである．Point-drag で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，ターゲットが細い場合には微細なカーソル制御が求められる．一方で Cross-drag は，トリガを押下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，ターゲットがある程度の長さ W をもっていれば微細な操作が必要ない．

通常の Point-drag によってウィンドウ間でファイルを移動している最中にウィンドウリサイズが発生してしまうなど，Cross-drag するのが望ましくない状況もあるため，機能の発動を除外するパターンを適用先のアプリケーションごとに設定する．また一般的な DnD 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ている（逸脱可能な範囲を限定していることもある）．スクロールバーやウィンドウの枠，シークバーなどがよく知られた例である．提案手法でもこれらと同様に，一度ターゲットに衝突するとトリガを離すまでドラッグが継続される．

2.2 Cross-drag 中の他のターゲットへの衝突

ドラッグ中に他のターゲットを追加でドラッグするためには，従来手法では Ctrl キーを押して再度トリガを押下するといった別途のコマンドが必要である．つまりポインティングデバイス単体の操作では，ドラッグ中のターゲットをドロップするまで他のターゲットを DnD できない．提案手法でも同様に，トリガを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．

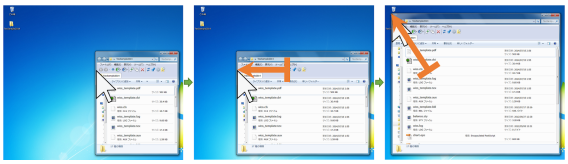

図 3. 複数ターゲットへの Cross-drag を許す場合の挙動例．ウィンドウを左へ拡大中に，上の枠にも衝突して左上方向に拡大する．

これとは異なるアプローチとして，あるターゲットを Cross-drag している最中に他のターゲットに衝突した場合には，それらを同時に移動させる設計も考えられる．こうすることで，たとえば図 3 のように，ウィンドウの左枠を Cross-drag で広げている間に上枠にも衝突し，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ていく必要があるため，ウィンドウサイズによっては微細な Steering 操作（一定の幅から出ないように経路を通過する操作 [10]）が必要になり，従来は存在しなかった問題を新たに生むことになってしまう．これを避けるために，提案手法では単一のターゲットのみ DnD する方針をとった．こうすることでドラッグ開始以後の挙動（図 2 の step 4〜5）が従来手法と同一になり，ユーザにとって新たに学習する必要のあるモードを step 2〜3 のみにできる．

2.3 効果

図 2（上）のように，従来の Point-drag ではカーソルをターゲット上に乗せる時点 (step 2) と，ドロップ位置を決定する時点 (step 4) で合計 2 回の位置調整をする．これに対し Cross-drag では，ターゲットの長さ W が十分であれば 1 回目の位置調整が粗くてもよくなる．したがって図 2 のように，カーソル→ターゲット→ドロップ位置，という順に配置されているときには，トリガを押した状態で最初からドロップ位置だけを目指すようにカーソルを動かせばよい．すなわち，合計 2 回のポインティング操作を，条件によっては「ドロップ位置をポインティングする」という 1 回の操作に置き換えられる．さらに，たとえば図 1(左) において「エクスプローラの左枠をディスプレイの左端まで広げよう」と思ったときには，トリガを押下したままカーソルを勢いよく左に移動させるだけでよく，ユーザが細長いターゲットを DnD するときの意識しなければならないことを大幅に削減できる．

3 アプリケーション例

提案手法を実際の GUI 環境に適用したアプリケーションについて述べる．なお，これらの例のうち，ウィンドウのリサイズと，表計算ソフトのセルの位

図 4. ウィンドウのリサイズアプリケーション.

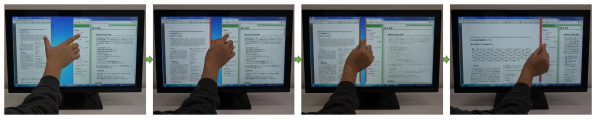

図 5. タッチ操作によるウィンドウのリサイズ.

置調整のアプリケーションは，筆者らが以前にデモ発表を行った [9] ものの改良版である．

3.1 ウィンドウのリサイズ

ウィンドウの外枠を DnD してリサイズするとき，ポインティング位置がずれて問題が生じることがある．枠の外側をポインティングして最前面ウィンドウを切り替えてしまったり，右枠の内側をクリックして意図せずスクロールするといったものである．こういった場合は，状態を戻すために再度ウィンドウを切り替えたりスクロールし直してから，再びウィンドウの枠をドラッグする必要がある．本アプリケーション（図 4）は Win32API を使用し，全てのウィンドウに対して提案手法によるリサイズができる．また，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，カーソルを赤色の拳形状に変化させて移動方向を向くようにする．

このアプリケーションは間接的なカーソル操作のみならず，指でのマルチタッチ入力においても有効に機能する．図 5 に示したのは，まず 2 つのウィンドウをドッキングするように外枠同士を独立に移動させ，その後に境界部分を 2 本指で摘むように密着させてリサイズする例である．この「2 つのウィンドウの境界部分を摘んで位置調整する」という作業においても，各指がそれぞれ最初に衝突したターゲットをドラッグし続けるという操作はカーソル操作の場合と一貫している．ここでも視覚的および聴覚的フィードバックをするために，指がターゲットに衝突したときには音を鳴らし，カーソルの代わりにドラッグ中のターゲットを赤く表示している．指でのタッチ操作はカーソル操作以上に微細な操作が困難であり，十分に時間をかけても正確にタッチするのは困難であることがわかっている [11]．Cross-drag であればターゲット上に指を正確に乗せる必要がなくなり，粗い操作でも満足にドラッグを開始できる．

3.2 表計算ソフトのセルの位置調整，リサイズ

多くの表計算ソフトでは，行番号の英字または列番号の数字が書かれた箇所の境界線をドラッグすることでスプレッドシートのセルをリサイズする．ポインティングミスをして行番号（または列番号）が表示された箇所をクリックすると 1 行（列）選択状態になってしまう．この場合，ウィンドウの操作ミスの例とは異なり，状態を戻すための操作は必要ないが，こちらもウィンドウのリサイズと同様に微細な操作が求められる．

本アプリケーションは VBA を使用し，セルを囲う全て境界線をドラッグ可能なターゲットと見なして Cross-drag を行う．こうすることで，ドラッグ対象のターゲットに十分な長さを持たせている．

3.3 ウィンドウ内のレイアウトの変更

プログラミングの IDE やファイルエクスプローラ，高機能なテキストエディタのように，ウィンドウ内を複数領域に分割して利用するソフトは多く存在する．ユーザは境界線を DnD することで各領域の位置やサイズを変更する．この境界線もウィンドウの外枠の例と同様に細長く，DnD が困難なものがある．中には Windows 版 Evernote(Ver. 5.5.2) のようにドラッグ可能な境界線が 1px だけ設けられているソフトもあり，タッチではなくカーソル操作であっても DnD が困難な事例がある．

実装したアプリケーションでは，ウィンドウリサイズと同様に視覚的・聴覚的フィードバックのある Cross-drag 操作によってレイアウトを変更することができる．実用上の制約として，ユーザは事前にドラッグしたい境界線をポインティングによる選択でシステムに登録する必要がある．これはウィンドウ内の境界線をシステム側から正確に特定することが現時点で困難なためであるが，将来的にはウィンドウ内にある境界線のサイズや連結関係などから自動で求めるように改善したい．

4 議論

4.1 ターゲットの配置間隔

スタイラスによって間接的にカーソルを操作する環境では，ターゲット同士が 256px 離れた場合の Crossing 操作の有用性が確認されている [8]．また指での直接タッチによる Crossing でも，ターゲット間の最小距離を 166px（実距離で 50.8mm）に設定して有用性が確認されている [12]．

しかし Crossing 操作は原理上，ターゲット同士が近接しているときには目的外のものを避ける必要があり，ポインティング操作に近くなってしまう．つまり，粗い制御でもよいという利得が小さくなるのである．これと同様に Cross-drag でも，長さ W が同じターゲットが 3 つ以上近接しており，間に挟まれたターゲットをドラッグする場合には有用性が低減する．また，3 つ以上のターゲットが完全に密着している場合には，間に挟まれたターゲットの Cross-

drag は Point-drag と同じになる．原理上はターゲット間に少しでも隙間があれば利得が生まれ，Point-drag と同等以上のパフォーマンスを示すが，そのターゲット間隔と性能向上の関係を評価するのは今後の課題としたい．

4.2 提案手法を使用するためのトリガについて

アプリケーションによっては，Cross-drag が行いづらい状況が生じうる．例として 3.1 節のウィンドウのリサイズアプリケーションについて考える．ウィンドウ内のコンテンツをドラッグして選択しているときに不用意に外枠を Cross-drag してしまったり，逆に Cross-drag でリサイズしたいときにコンテンツを範囲選択してしまう，といった競合が起こりうる．またタッチ操作ではコンテンツのスクロールが発生することもある．こういった影響を避けるためにウィンドウの外側から Cross-drag することも可能だが，利得が小さくなるうえに，背後に別のウィンドウがあればそちらをクリックしてしまう．また影響度にも差があり，リサイズしたいときにウィンドウ内のコンテンツを選択してしまうという前者の操作ミスはキャンセルが容易だが，逆に後者の不用意なリサイズの開始は元に戻す作業が煩雑である．ユーザがこれを避けるために慎重に操作しようとすると，それはトリガ押下が可能な範囲が狭まることと同等であり，提案手法による利得が小さくなってしまう．

筆者らのデモ発表 [9] では，Cross-drag を発動するために専用のトリガを設けることで操作の競合を避けていた（多ボタンマウスを使用）．通常のポインティング操作とは別途のトリガとして，他にもキーボードの特定のキーであったり，ユーザが機能を割り当てられるボタンをもったスタイラスを利用する方法などがある．これは DnD のために特別なポインティングデバイスを利用することをユーザに強いることになる．あるいはマウスを利用しているのであれば，アメーバ状カーソル [13] で利用されているように，左右ボタンの同時押下をトリガにすることも可能である．

理想的には別途のトリガを利用したりマウスに限定することなく操作できる手法が望ましいと考えるが，本稿で述べたアプリケーション例のうち，3.2 節の表計算ソフトだけは別途のトリガが必須であることがわかっている．これは，境界線を通過して複数のセルを選択する操作と，その境界線を Cross-drag する操作がシステム側から判別するのが不可能なためである．セルと境界線という 2 種類のドラッグ可能なターゲットが密接して並んでいる表計算ソフト特有の制約であるといえる．逆に言えば，別途のトリガなしで提案手法を利用したい場合には，表計算ソフトのセルリサイズはふさわしくない適用先の 1 つであることがわかる．あらゆる場面で普遍的に利用できる手法を目指すべきなのか，あるいは専用のものを使ってでも便利に操作できることを目指すのかという思想によっても意見が分かれるところであり，この点についても会議で議論したい．

5 関連研究

ここでは本稿の議論対象である細長いターゲットのドラッグを支援する研究について述べ，提案手法の新規性と優位性を明らかにする．その他の DnD に関わるものや，ドロップ位置調整（図 2 の step 4）を支援する手法については割愛する．

5.1 カーソル速度変更とターゲット拡大

カーソルの微細な制御をせずにポインティングを行う手法に，カーソルの速度をターゲットとの距離に応じて変更する Semantic Pointing[3] や，ターゲット上でカーソルの速度を低減させる Sticky Icons[4] がある．これらは相対的にターゲットを拡大する手法と見なすことができる．また Mac OS の Dock のように，カーソルとの距離に応じてターゲットサイズを変更する手法も利用されている．これらのうち，カーソル速度を変更する手法は，ターゲットが密に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問題が指摘されている [14]．また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 [2]．

カーソル速度低減，およびターゲット拡大手法のいずれも，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細いターゲットを通過することを許す提案手法とは対極にある解決方法であるといえる．提案手法では，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てないため，ここに挙げられている問題が生じることはない．

5.2 エリアカーソル

カーソルを，一点を指すシングルポインタではなく，広範囲を指せるエリアカーソル [5] にすることで，微細な制御を軽減するアプローチも存在する．これをさらに発展させて，カーソルが常に直近のターゲットを捕捉した状態を保つ Bubble Cursor[6] や，選択操作が困難な高速移動中にエリアを拡大する DynaSpot[7] も提案されている．エリアカーソルでは，複数のターゲットが配置されている場合に「カーソルに近いが選択したくないターゲット」がエリアに含まれてしまい，どの地点まで移動すれば目的のターゲットを捕捉した状態になるのかがユーザにとって把握しづらい問題がある．そうなると，できるだけ目的のターゲットに近づくように移動してしまい，

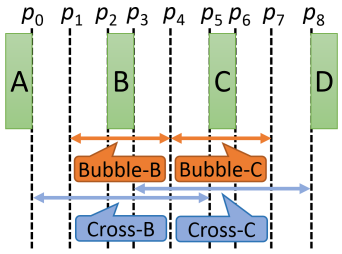

図 6. Bubble Cursor と Cross-drag で，ターゲット B と C を DnD したいときにトリガ押下が可能な横方向の範囲．「（手法）–（ターゲット）」の組み合わせで選択可能な範囲を示している．

遠くても選択できるという効果が機能しなくなってしまう [15]．これに対し提案手法はあくまでシングルポインタのカーソルを利用するため，操作時の曖昧性を排している．

さらに詳細に議論するため，ここでエリアカーソルと提案手法における利得を比較する．なお，ターゲットの拡大手法によって得られる利得もカーソル拡大と同様に考えられるため，ここでは代表してエリアカーソルと比較する．直近のターゲットを選択する Bubble Cursor を利用した場合には，図 6 のようにカーソルが p_1〜p_4 の間にあるときにはターゲット B が選択され，p_4〜p_7 の間ではターゲット C が選択される．一方で Cross-drag では，ターゲット B をドラッグしたいときに，カーソルを p_0〜p_5 の範囲に移動させてからトリガを押下し，ターゲット B に衝突する．またターゲット C をドラッグしたい場合には p_3〜p_8 の間でトリガを押下すればよい．Cross-drag ではターゲット B と C の選択開始可能範囲が p_3〜p_5 で共有されている．エリアカーソル・Cross-drag ともにトリガを押下し始める位置が本来よりも広くとれる手法だが，ドラッグを開始するために，（図 6 における）横方向の微細な制御が必要ないことの利得は Cross-drag の方が大きいことがわかる．一方でエリアカーソルは，2 次元方向に遠いターゲットを選択することできる利点も持ちあわせており，その分だけ DnD 全体の操作時間を短縮できることが期待される．本稿ではターゲットの長さ W はある程度大きいと仮定しているため，その範囲内にカーソルを移動させるのは困難ではなく，本稿の議論してきた観点では長さ W 方向への移動を容易にすることが優位性をもつわけではない．

5.3 移動方向を考慮したエリアカーソル

移動方向を考慮するエリアカーソルに，Fan Cursor[16] や，アメーバ状カーソル [13] 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ことで，Cross-drag と同等の選択開始範囲を得ることも可能である．たとえば Fan Cursor が図 6 の p_0 にカーソルがあるときに，少しでも右方向へ移動すればターゲット B を選択することが可能になり，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．この利点が有効にはたらくのは，ドロップしたい位置が図 7 のようにカーソルを引き返す方向にある場合である．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，ポインティングによる選択手法として有用なものであっても，DnD の一連の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．

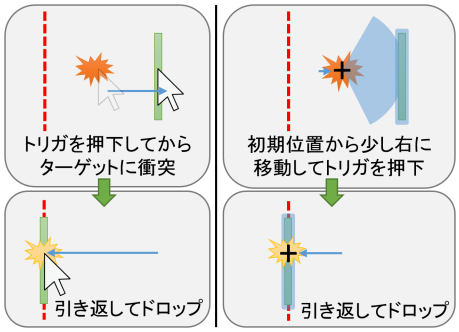

図 7. Cross-drag（左）と Fan Cursor（右）で，初期移動から引き返した方向にドロップする操作の比較．赤色の破線はドロップしたい目標位置．

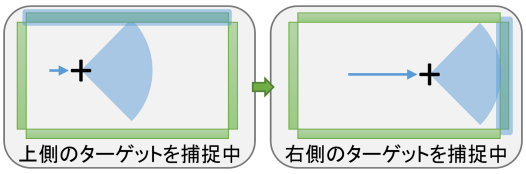

図 8. 上下左右にターゲットが近接した状況．Fan Cursor は速度に応じて扇型を 90° 〜180° に広げるが，ここでは前方に最も遠くまで選択できる 90° の例を示している．

ここまでは主に，ウィンドウの枠のようにターゲットの長さ W が十分に大きく，ターゲットの配置が図 6 のように 1 次元方向に並んでいることを想定した議論であった．一方で図 1（中）のように表計算ソフトの横長のセルを左右方向にリサイズする例では，Fan Cursor やアメーバ状カーソルでは（扇型の範囲やアメーバ形状のパラメータにもよるが）上下のボーダーラインを選択してしまい，意図しないターゲットを DnD する問題が生じうる．このような状況では目的のターゲットにかなり接近する必要があり，エリアカーソルによって得られる利得が小さくなってしまう（図 8）．したがって，図 6 のように 1 次元方向にのみターゲットがある場合には，移

動方向を考慮したエリアカーソルも適用しうるが，2次元方向にターゲットが近接している図8のような状況では効果的ではない．さらに，カーソルがどこまで進んだときに捕捉対象が切り替わるのかが分かりづらい問題もある．これはつまり，同一の操作方法を適用した場合にメリットを享受できる状況が限定されているといえる．一方でCross-dragも，目的のターゲットの遠くでトリガを押下してから近づけると，上下に衝突しないように移動していく必要があるため，なるべくターゲット付近でトリガを押下するようになり，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．どの程度ターゲット同士が離れていれば有用性を発揮できるのかは，先行研究 [16][13] において図8のような条件下で実験が行われておらず議論が困難だが，提案手法による実験を現在実施中であり，結果を基にした比較も今後行っていきたい．

6 まとめと今後の課題

細長いターゲットのDnDが困難なことに着目し，そのうちドラッグ開始を容易にする手法について検討した．従来研究のうち，Crossing操作を用いた手法Cross-dragの有効な点や制約について議論し，その他の手法と比較して新規性と優位性を考察した．そして提案手法を実際のGUI環境で利用するためのアプリケーションを複数紹介した．

筆者らの以前のデモ発表では，特別なトリガを利用することで従来のポインティング操作との競合を一切排除していたが，体験者から得られた評価には「たしかに便利だが，普通のマウスでも使えるような操作にしてほしい」というものが多かった．よって本稿では通常のトリガでも操作できるように仕様を変更し，そのためのアプリケーション（3.1，3.3節）を実装したうえで，新たに生じうる問題について考察した．さらに，アプリケーションの性質から別途のトリガを避けることが困難なもの（3.2節の表計算ソフト）があることも説明し，提案手法を適用した場合の利点/制約を議論した．

操作手法をさらに改善するためのアイデアも存在する．第一著者が3.1節のウィンドウリサイズアプリケーションを常駐させて使用してきた限りでは，リサイズしたいときにはウィンドウの枠に対して，トリガを押下してすぐに・ほぼ直角に・勢いよく衝突するクセがついてくるように感じている．これについてはよりシステマティックな分析が必要だが，トリガを押下してからの時系列的な挙動や，ターゲットに進入する角度・スピードも考慮することで，システム側から通常のポインティング操作とCross-dragを判別できる可能性があると考える．綿密に観察されたものではないため議論では取り上げなかったが，こういった改良を施しての評価も行っていきたい．

参考文献

[1] Accot, J. and Zhai, S. Refining Fitts' law models for bivariate pointing. In *Proc. of CHI 2003*, pp.193–200 (2003).

[2] McGuffin, M. and Balakrishnan, R. Acquisition of Expanding Targets. In *Proc. of CHI 2002*, pp.57–64 (2002).

[3] Blanch, R., Guiard, Y. and Beaudouin-Lafon, M. Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation. In *Proc. of CHI 2004*, pp.519–526 (2004).

[4] 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 1997*, pp.266–271 (1997).

[5] Kabbash, P. and Buxton, W.A.S. The "prince" technique: Fitts' law and selection using area cursors. In *Proc. of CHI 1995*, pp.273–279 (1995).

[6] Grossman, T. and Balakrishnan, R. The Bubble Cursor: Enhancing Target Acquisition by Dynamic Resizing of the Cursor's Activation Area. In *Proc. of CHI 2005*, pp.281–290 (2005).

[7] Chapuis, O., Labrune, J.-B. and Pietriga, E. DynaSpot: speed-dependent area cursor. In *Proc. of CHI 2009*, pp.1391–1400 (2009).

[8] Accot, J. and Zhai, S. More than Dotting the i's — Foundations for Crossing-Based Interfaces. In *Proc. of CHI 2002*, pp.73–80 (2002).

[9] Yamanaka, S. and Miyashita, H. The Nudging Technique: Input Method Without Fine-grained Pointing by Pushing a Segment. In *Adjunct Proc. of UIST 2013*, pp.3–4 (2013).

[10] Accot, J. and Zhai, S. Performance Evaluation of Input Devices in Trajectory-Based Tasks: An Application of the Steering Law. In *Proc. of CHI 1999*, pp466–472 (1999).

[11] Bi, X., Li, Y. and Zhai, S. FFitts Law: Modeling Finger Touch with Fitts' Law. In *Proc. of CHI 2013*, pp.1363–1372 (2013).

[12] Luo, Y. and Vogel, D. Crossing-based selection with direct touch input. In *Proc. of CHI 2014*, pp.2627–2636 (2014).

[13] 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, pp.21-22 (2006).

[14] 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, pp.140–152 (2011).

[15] 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), pp.1–8 (2011).

[16] Su, X., Au, O.K.-C., Lau., R.W.H. The implicit fan cursor: a velocity dependent area cursor. In *Proc. of CHI 2014*, pp.753-762 (2014).