平成 17 年度

筑波大学第三学群情報学類

卒業研究論文

題目 時間情報を利用した板書の自動補正システム

主専攻　知能情報メディア

著者名　酒井 慎司

指導教員　田中 二郎　三末 和男　志築 文太郎　高橋 伸

要　　旨

近年プロジェクタやプラズマディスプレイといった大画面デバイスを用いたプレゼンテーションや授業を目にする機会が増えてきた。しかしそれらは、完成済みの資料を大画面で表示し口頭で説明するといったようなものがほとんどであり、その場に応じて説明の内容や順番を柔軟に変更することはしにくい状況にある。それに対し、これまで長く授業などの場で用いられてきた板書は、その場で書くために時間はかかるが、そのような対応が可能な非常に柔軟な手法である。

これまで板書はほとんどがその場だけのものとして扱われてきたが、近年の印刷できるホワイトボードや電子ホワイトボードの出現により、保存され再利用することが可能となった。本研究では、板書を保存し再利用することを想定し、そのためにリアルタイムで板書を補正し、また再利用の際に板書の流れを効率的に再現するシステムを開発した。

目次

図目次

第1章　序論

1.1 背景

近年、プラズマディスプレイやプロジェクタなどの大画面デバイスを目にする機会が増えてきた。また、同時にそれらの大画面デバイスを用いた授業やプレゼンテーションを目にする機会も増えてきた。

しかし、それら大画面を用いた授業やプレゼンテーションの場合、多くはすでに完成済みの資料を大画面で表示し、口頭で説明するといったものである。そのため、たとえばその場の話題や聞き手の質問に応じて板書の内容や順番を変更するといった自由な授業形態はとられにくい状況にある。

しかし、これまで長く授業などの場で用いられてきた板書は、実際にその場で書くために時間がかかるといったことはあるものの、その場に応じて書く内容や順番を自由に変更することが可能な非常に柔軟な手法である。また板書は会議の場において、アイデアや意見を持ち寄る共有スペースとして用いられることもある。その中で書いたり消したりを繰り返しアイデアの想起に繋がることは多く見受けられるであろう。

そこで、これら大画面デバイスと板書を組み合わせ、電子ホワイトボードのような手書き入力の可能な大画面デバイス上で板書を行うことで、新たな利点が生まれるのではないかと考えた。

1.2 板書の持つ可能性

これまで、板書はほとんどがその場だけのものとして授業や会議の時間が終わると消されてしまっていた。しかし最近では会議などのアイデアを出し合った板書をカメラに収めることが行われたり、書いたものを印刷できるホワイトボードも出現してきた。また Tivoli[1]のような電子ホワイトボードの出現が板書というものに大きな影響を与えた。

それはまず、板書を電子的に保存することが可能になったということである。つまり板書を半永久的に保存することができ、それを複製したり、配布したり、またウェブ上で公開したりすることでそれまでほとんどその場だけのものであった板書を再利用することが可能になった。そして、板書の再利用が可能になったことで以下のような利点が考えられる。

- 良い板書を共有できる

現在生徒はあらかじめ割り当てられた教師の授業しか受けることができない。そのため 1 つの授業に対して見ることのできる板書も 1 つであり、それ以外の板書を目にする機会はほとんどない。しかし、板書が保存され、自分が受けていない授業の板書も見ることができれば、生徒はより自分に合った授業を見つけることが出来るのではないかと思われる。

また、板書が使い捨てされている現状では、たとえば同じ教科を教える教師同士がお互いの板書を目にする機会はほとんどない。しかし、板書を保存しそれをお互いに評価しあい意見交

換をすることができれば、授業研究の面でも有益な効果が得られるのではないかと考えている。

・ アイデアの種として再利用できる

また、会議などの場では意見やアイデアを持ち寄るスペースとして板書が用いられる。そういった場では最終的な結果よりも、むしろそのプロセスが重要な意味を持つと考えられる。そこで、板書を時間軸に沿って保存し再現することができれば、あるアイデア想起のプロセスを再確認したり、たとえばそのプロセスをその場にいなかった人に見せたりすることで、また新たなアイデアの種となり得るのではないかと考えた。

1.3 作成シーンでの問題点

板書を再利用することを考えた場合、その板書は多くの人の目に触れ、また何度も繰り返し使用されることが考えられる。しかし、板書を保存し再利用することを想定するといくつかの問題点があると考えられる。たとえば、現在授業などで板書を作成するシーンでは以下のような場面がしばしば見受けられる。

① 手振れによってストロークが乱れてしまう。

これはまず板書というものが、人々が普段行っている紙の上での筆記作業とは異なっていることが原因と考えられる。それは文字を書く面の違いである。多くの人にとって実際に手で文字を書くときには、机などの水平面の上で行う場合がほとんどであり、またそういった水平面上で書くことに慣れていると思われる。しかし板書はそれとはまったく違う垂直な面に向かって文字を書くことであるので、当然水平面上で書くこととは勝手が違う。

また、それに加えて文字を書く際の支点が存在しないということが原因と考えられる。普段紙の上で文字を書く際には、自然と机の上に書き手の肘を置き、それを支点としてペンを動かすことでぶれることのないストロークを書くことができている。しかし、板書では書く面が垂直であるために肘を置くことができない。そのため紙の上に比べ、手振れによるストロークの乱れが多く発生すると考えられる。

② 文のベースラインが乱れてしまう。

これは、①の原因でもある書く面が垂直であること、また、もう 1 つの原因としてスケールの違いが挙げられる。板書は、多くが他の人に見せることを目的として用いられるため、自然と紙の上のものと比べて全体のスケールを大きく書く必要がある。そのため、自分が書いた文字や文の全体像を把握しにくくなり、結果としてたとえば文字が右上がりになってしまうなどのことが起こるのである。

③ 図形を正確に描くことが難しい。

説明のために円を描こうとしているのだが、うまく描けず何度も描いたり消したりを繰り返すといった場面を目にすることがある。これは、手書きで図形を正確に描くことが難しいとい

うことを示している。板書は構成や文などを自由に書けるという特徴を持っているが、その中で図形を用いることは多く見受けられる。しかし、思うような図形をフリーハンドで正確に描くことは難しく、また図形を用いる場合の多くはある程度の正確さが要求される。

これらの事例は全て、結果的に板書を見にくくする原因となっている。しかし、板書を保存し再利用するということは、多くの人の目に触れ、また何度も繰り返し使用されることが考えられるため、ある程度見やすくする必要がある。そこで、板書を再利用する際にこれらの問題を解決すれば板書をもっと見やすい形で多くの人に提供することができると考えられる。

1.4 再利用シーンでの問題点

また板書を再利用する際、最終的な結果を見せるだけでは見る側はどこに注目してよいのか迷ってしまう。また板書では、一度書いたものを消してまた書いたり、同じ図や式を、数値だけ変えて使いまわしたりといったことも行われるが、最終結果だけを見せたのではその過程までは追うことができない。

そこで、再利用する際にはそういった問題を解決し、流れに沿った形で提供することが必要であると考えられる。

1.5 本研究の目的

そこで本研究では、現在の黒板やホワイトボードのような柔軟な操作性を残した上で、1.2 のように板書を保存し再利用するために以下のことを行うシステムを開発する。

- リアルタイムで板書の清書を行う。

  1.3 で述べたように、板書は人が手書きで行うものであるため当然乱れることがある。そのため保存し再利用するためには、ある程度の清書を行う必要があると考えられる。しかしそのために時間を費やすことは結果的に作業の量を増やすだけである。また、その際の清書の作業はほとんどが 1.3 で述べたようなストロークレベルの問題を解消することである。そこで、板書をする際にリアルタイムで処理を行うことでユーザの作業量を増やすことなく清書を行うことができる。

- 見る際に板書作成の過程を効率的に再現する。

  1.4 で述べた問題を解決し、流れに沿った形で板書を見せるために、本研究では時間軸に沿ってストロークを再生することで板書の流れを再現しようと考えた。しかし、ただストローク単位で再生したのでは、たとえばひとつひとつの文字の筆順まで再現されてしまい非常にたどたどしい。そこで、ストロークを文字や文といった単位でまとまりづけを行いそのまとまりごとに再生することで板書の流れを効率的に、また忠実に再現できるのではないかと考えた。そしてこの手法によって、画面単位では追うことのできなかった「消す」という作業まで追うこ

とができる。

第2章　　システムの構想

本研究では上述の背景をもとに、板書を自動的に補正し、またストロークを時間軸に沿って再生することで板書作成の過程を再現するシステムを開発した。以下にシステムの大きな 2 つの特徴について述べる。

2.1 板書の自動補正

板書は人が手書きによってするものである。そのため、個人個人の癖や手振れなどによって見にくくなってしまうことがある。また、板書を共有し資料として使用することを考えた場合、資料は多くの人の目に触れ、また何度も繰り返し使用されることが考えられるため、どんな板書であってもある程度見やすくする必要があると考えた。

そこで、本システムでは板書の自動補正として以下の操作を行う。

- 手振れの軽減
  手振れによるストロークの形状の乱れを軽減する。

- ベースラインの統一
  ベースラインの乱れを補正し、文字や図形の並びを自動的に揃える。

- 図形の認識･整形
  板書でよく用いられる図形について、手描きで描いたものを認識し、整形する。

2.2 ストロークのまとまりづけ

本システムでは、板書を資料として使用する際に最終結果だけを見せるのではなく、時間軸に沿ってストロークを再生することで授業や会議の流れを再現する。授業の板書の場合は、授業の流れを再現することでどこに重点が置かれているのかがわかる。会議の板書の場合は、あるアイデアの発端となった板書の流れを再現することでアイデア想起のプロセスを再確認できる。また板書特有の「一度書いたものを消してまた書く」といった操作や「同じ図を数値だけ変えて使いまわす」といった操作まで追うことができる。

しかし、ただ時間軸に沿って単純にストロークを再生すると、たとえばひとつひとつの文字の筆順まで再現されてしまい効率がよくない。また、見ている側としてはむしろたどたどしく感じるのではないかと思われる。そこで本システムではストロークをある程度のまとまりにして管理する。再生の際にはそのまとまりごとに再生することで授業や会議の流れを効率的に再現することが出来る。

第3章　開発したシステムの説明

このシステムでは、ユーザは画面上で自由な描画を行うことができる。またユーザの描画したストロークはリアルタイムで自動的に補正され、図形は自動的に認識され整形される。描画を終え再生を行うと、その板書の流れが効率よく再現される。

3.1 画面の説明

システムを起動するとまず下図のような画面が表示される。

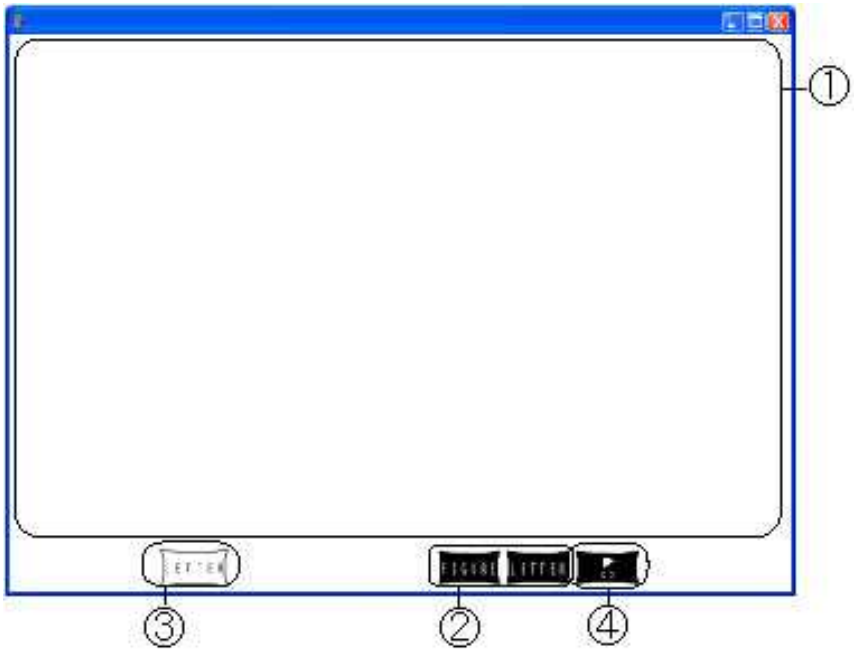


図3-1:　起動時のウィンドウ

① 描画スペース
　・・　自由にストロークを描き板書を行う
② モード切り替えボタン
　・・　フリーストロークモードと図形モードの切り替えを行う
③ 再生ボタン
　・・　板書の再生を行う
④ カレントモードウィンドウ
　・・　現在のモードを示す

3.2 モード

描画のモードには以下の 2 つのモードがある。

- フリーストロークモード
  文字や絵など自由なストロークを書くモードで、ここで描画を行うと書いたストロークの手振れによる形状の乱れが軽減され、文や式のベースラインも自動的に揃えられる。システムのデフォルトのモード。

- 図形モード
  板書によってよく用いられる図形の描画を行うモード。手描きで図形を描くと、正確な図形に整形される。

モードの切り替えはボタンによって行う。また現在のモードがカレントモードウィンドウに表示される。

3.3 手振れの軽減

ストロークを引くと、下図のようにストロークが滑らかにされ手振れによるストロークの乱れが軽減される。板書のように支点の不安定な状態で行われる描画においてこれは有効である。

図は左のようなストロークの乱れが、システムの処理によって右のように軽減された様子である。

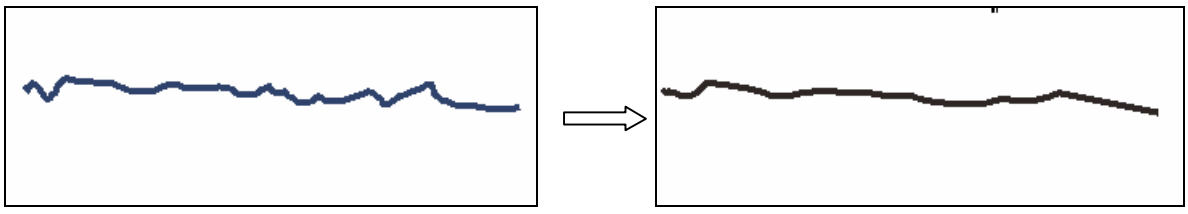


図3-2: 手振れの軽減

3.4 ベースラインの統一

文字がだんだんと右上がりになってしまったなどの場合にベースラインの乱れが自動的に補正される。縦書きか横書きかは自動的に認識され、文の最初の文字の位置に合わせられる。

図 3-3 は、板書シーンでよく見られる右上がりになった文字列のベースラインが自動的に揃えられた様子である。また、図 3-4、3-5 はそれぞれ縦書きと横書きの際の処理の様子を示している。

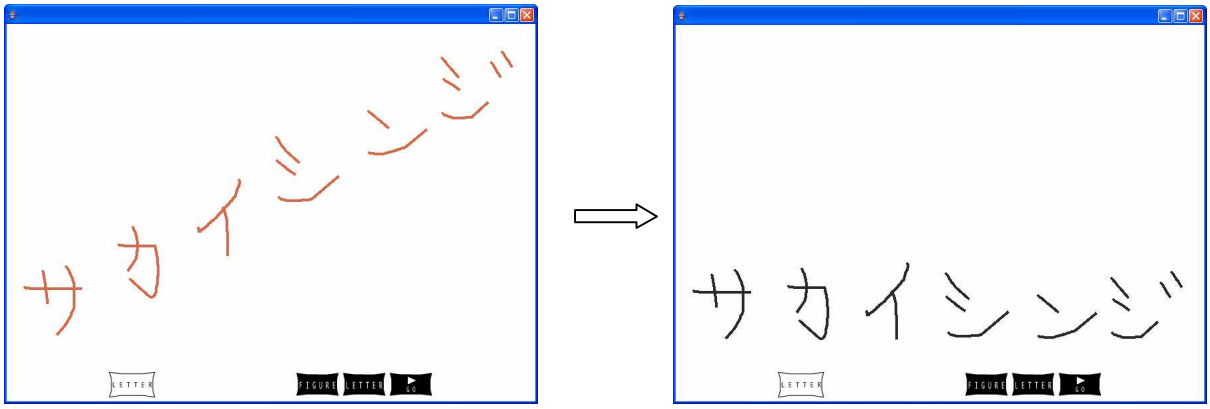


図3-3:　右上がりになった文字列のベースラインが揃えられる

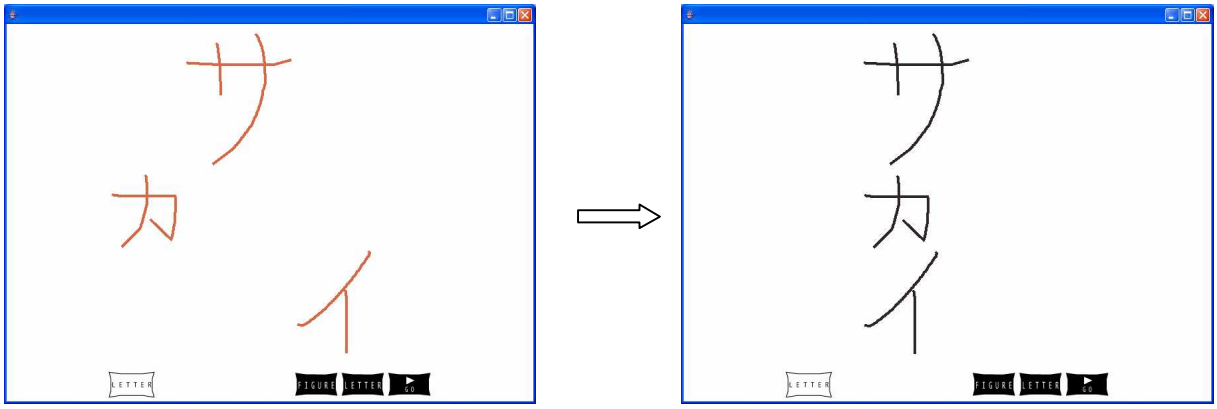


図3-4:　ベースラインの統一(縦書き)

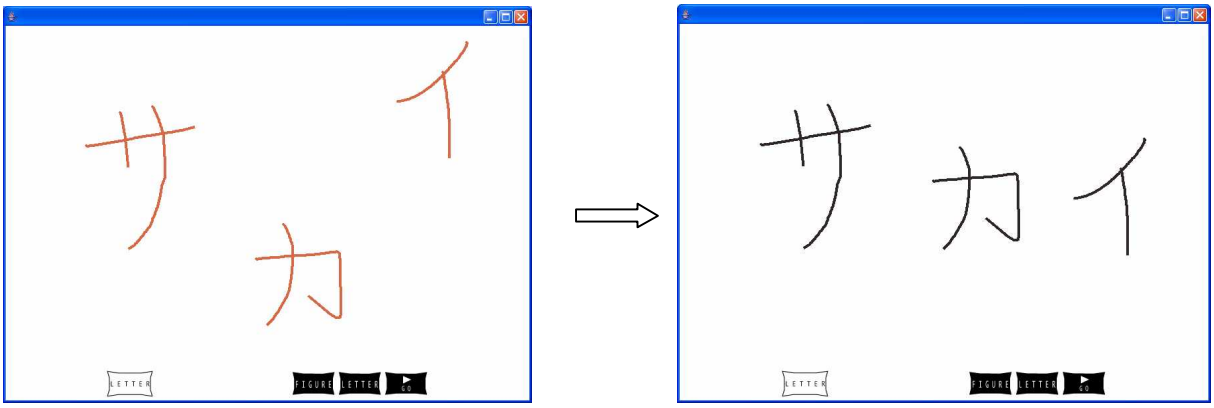


図3-5:　ベースラインの統一(横書き)

3.5 図形の整形

手書きで図形を正確に描くことは難しい。そこで手書きのストロークで描いた直線、矩形、楕円に関して自動的に認識と整形が行われる。

図 3-6、3-7、3-8 はそれぞれ直線、楕円、矩形の認識と整形の様子である。

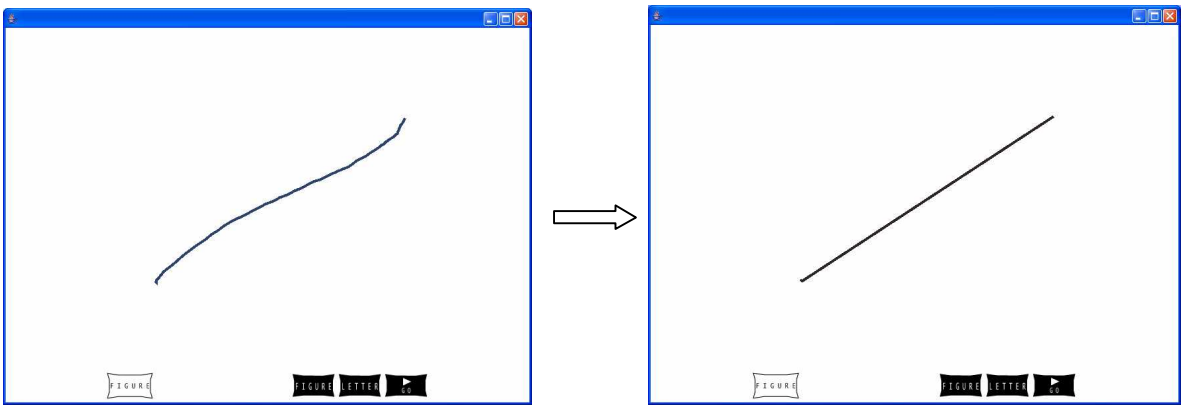


図3-6: 直線の認識・整形

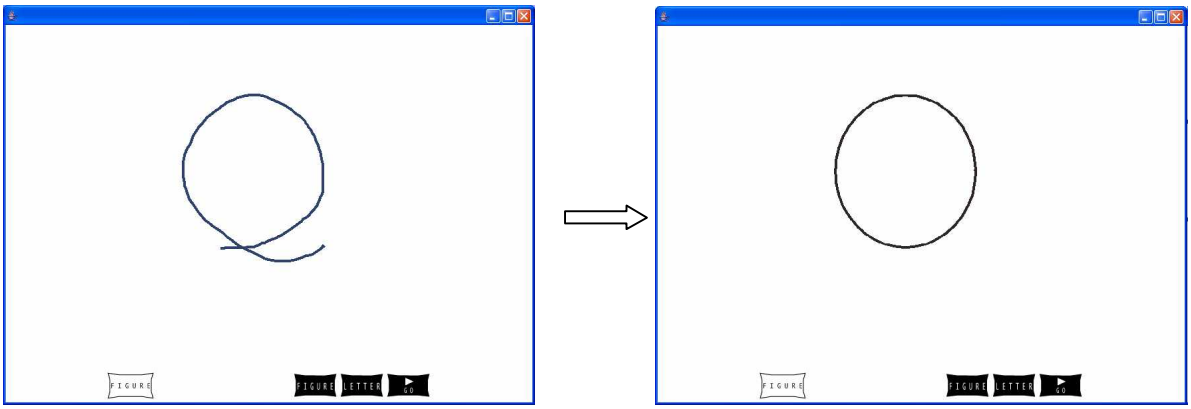


図3-7: 楕円の認識・整形

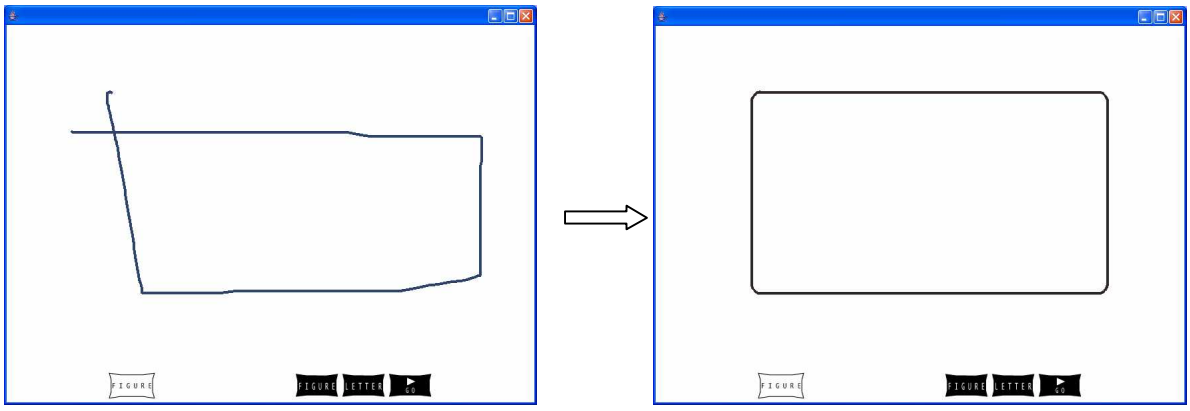
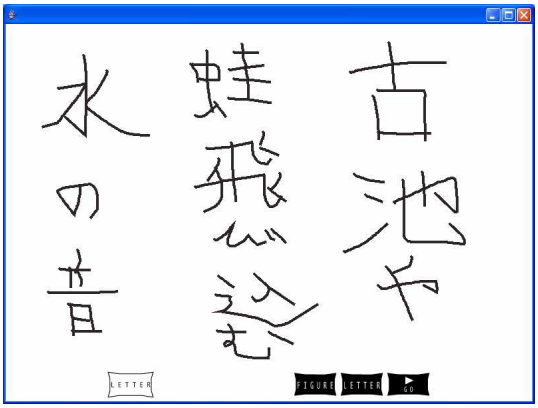


図3-8: 矩形の認識・整形

3.6 板書の再生

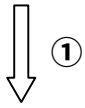
板書を書き終え、再生ボタンを押すと板書の再生が開始される。まず再生の準備として一旦

描画スペースを白紙の状態にし、その後単位時間ごとに 1 つのまとまりを描画する。図 3-9 で「古池や　蛙飛び込む　水の音」と書いた板書を例に挙げ説明する。

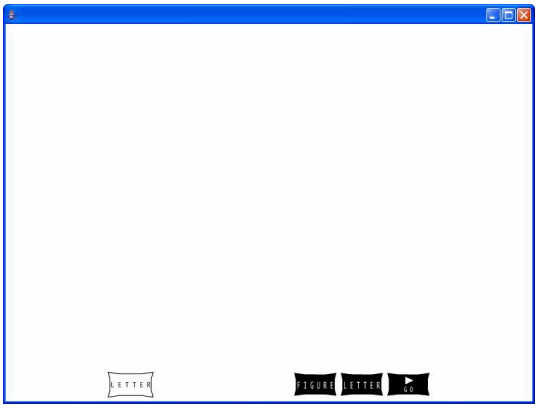


「古池や　蛙飛び込む　水の音」
と書いた板書。
ここでは
- 「古池や」
- 「蛙飛び込む」
- 「水の音」

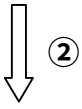
の 3 つのまとまりにまとめられたとする。



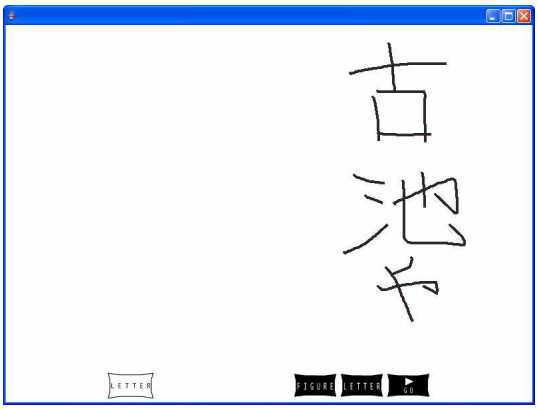
① まず全てのストロークが消去される。



全てのストロークが消去され白紙の状態となったシート。ここからまとまりごとにストロークが描画されていく。

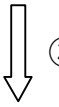


② 1 つ目のまとまりを描画する。

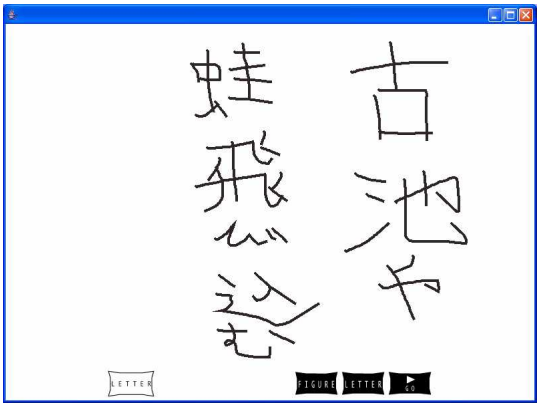


「古池や」が描画されたシート。

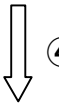
図3-9:　まとまりごとのストロークの再生



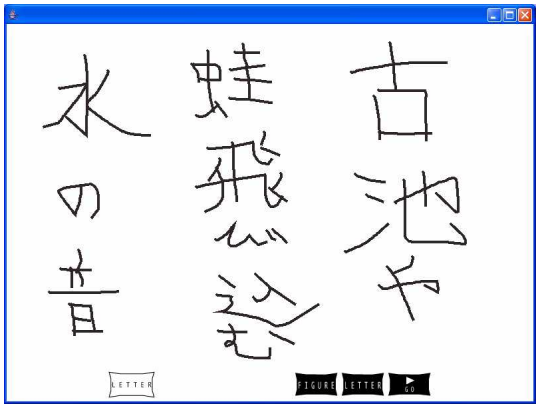
③ 2つ目のまとまりを描画する。



「蛙飛び込む」が新たに描画されたシート。



④ 3つ目のまとまりを描画する。



「水の音」が描画され、全てのストロークが描画されたシート。

図3-9: まとまりごとのストロークの再生(つづき)

第4章　実装

4.1 実装環境

システムの実装には Java2 SDK 1.4 を用いた。またストロークの処理にペンベースアプリケーション作成支援ツールキットの SATIN[2]を用いている。

4.2 主要クラスの説明

- ScribblerSheet クラス
  板書全体のデータを管理するクラスで、ストロークデータはすべてこの中で管理されている。

- Interpreter クラス
  ストロークのイベントを受け取り、実際に描画スペースへの描画を行うクラス。

- StkLib クラス
  ストロークの処理に関するメソッドが格納されているクラス。

- Counter クラス
  入力時間のブランクのカウントと、それによるまとまりづけを行うクラス。

- Player クラス
  板書の再生を行うクラス。

4.3 描画処理の流れ

ストロークが引かれペンが離れると、ストロークはその点列の多角的近似によって表現され、そのストロークイベントが Interpreter クラスに送られる。Interpreter クラスではストロークデータを受けて、フリーストロークモードならば手振れの軽減を、図形モードならば図形の認識と整形を行い最終的なストロークを描画する。また同時にストロークデータを ScribblerSheet クラス内の Vector に格納する。書かれた順に Vector に格納されることで自然とストロークに番号が割り振られる。

4.4 ストロークの自動補正

4.4.1 Bezier 近似による手振れの軽減

フリーストロークモードでストロークが引かれると、Interpreter クラス内でストロークの処理をする際に StkLib クラスの Bezier 近似を行うメソッドが呼び出され、ストロークを 3 次の Bezier 曲線で近似する。このメソッドでは点列の多角的近似で表現されているストロークの、それぞれの点列間に新たに点を追加したり点を変更することでストロークを滑らかにしている。

これにより図 3-2 のように手振れによるストロークの形状の乱れを軽減することができる。

4.4.2 図形の認識・整形

図形モードでストロークが引かれると、Interpreter クラス内でストロークの処理を行う際に StkLib クラスの図形の認識･整形を行うメソッドが呼び出され、以下の処理を行う。

1. 交点の検出

まず交点の検出を行う。矩形と楕円についてはストロークの交点を必要とするため、交点の検出によってまず直線か否かの判別を行う。

Interpreter クラスに送られたストロークイベントから点列を取り出し、その 1 つ 1 つの点列を結ぶ線分の全ての組み合わせについて以下の処理を行うことで、交点の検出を行っている。

(1) 全ての線分について、その線分を含む直線の傾きと切片を求める。
(2) ある線分と、その他の線分のうちそれに接続するものを除いた全ての線分について線分を含んだ 2 直線が交点を持つか否かを判別する。
(3) 交点が存在した場合は、さらにその交点が線分の範囲内にあるかを調べる。
(4) (3)において交点が線分上にあったならば交点が存在したとしてその座標を記憶しておく。

交点が存在しないストロークについては直線と認識し、始点と終点を結ぶことで直線を描画する。

2. 矩形と楕円の判別

矩形と楕円の判別は頂点の数をカウントすることで実現した。

当初 SATIN の direction を使用し、その変化を取り出すことで頂点を認識しようと試みた。しかし SATIN の direction は 4 方向のみの実装であったため、小さな方向の変化についても認識してしまった。

そこで、新たに図 4-1 のような 8 方向の direction を実装した。

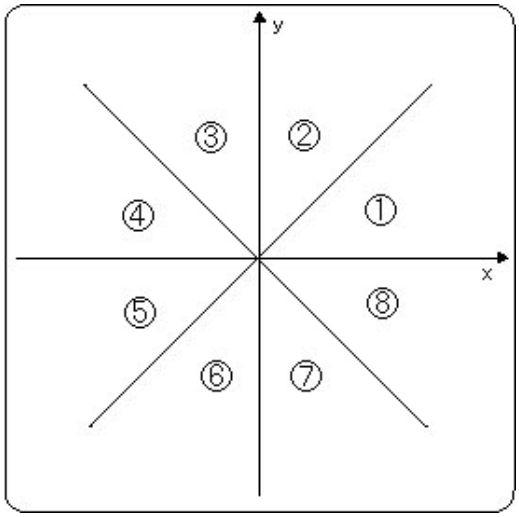


図4-1： 8 方向の direction
(軸と重なるストロークは角度の小さい側に含む)

当初はこの direction が 1 つでも変化した点は頂点として取り出すことをしていた。しかし、手書きで図形を書く際にたとえば①→②といった小さな方向の変化は頻繁に起こることがわかった。そのため方向の変化がπ/4より大きくなったときはじめて頂点と認識するようにした。つまり方向が①から②に変化しただけでは頂点とは認識されないが、①→③、①→④、…といった変化については頂点として認識するということである。この方法によりある程度曖昧な頂点については認識せず、明確なものについてのみ認識することに成功した。そしてその頂点の数を数えることで矩形と楕円の判別を行っている。

3. 描画

描画する前にまず図形の位置と大きさを算出する。具体的にはストローク上の交点で閉じられた部分の点列について、x 座標と y 座標それぞれの最大値と最小値をそれぞれ取り出す。

矩形であれば求めた x 座標と y 座標を組み合わせた4点を結ぶストロークを描画する。また楕円であれば求めた座標から中心の座標、および短径、長径の長さを算出し楕円を描画する。

4.5 ストロークのまとまりづけ

ストロークのまとまりづけを行うために、本システムではストロークの入力時間のブランクを利用している。閾値を設定し、入力時間のブランクがそれよりも長いかどうかでまとまりづけを行っている。

4.5.1 2 段階のまとまりづけ

本システムでは文字などのストローク単位のまとまりと、文などの文字単位のまとまりの2段階

のまとまりづけを行っている。当初はまとまりの段階は1つであったが、それではベースラインを揃える際に全てのストロークのベースラインが揃ってしまい、文字の形が崩れてしまう。そこで、文字などの複数のストロークから構成されているものの形を保つために閾値を 2 つ設定し、それにより文字レベルのまとまりと文レベルのまとまりの 2 段階のまとまりを認識することが可能となった。

4.5.2　時間ブランクの検出

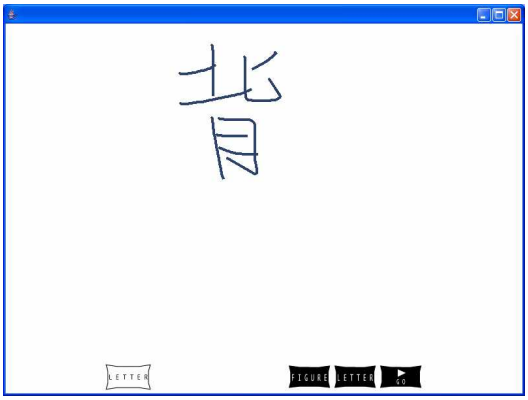
ストロークが引かれペンが離れるとそのストロークイベントが Interpreter クラスに送られ、ストロークに処理を施し描画される。それと同時に Counter クラス内の Timer が起動され、0.1 秒単位の時間のカウントを始める。次のストロークを引く mouseDown イベントが起こるとその Timer は停止され、そのストロークが描画されると初期化された Timer が動き始める。

次のストロークが引き始められないまま時間ブランクが設定した閾値を越えると、閾値を越える直前のストロークの番号を記憶する。これにより、仮想的にストロークをまとまりとして管理することができる。

4.5.3　フィードバック

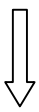
まとまりが切り替わると、ストロークの色を変えることでユーザへフィードバックを行う。描画されたストロークは、まず文字レベルのまとまりづけが行われ時間ブランクが閾値を越えまとまりが切り替わると色を変える。そして時間ブランクが文レベルの閾値を越えると最終的な色に変える。

下図で例を挙げて説明する。



ユーザはこれから「背中」と記述する。

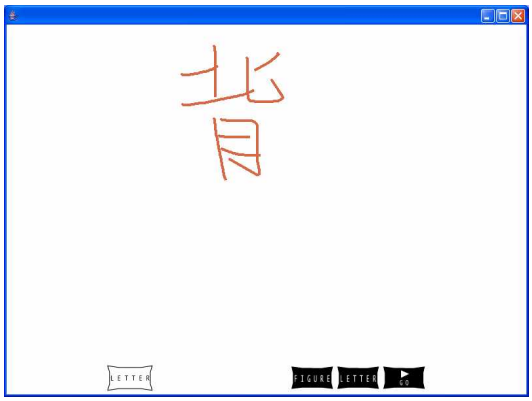
この状態で文字レベルの時間ブランクが経過する。



①

① 文字レベルの時間ブランクが経過したことで、直前のストロークが 1 つの文字の最後のストロークであると認識し、そのストロークの番号を記憶する。

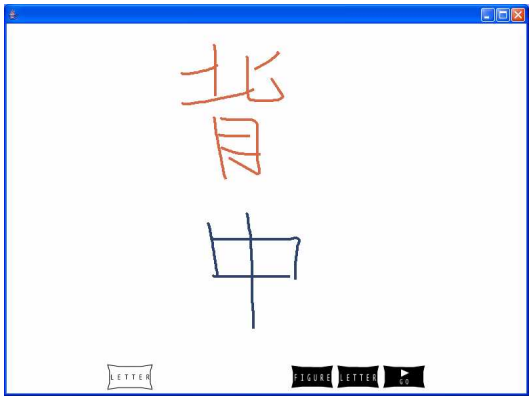
図4-2:　2 段階のまとまりづけ



それまでのストロークの色を変えることでユーザへのフィードバックを行う。

②

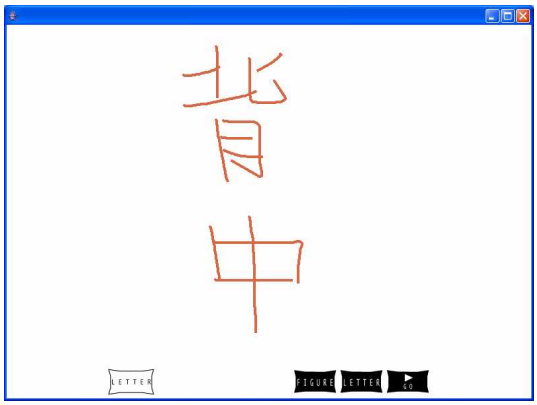
② 文レベルの時間ブランクが空く前に次のmouseDoun イベントを検出する。



この状態で再び文字レベルの時間ブランクが経過する。

③

③ 先程記憶した番号のストロークから直前のストロークまでが１つの文字であると認識し、新たに直前のストロークの番号を記憶する。

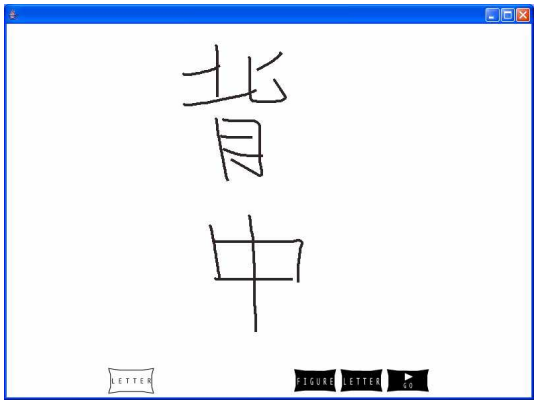


ユーザへのフィードバックを行う。
この状態で文レベルの時間ブランクが経過する。

④

④ 文レベルの時間ブランクが経過したことで、直前のストロークが１つの文の最後のストロークであると認識し、そのストロークの番号を記憶する。

図4-2: ２段階のまとまりづけ(つづき)



ストロークの色を変えることでユーザへのフィードバックを行う。

図4-2： 2 段階のまとまりづけ(つづき)

4.6 ベースラインの統一

ベースラインの統一は分類としては「ストロークの自動補正」に属するのだが、その手法として「ストロークのまとまりづけ」を利用している複合的な処理であるためここでは別個に述べる。

まず、まとまりづけを行う際にその文のベースラインの位置(縦書きであれば x 座標、横書きであれば y 座標)を決定する。現在はその文の最初の文字の位置をベースラインとして決定している。それは最初の文字の位置が最もユーザが思い描いた位置に近いと考えたからである。

文のまとまりが切り替わると、同時にまとまりの中のすべての文字に関してベースラインとの距離を算出し、算出された距離だけストロークを移動させている。この際に文字の形状(文字中のストロークの位置的な関係)を崩さないために、前もって文字単位のまとまりづけを行っている。つまり 1 つの文字の中のストロークをすべて同じ距離だけ移動させることでストロークの位置的な関係性が崩れないということである。

4.7 再生部

再生ボタンが押されると、Player クラスが呼び出される。Player クラスでは、Timer により 1 秒単位で処理が行われる。まず、再生の前段階として現在描画されているすべてのストロークを消去し描画スペースを白紙の状態にする。そしてそのうえで、各まとまりの最後のストロークとして記憶しておいた番号までのストロークを単位時間ごとに描画する。すべてのストロークを描画し終えたら Timer を停止する。

これによりまとまりごとにストロークを再生する。

第5章　関連研究

手書きストロークの整形を行うシステムとしてPegasus[3]がある。Pegasusは、手書きの直線を幾何学的制約に基づいて整形し、また次のストロークの予測をユーザに提示することで、対話的に幾何学図形を描くことができるシステムである。しかし Pegasus は正確な幾何学図形を描くことに特化したシステムであり、本研究のように自由な手書きの描画を行うものではない。

また、手書きの図形を動的に整形するシステムとして Fluid Sketches[4]がある。これは、ストロークの点列が追加されるごとに常微分方程式やバネモデルを用いて形状の認識予測をし、その予測にしたがって動的にストロークを整形するというものである。本研究ではストロークを引き終わったタイミングで整形を行っている。しかし、板書の中の特に矩形や楕円といったものは文字や文を囲む場合が多いと思われる。その場合 Fluid Sketches のようにストロークを引くと同時に整形が行われれば、思うとおりの領域が正確に囲めているかをユーザがより早く確認することができるため、今後に向けて非常に参考となるシステムである。

柔軟な発表を可能にするシステムとしてことだま[5]がある。ことだまは、キーボードに不慣れなユーザにもペンによって電子プレゼンテーションの作成を可能にし、また図形や文字の認識も行っている。またアニメーション効果の順番を可変にすることで、発表中に内容の微調整を行うことを可能にした。本研究では、従来の板書のような自由な記述を目的とし、さらに柔軟な発表を可能とするシステムを目指している。

第6章　結論

6.1 まとめ

電子ホワイトボードなどの出現によって板書を保存し再利用することが可能となった。しかし、再利用することによって板書がこれまで以上に多くの人の目に触れ、また何度も繰り返し使用されることが考えられる。そこで、本研究では板書を保存し再利用することを前提として以下の機能を持つシステムを開発した。

- リアルタイムで板書の清書を行う。
  手振れの軽減、図形の認識と整形、またベースラインの統一をすることで、板書の見にくさを解消する。

- 流れに沿って効率的に板書を再生する。
  板書を再利用する際に、最終結果だけを見せるのではなく時間軸に沿って板書を再生することでその流れも同時に再現する。しかし、ただ時間軸に沿ってストローク単位で再生するのでは効率がよくない。そこでストロークをある程度のまとまりとして管理し、そのまとまりごとに再生することで効率的に板書を再生することができる。

6.2 今後の課題

今後の課題として、いくつか挙げる。

- モードレスにする
  現段階では 2 つのモードの切り替えにボタンという方法を採用している。しかし、この方法はユーザの作業を妨げてしまうと考えられるため、モードレスにすることが必要である。現在考えているのは交点の存在するストロークについて、その閉じた領域の中に他のストロークが存在するか否かを判別する方法である。交差したストロークの領域内に他のストロークが存在した場合は図形として認識し、矩形か楕円かの判別を行い整形する。また、交点の存在するストロークの領域内にあとから他のストロークが追加された場合も、ストロークが追加されたタイミングでストロークを図形として整形し直すといった方法である。

- 位置的なまとまりづけ
  現在まとまりづけには時間情報しか利用していない。今後は時間情報だけでなく、全ての板書を書き終えたら位置的な近さによってまとまりの調整を行うことを考えている。たとえば、漢字の間違いや点の打ち忘れを指摘されあとから訂正したり点を付け足したりした場合、そこまで再現するのは効率的ではない。そこで、位置的な近さを見てまとまりを調整することでこのよう

なことが解消されると考えている。

- まとまりを利用した図形の認識･整形

本システムでは入力時間のブランクを検出することで文字と文の 2 段階でストロークのまとまりづけを行っている。現在はベースラインの統一において利用しているだけだが、これを図形モードの中で利用することで矩形を一筆書きでなくても認識することができたり、また図形同士の位置関係を補正したりすることができると考えている。たとえば、図6-1のようなストロークが同じまとまりの中で描かれた際に円の座標や半径を合わせたりするということである。

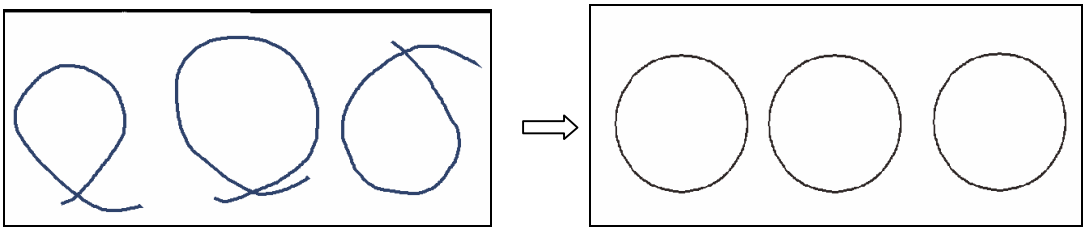


図6-1: まとまりを利用した図形の補正

- 矢印の認識･整形

現在は板書でよく用いられる図形として直線、矩形、楕円の 3 種類についての認識･整形を実現している。しかし、その他に板書でよく用いられる図形として考えられるのが矢印である。矢印の認識についてはすでに[6]や[7]でも実現している。しかし本研究では、そのどちらとも異なる時間情報を用いた認識手法によって、よりユーザへの制約を軽減した形での実現が可能ではないかと考えている。具体的には、上記の「まとまりを利用した図形の認識･整形」によって多くの場合2つのストロークによって描かれる矢印についても認識･整形が可能になると考えている。また、整形された矢印をドラッグすることで矢印が滑らかに伸びたり縮んだり、[8]や[9]のように矢印で結んだ図形の関係を認識し、移動の際にはその関係性を崩さずに移動するといったことも追加機能として考えている。

謝辞

本論文を執筆するにあたり、指導教員である田中二郎先生をはじめ、三末和男先生、志築文太郎先生、高橋伸先生には研究全般にわたり様々な助言やご指導をいただきました。深く感謝を申し上げます。また田中研究室の皆様にも大変貴重な意見をいただきました。特に NAIS チームのメンバーの皆様には研究全般にわたり大変お世話になりました。本当にありがとうございました。

参考文献

[1] Elin Ronby Pedersen, Kim McCall, Thomas P. Moran, and Frank G. Halasz. Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. Conference of Human Factors in Computing Systems, pp.391-398, 1993.

[2] Jason I. Hong and James A. Landay. SATIN: A Toolkit for Informal Ink-based Applications. UIST 2000, pp.63-72, 2000.

[3] Takeo Igarashi, Sachiko Kawachiya, Hidehiko Tanaka, and Satoshi Matsuoka. Pegassus: A Drawing System for Rapid Geometric Design. Conference of Human Factors in Computing Systems, pp.24-25, 1998.

[4] James Arvo and Kevin Novins. Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes. UIST 2000, pp.73-80, 2000.

[5] 栗原 一貴, 五十嵐 健夫, 伊東 乾. ことだま：ペンベース電子プレゼンテーションの提案. 第12 回 インタラクティブシステムとソフトウェアに関するワークショップ(WISS2004), pp.77-82, 2004.

[6] Christine Alvarado and Randall Davis. SketchREAD: A Multi-Domain Sketch Recognition Engine. UIST 2004, pp.23-32, 2004.

[7] Levent Burk Kara and Thomas F. Stahovich. Hierarchical Parsing and Recognition of Hand-Sketched Diagrams. UIST 2004, pp.13-22, 2004.

[8] Kazuo Misue and Jiro Tanaka. A Handwriting Tool to Support Create Activities. Proceedings of 9th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, pp.423-429, 2005.

[9] James Arvo and Kevin Novins. Appearance-Preserving Manipulation of Hand-Drawn Graphs. Computer graphics and interactive techniques in Australasia and South East Asia, pp.61-68, 2005.