作成手順を考慮した切り絵制作支援ツール

システム情報工学研究科　コンピュータサイエンス専攻
博士前期課程 1 年　201120729　中島 健次郎
指導教員　三谷 純
2011 年 11 月 24 日

1. はじめに

「切り絵」とは白黒に塗り分けた下絵を黒い紙に固定し、デザインカッター等を用いて不要部分を切り抜いて作り上げる絵画手法の一つである(図 1)。と寄席で見るような紙を折り曲げハサミで切り抜く「紙切り」と異なる。

基本的な「切り絵」は黒い紙を切り抜き、白い紙に貼りつけて制作され、白黒のモノクロで表現される。切り抜く領域を明示的にするために白黒に塗り分けた下絵を準備し、これをホチキスなどで黒い紙に固定し、下絵の白い領域を切り抜いていく。下絵の黒い領域は一つに繋がっていないといけないという制約が存在するため、素人がオリジナルな切り絵をデザインするためには経験と知識が必要となる。そのため、大抵の切り絵初心者が切り絵を始める際には、出来合いの簡単な切り絵サンプルを下絵にして切ることになる。また、綺麗な仕上がりを得るためには、切り順も重要な要素であるが（詳細は 6.1 節で述べる）、適切な順序を見つけるには経験が求められる。

本研究では、切り絵制作に必要な下絵の作成と、切り抜きを行う輪郭線の抽出、および適切な切り順の提示を行うシステムを提案する。

図 1：左）切り絵作品、右）紙切り作品
（著者作成による）

2. 関連研究

Xu らは輝度値を含む画像、主として写真から切り絵の下絵を作り出す手法を提案した[1]。この手法はまず、画像中の任意の部分領域を選択ツール[2][3]を用いて選択し、その部分領域内で閾値による二値化を行う。この部分的な二値化の繰り返しで、ユーザが望む形で画像を白の領域と黒の領域に分ける。白と黒に分けられた画像の中で、面積が最小の黒領域と他の黒領域が繋がるまで元の画像の輝度値を考慮しながら領域を自動変形する。全体として黒い領域が一繋がりとなるまで、この処理を繰り返す。この手法は主として写真画像を対象としているが、本研究では手書きのスケッチ画を対象とする。

Igarashi らはステンシルのデザインのためのドローエディタを提案した[4]。ステンシルとは穴の開いた型板を用いて上から図柄を描く絵画技法であり、型番は一枚の繋がった板からなる。図柄の形状の制約は切り絵と同じである。Igarashi らのドローエディタではユーザが切り抜く領域を自由に描くことができる。このとき、切り抜く領域の周りの輪郭線に沿って一定幅が切り抜かれない領域として残され、それらが重なり合わせることで、結果として一繋がりの図形を得る。ユーザが自由にデザインした図形をベースとするが、切り絵の基本的な表現を考慮した下絵作成のアプローチとは異なる(詳細は 4.1 で述べる)。

3. 提案手法の流れ

提案手法では図 2 に示すような、実際の切り絵制作における手順を考慮した流れで、望むデザインの切り絵の制作を支援する。

実際の切り絵では、まず製作者はデッサンをして、望む切り絵の全体の構成概要を形にする。このデッサンを白黒に染め分け、切り抜くための下絵を作る。この際、黒い領域は一繋がりにする。次に下絵を黒い紙に固定し、白黒の境界線に沿って切り抜く。切り抜かれた黒い紙だけを白い台紙に貼り付ける。

以上を踏まえた上で、本システムでは、まずユーザにはモチーフをスケッチしてもらう。そのスケッチ画像を入力として受け取り、スケッチ画像から対話的に二値化し、一つながりの黒い領域からなる下絵を構築する。この下絵から輪郭線を抽出し、輪郭線を複数の線分に分解して、適切な切り順を算出する。最終的に、輪郭線の切り順が示された輪郭線の画像を出力する。ユーザは出力された画像をホチキスなどで、黒い紙に固定し切り順に従って切り抜きを行う。

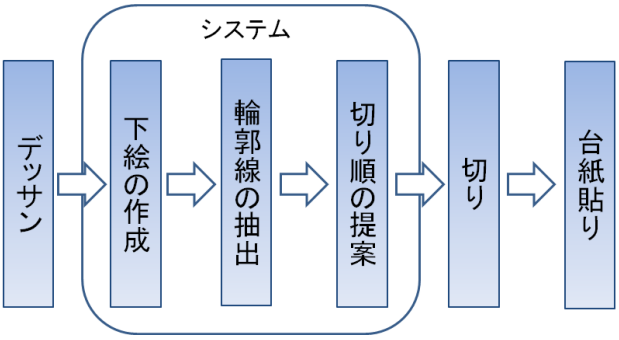


図 2：制作手順とシステムの流れ

4. 下絵の提案

デッサンを入力として、切り絵の基本的な表現を考慮した下絵作成のアプローチを元に下絵を提案する。

4.1. 切り絵の基本的な表現

切り絵の基本的な表現は、線によって分けられた閉領域を白(切り抜く領域)と黒(切り抜かず残す領域)のどちらに割り当てるかによって決まってくる。隣り合う領域を片方が白、片方が黒と割り当てたならその境界線をそのまま切る線とすればいい。隣り合う領域を両方とも白と割り当てた場合はその境界線を一定の幅で切り抜かない黒い領域として残すのが基本的な表現となっている(図 3 左)。逆に、隣り合う領域を両方とも黒と割り当てた場合はその境界線を一定の幅で切り抜き白い領域としたいが、閉領域の境界線をそのまま切り抜いたうえで、その中の領域を黒として残すことはできない。不自然でない様に境界線を部分的にだけ切り抜き、隣り合う黒い領域を繋げる必要がある。基本的に繋げる個所となるのは線の交差点である(図 3 右)。これらの基本表現が混ざり合い全体として切り絵の作品となる。

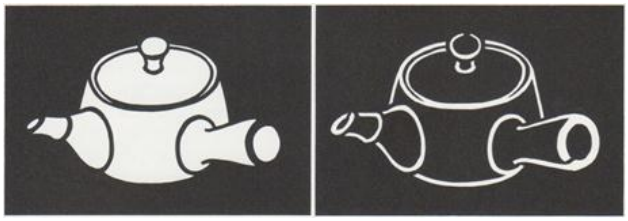


図 3：切り絵の基本的表現(きりえ全科[5]より)

4.2. 提案手法

入力をスケッチ画像として、切り絵の基本的表現を考慮した流れで下絵を出力する。そのために、まずスケッチ画から境界線を抽出し、入力画像を閉領域で分割する必要がある。領域分割後、領域内に含まれるストロークや輝度値の情報から領域ごとの割り当てを推測する。その結果、孤立してしまう領域が出てきたのなら、線の交差点で繋げるなどの基本的な手法によりできる限り一つながりになるように処理を行う。それでも、孤立してしまう領域が出てきたのなら、ユーザと対話的に一つながりの下絵の提案を実現する。

デッサンによって作られるスケッチ画像では物体の輪郭線は一本とは限らず複数のストロークによって表現される。複数のストロークをまとめ、境界線を抽出するために Laplacian of Gaussian フィルタを用いて境界線を抽出している。しかし、スケッチでは、ストロークで陰影と色の濃さも表現されることがあるため、画像によってはうまく輪郭線のみを抜き出せていないのが現状である。

5. 輪郭線の抽出

入力はすでに完成された下絵として、切るべき輪郭線を抽出する。下絵は 2 値化されているラスタ形式の画像であると仮定して、一般的な 8 近傍の輪郭抽出アルゴリズムで境界を求める。輪郭のピクセルの中心を頂点として隣接する頂点同士で線を引き、連結情報を残しておく。このままでは線の長さが短く、数が多すぎ、なめらかな折れ線とは言えないので平滑化を行う。平滑化には Laplacian smoothing を用いる。平滑化された後、ある頂点に隣接する 2 つの線分が平行に近い場合この頂点は不要とみなし削除する。そして、2 つの線分を 1 つの線分にして、線の数を減らす。この平滑化と線の削除を繰り返す。

6. 切り順の提案

輪郭線を切る際に、切る手順が適切でないと、切る対象としている紙が部分的にずれて、最終的な質に影響を与えることがある。そのため適切な切り順を見つけることが重要となる。

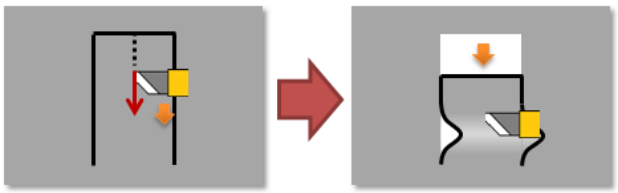
6.1. 「ずれる・ずれない」の判定法

輪郭線の一部に対して切る操作を行う際、紙が部分的にずれる場合と、ずれないで切れる場合がある。どのような時に「ずれる」かを判定する方法を以下で提案する。

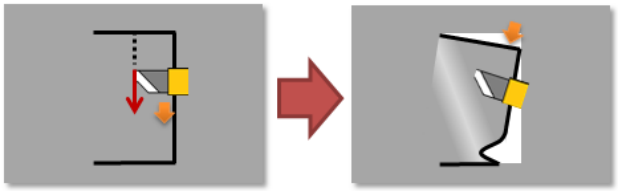
例として、図 4 に示す 5 つの状況を考える。前提として、紙の端はホッチキスやテープなどで固定されていて、紙全体がずれることが無いものとする。

(a),(b),(c)は切り始める点(以降、始点)が、すでに切られている線で 3 方向が囲まれているが、(a),(b)はずれてしまい、(c)のみ、ずれずに切れる。

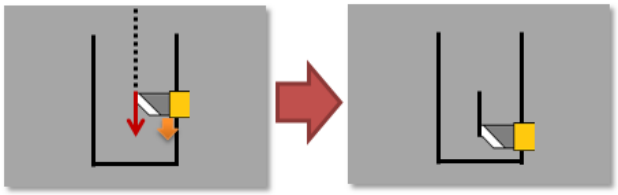
紙を切る際、カッターが進む方向に紙を引っ張る力が紙の張力に反発して釣り合うことで切ることができる。切る線が直線であると仮定し、切り終える点を終点とすると、紙を切ろうと引っ張る力が働くのは、終点を端点して始点を通る半直線(以降、張線)である。この張線が既に切られている線(以降、既切線とする)と交差する場合、紙は切れることなく部分的にずれるということになる。この推測から、状況(a),(b),(c)の違いが説明できる。



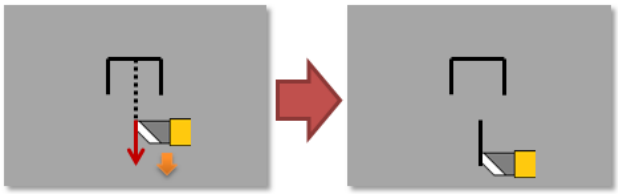
(a) ずれる



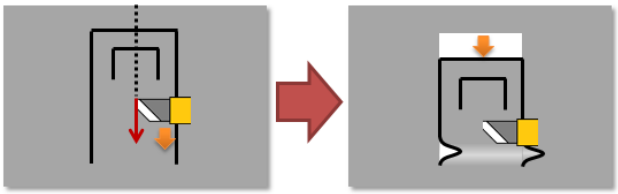
(b) ずれる



(c) ずれずない



(d) ずれずない



(e) ずれる

図 4：ずれる状況とずれない状況

なお、(a)では、ずれる方向が切る方向と同じなのに対して、(b)では、ずれる方向が切ろうとした方向と角度が異なる。これは、切ろうとして引っ張る力に対して一部の方向だけに張力が働いたためと考えられる。既切線の左側の端点と始点を結ぶ直線の方向に張力が働き、その直線方向に分解できる力は釣り合うが、残りの分解された力は釣り合うことなく、結果としてその方向にずれると考えられる。これを踏まえると、(d)では、既切線の左側の端点によっては釣り合わなかった力が右側の端点によって釣りう。こうして(d)では、紙の張力が働き、切ることができていると考えられる。

以上のことから、始点が張線と交わる既切線によって囲まれているか否かによって、部分的にずれるかどうか判定できると考えられる。ここでいう「点が線に囲まれている」という表現を、「線で作られる凸包に点が含まれる」と定義する。

(e)は、考慮しなければいけない凸包は張線と交わる全てのカットされた線の凸包であることを示している。

以上を踏まえ、ずれてしまう切り方かどうかは、アルゴリズム 1 に示す擬似コードで判断できる。

```
PROCEDURE ずれの判定(切り込みの始点, 切り込みの終点)
  ray ←「切り込みの終点」を端点として「切り込みの始点」を通る半直線
  FOR 既切線
    IF ray が既切線を横切る AND
    既切線の凸包に切り込みの始点が含まれる
    THEN
      RETURN ずれる
    END IF
  END FOR
  RETURN ずれない
END PROCEDURE
```

アルゴリズム 1：ずれの判定

6.2. 切り順の提案手法

目的である切り順の提案を行うために、どの線をどの方向から切るか、という情報をリスト構造で保持し、切り順を表現する。どこも切っていない状態から、次の線を一つ一つ切り、ずれる切り方なら、別の切り方を調べる。それを繰り返し、終始ずれないように切りを終えたのならその切り順を出力する。

現在のシステムでは、深さ優先探索を行うものとした(図 5)。

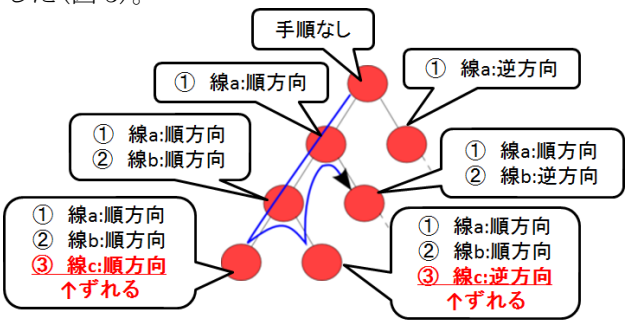


図 5：探索のイメージ

7. 実装

提案するシステムは Java を用いて実装した。初期状態として与えられた複数の線分に対してどのような順番で切ると、ずれることなく切れるかシミュレーションできるようになっている。ユーザが切るべき線にカーソルを合わせクリックすることで、その線を切ることができる。このとき、両端点のうちカーソルの位置が近い点が始点、遠い点が終点となる。切ろうとしている線の張線と既切線が交差するなら、その既切線で作られる凸包を全て表示する。この凸包に始点が含まれているなら、クリックしても切ることはできない(図 6)。

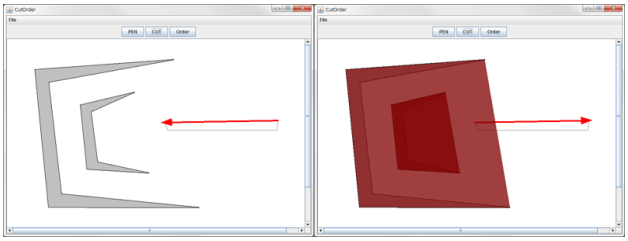


図 6：切る方向と関わる凸包の表示

さらに、切り順の提案も可能で、探索によって解が得られた場合は、線に切るべき順の番号を表示する。その順に従って正しく切れることを確認できる(図 7)。

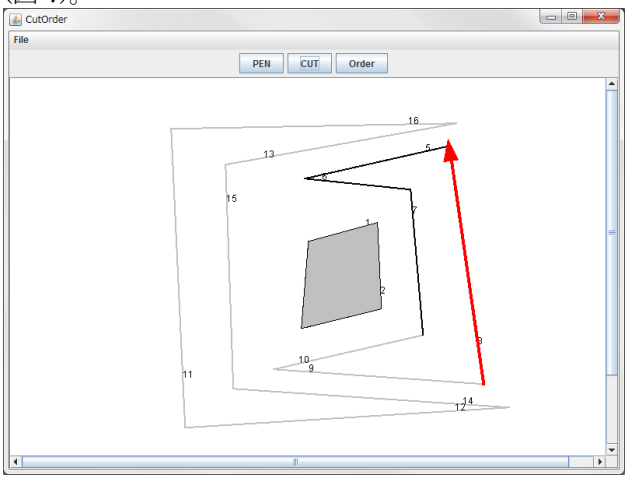


図 7：切り順の表示

8. まとめと今後の課題

本研究では、切り絵の完成までの手順を考慮した上での支援ツールを提案すること目的としている。今回はその切り絵の完成までに支障となる切り方について考察し、ずれの発生の有無を判定するアルゴリズムを考案した。その上で、適切な切り順を探索で求めることを行った。

探索方法として深さ優先探索を用いたが、この手法では線の数が増えると時間がかかりすぎる問題点がある。現在は、簡単な例でしか満足いく時間内では切り順が提案できていない。これを解決するには個々の切り順を評価するヒューリスティック関数を用いて、切り順が目標のずれずに切り終える手順までどのくらい妥当そうかを評価し、探索に組み込む必要がある。

また、実際の切り絵経験者はずれないように推測しつつ効率よく切ることを考えて切っている。例えば、距離が近い線でほぼ同じ方向の線が複数あったのなら、その線群を先にまとめて切ったりする。現在の出力される切り順ではずれずに切れることしか考えていないため、作業効率は考えられていない。作業効率を上げる考え方を探索に入れることも今後の課題である。具体的には個々の切り順を評価するコスト関数を用いて、どれほど効率のよい切り方になっているかを評価し、探索に組み込む必要がある。切り絵経験者の観点も組み込むつもりである。

探索アルゴリズムには A*探索を用いることを今後の課題としている。探索で得られた切り順はカッティングプロッタへの応用も期待される。カッティングプロッタにおいても、紙全体をうまく固定できない場合にずれは起こりうる問題であり、かつ、切り絵の面から見た効率の良い切り順はカッティングプロッタにおける加工時間の短縮につながる可能性がある。

そのほか、現状では下絵の提案において解決すべき多くの点が残されている。境界線の抽出においては 4 章で述べた問題がある。さらに、孤立領域を繋げる手法において、線の交差点で繋ぐだけではユーザの望む表現とならない例も多く存在する。望む下絵を得るにはスムーズな対話的処理が必要である。

また、切るべき線の数を減らすことも課題である。そのために輪郭抽出を行った後の平滑化と線の削減作業を繰り返せばいいが、目的の形状からずれが生じてしまうと考えられる。探索時間を考えた線の数と目的の形状を維持するために必要な線の数をうまく設定するのも課題となる。

最後に、提案された切り順の評価する必要がある。現在切り絵素人の方数名に対してのユーザテストを考えている。簡単なサンプルではなく多少複雑な切り絵のサンプルを用意して、切り順を提案した場合と、しない場合どちらが切りやすかったかを判断してもらうことを想定している。切り絵経験者を集め、経験者にとっても提案する切り順は有効だったかどうかも評価したい。

参考文献

[1] J. Xu, C. S. Kaplan, and X. Mi. Computer-generated papercutting. In Proc. PG '07, pages 343–350, 2007.

[2] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. ACM Trans. Graph., 23(3):303–308, 2004.

[3] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. pages 191–198. ACM Press, 1995.

[4] Igarashi, Y. and Igarashi, T. Holly: A Drawing Editor for Designing Stencils. Computer Graphics and Applications, IEEE, pages 8-14, 2010.

[5] 日本きりえ協会(1997)『きりえ全科』誠文堂新光社 205pp.