Topology-Driven Vectorization of Clean Line Drawings

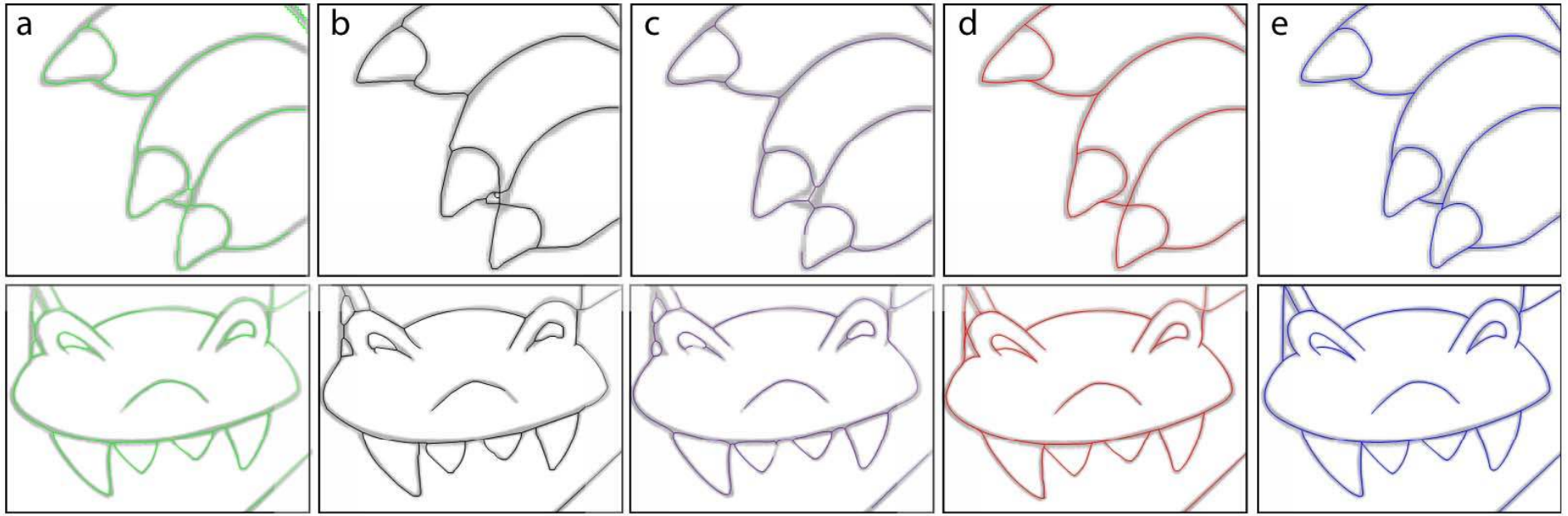
Gioacchino Noris1,2
Alexander Hornung2
Robert W. Sumner2
Maryann Simmons3
Markus Gross1,2

1ETH Zurich
2Disney Research Zurich
3Walt Disney Animation Studios

モチベーション

線画のベクトル化

- Figure 19（論文の図を拡大して観察）
  - 上段は爪と爪の間のジャンクション
  - 下段は目の形



線画のベクトル化が難しい理由

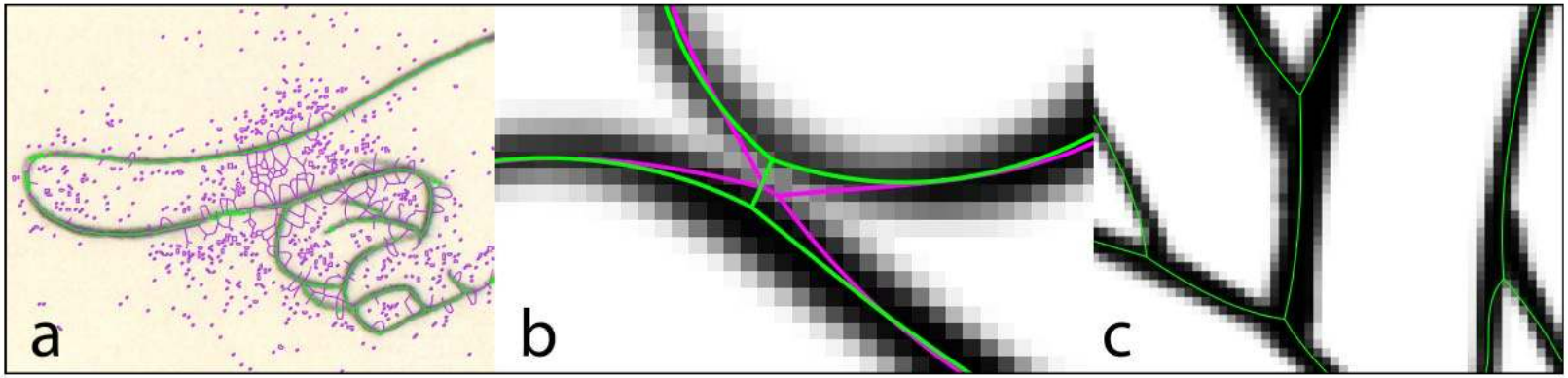


Figure 1: *Vectorization Challenges. Noise (a) and spatially adjacent strokes (b) require fine tuning of threshold parameters in existing approaches. Even for clean, high resolution input images, the superposition of strokes near junctions and sharp corners results in inaccurate centerline placement (c). (Results of Adobe Live Trace for different threshold settings shown in purple and green.)*

線画のベクトル化が難しい理由

- 今までの手法は局所的な処理のみ
  - aとbの区別がつかなかった
- 提案手法は位相構造が確定するまで円を拡張

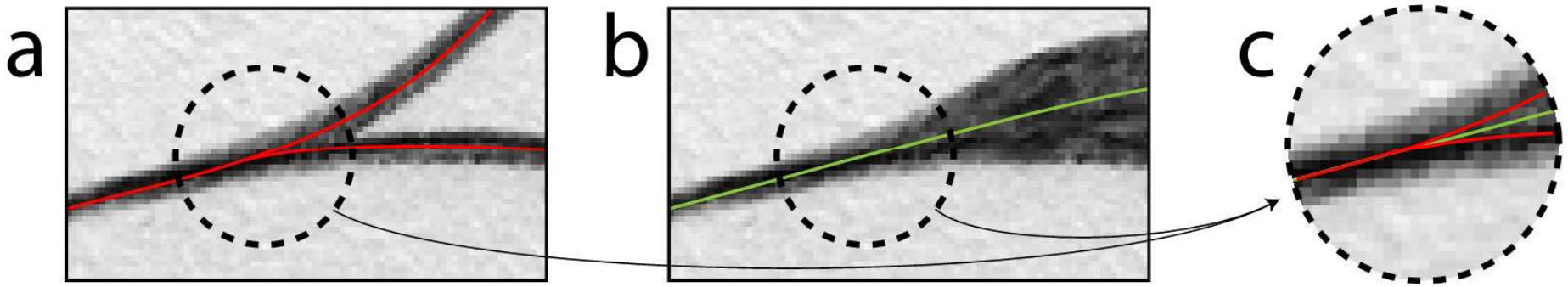
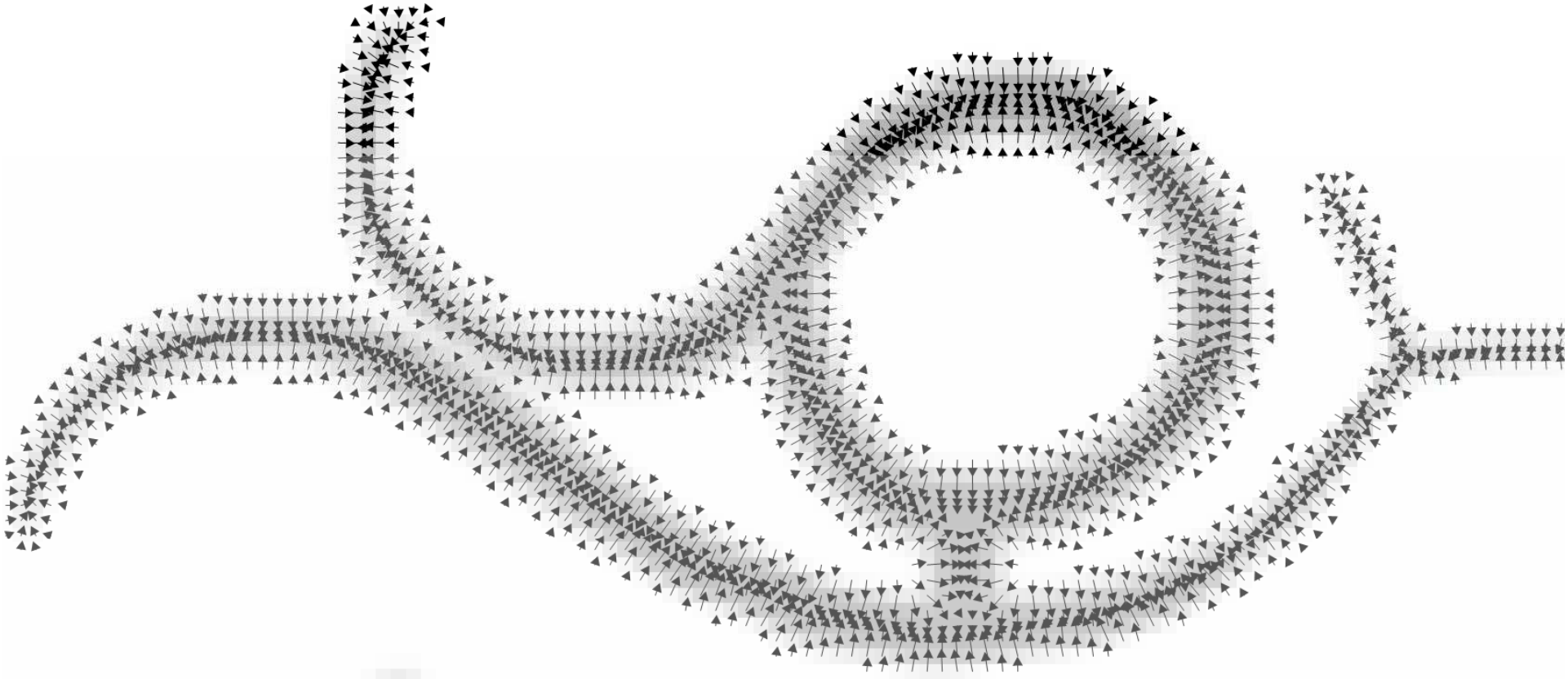


Figure 5: *Local Ambiguity. A junction (a) and a stroke with varying thickness (b) cannot be distinguished by considering the local appearance only (c).*

提案手法の概要

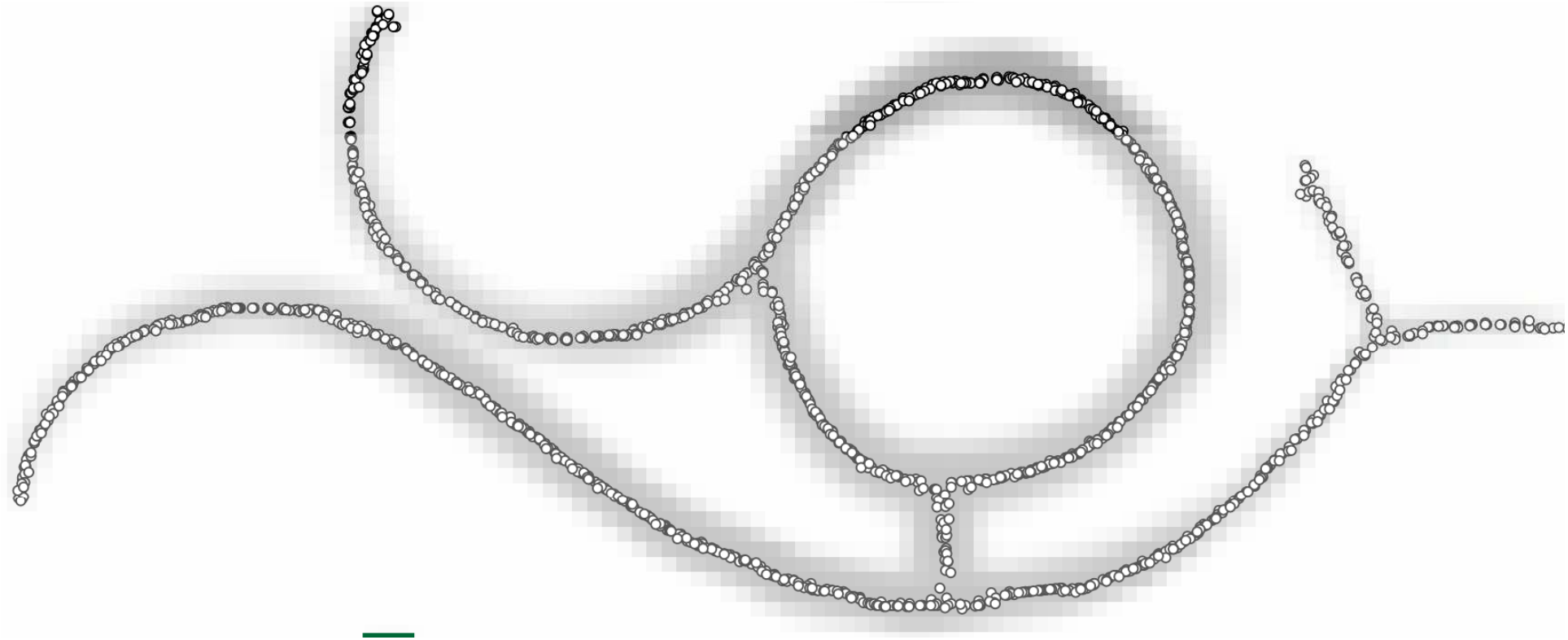
提案手法の概要

- エッジ上のピクセルをパーティクルとし、
  画像勾配の方向へ少しずつ、繰り返し動かす
  - エッジの中央へパーティクルが集まってくる



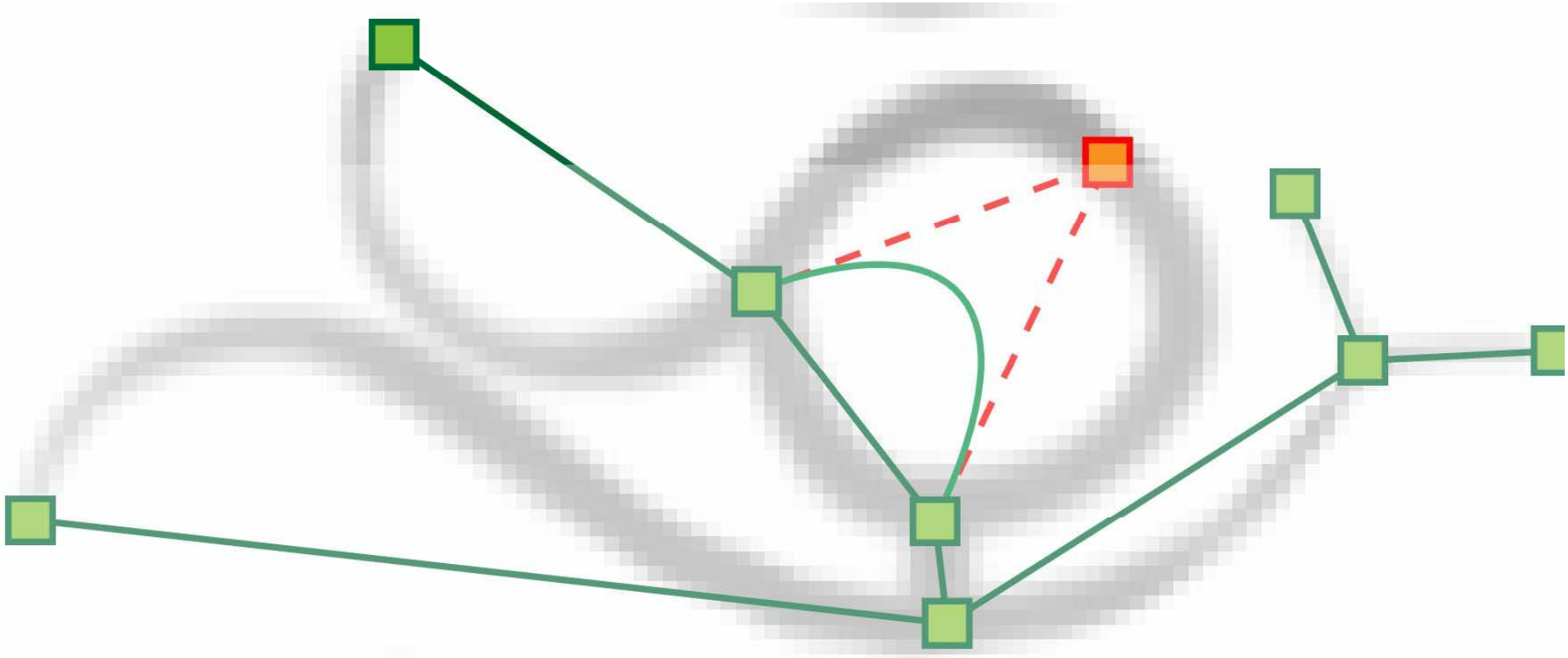
提案手法の概要

- 近傍のパーティクル同士を繋いでグラフを作る
- 無駄な接続を削除するため最小全域木を計算



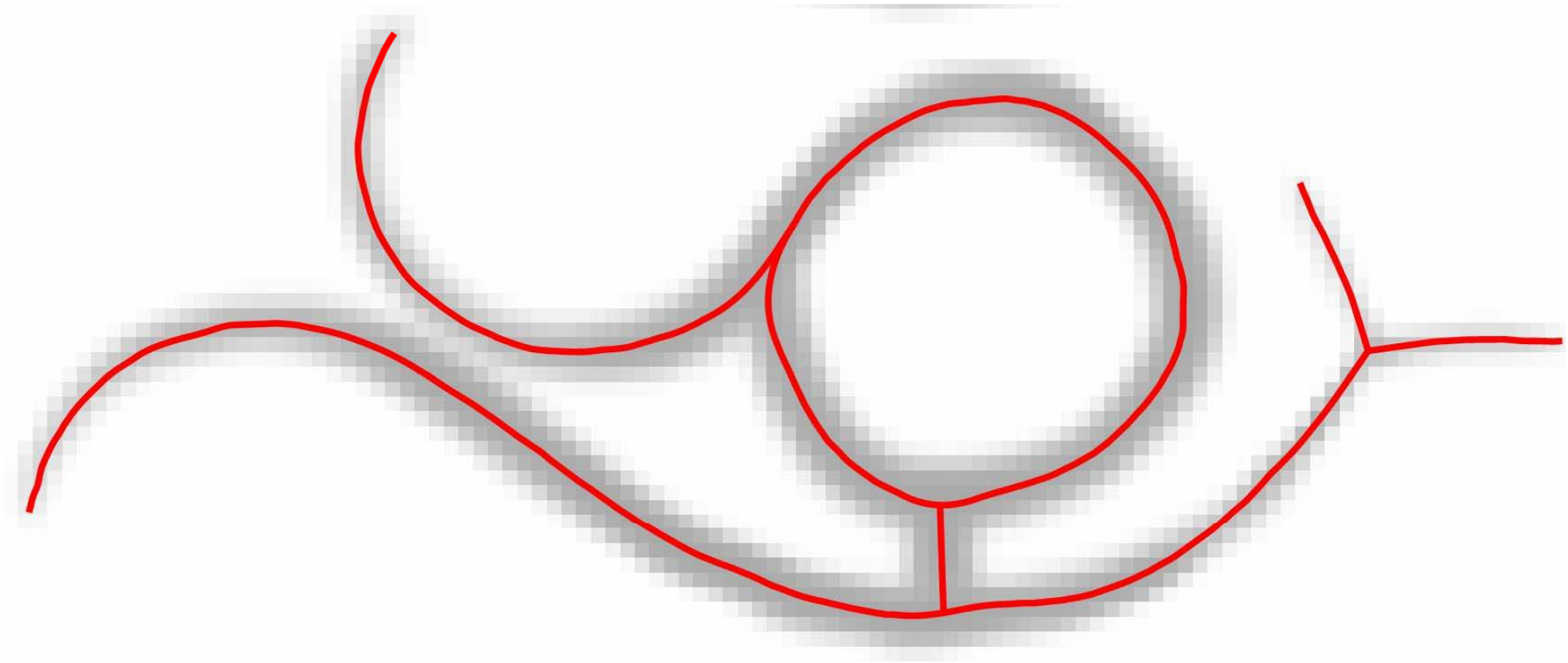
提案手法の概要

- 接続数が2以外のパーティクルに関して、位相構造を観察しつつ、形を整える



提案手法の概要

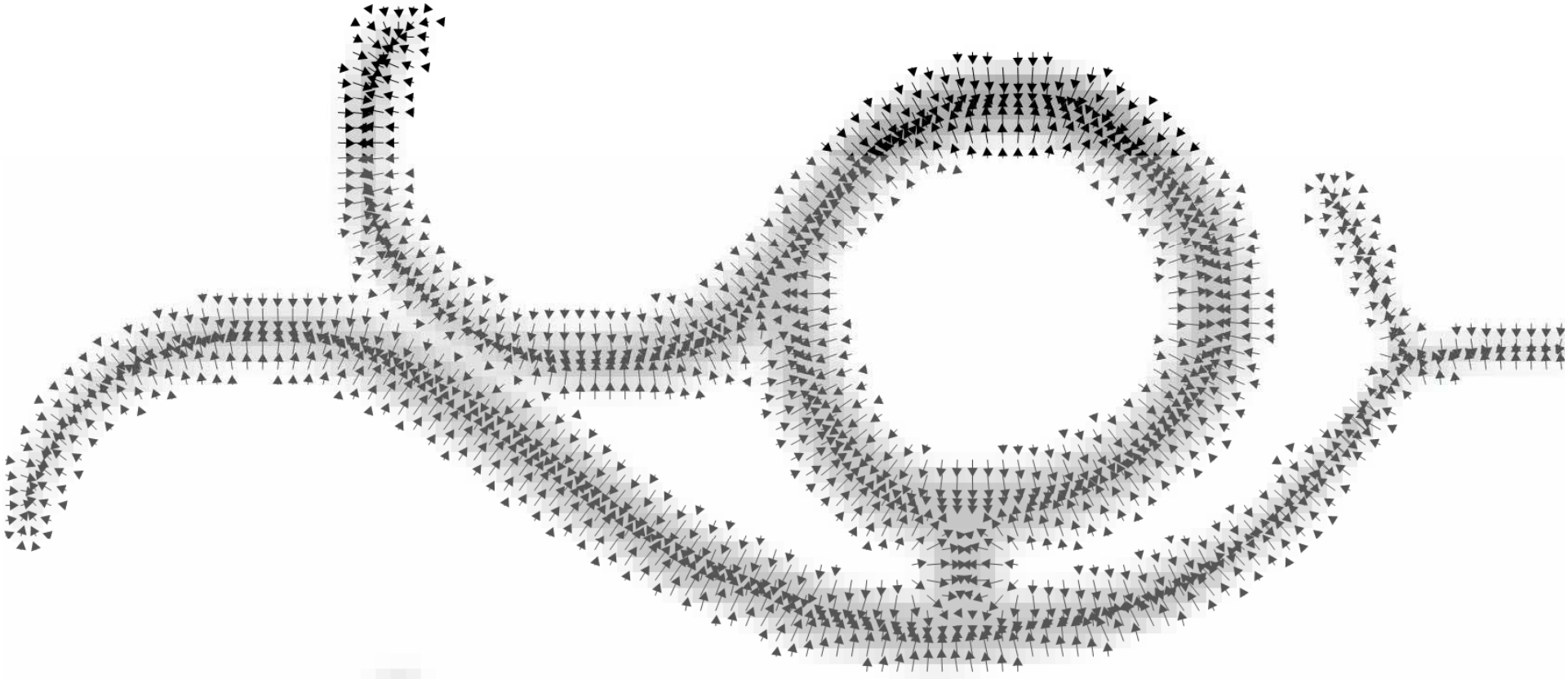
- 綺麗なベクトル化が完成



アルゴリズムの詳細

各パーティクルの移動

- 画像をIとする時、勾配は(dI/dx, dI/dy)
  - 下図の各矢印で表現する
- X’ = X + (dI/dx, dI/dy)*α　α=0.1　・・・で更新



各パーティクルの移動

- Jpeg圧縮等のノイズを閾値（ε）で削除
- 各パーティクルは、自分の進行方向と反対方向の勾配に出会ったら停止する

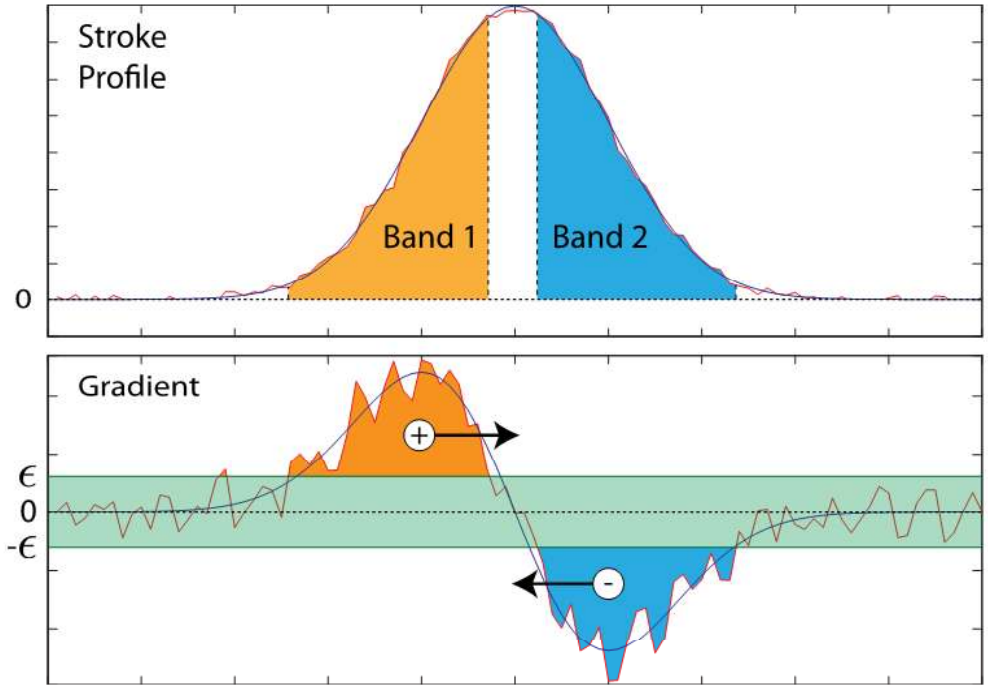


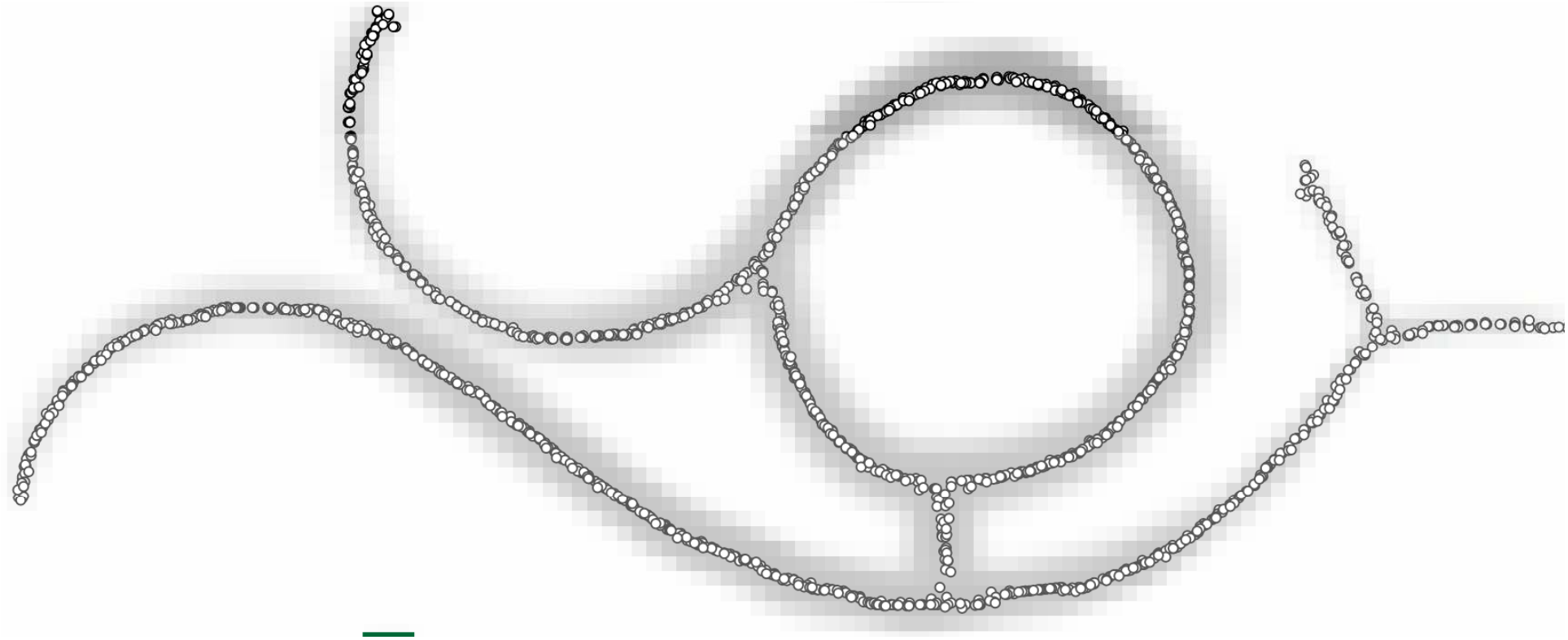
Figure 3: *The gradient threshold ϵ defines two bands of pixels with opposite gradient directions.*

各パーティクルの移動

```
% x- and y- derivatives as the moving direction of each particle
index = sub2ind(size(M), y, x);
DY = Ig - circshift(Ig, [1 0]);
DX = Ig - circshift(Ig, [0 1]);
dy = double(DY(index));
dx = double(DX(index));
mag = sqrt(dx .* dx + dy .* dy) + 1e-10;
dy = dy ./ mag;
dx = dx ./ mag;

% Move each particle
subplot(3, 3, 4); plot(x, -y, '.'); pause(0.02);
for i = 1:40
    x = max(1, min(x - dx * 0.1, size(M, 2)));
    y = max(1, min(y - dy * 0.1, size(M, 1)));
    index = sub2ind(size(M), floor(y), floor(x));
    d = double(DY(index)) .* dy + double(DX(index)) .* dx;
    dy = dy .* (0 < d);
    dx = dx .* (0 < d);
    subplot(3, 3, 4); plot(x, -y, '.'); pause(0.02);
end
```

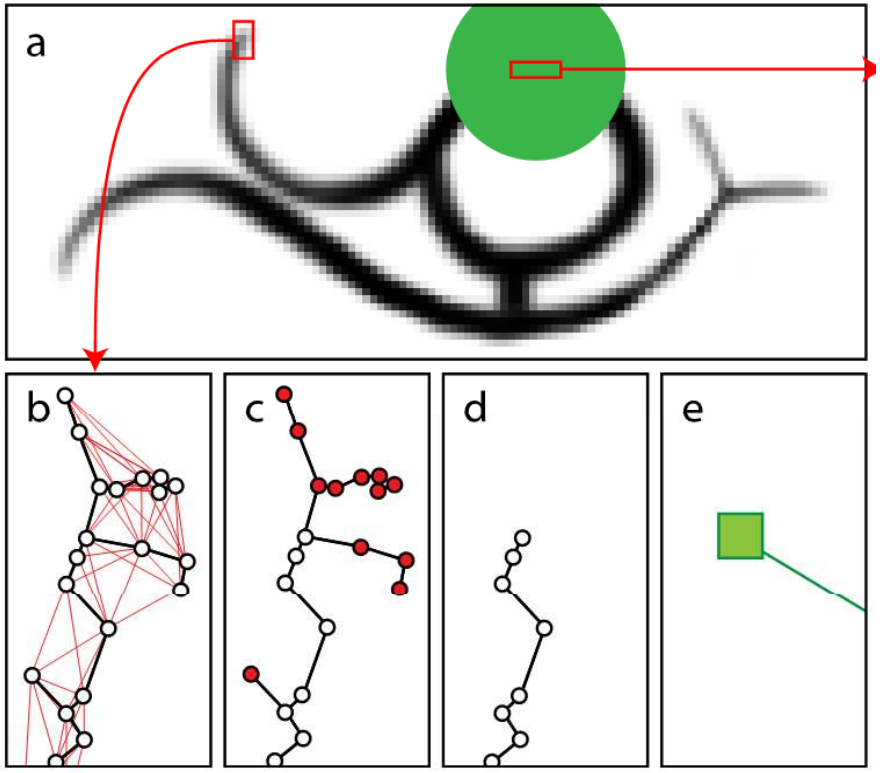
グラフの作成と無駄な接続の除去

- 近傍のパーティクル同士を繋いでグラフを作る
- 無駄な接続を削除するため最小全域木を計算



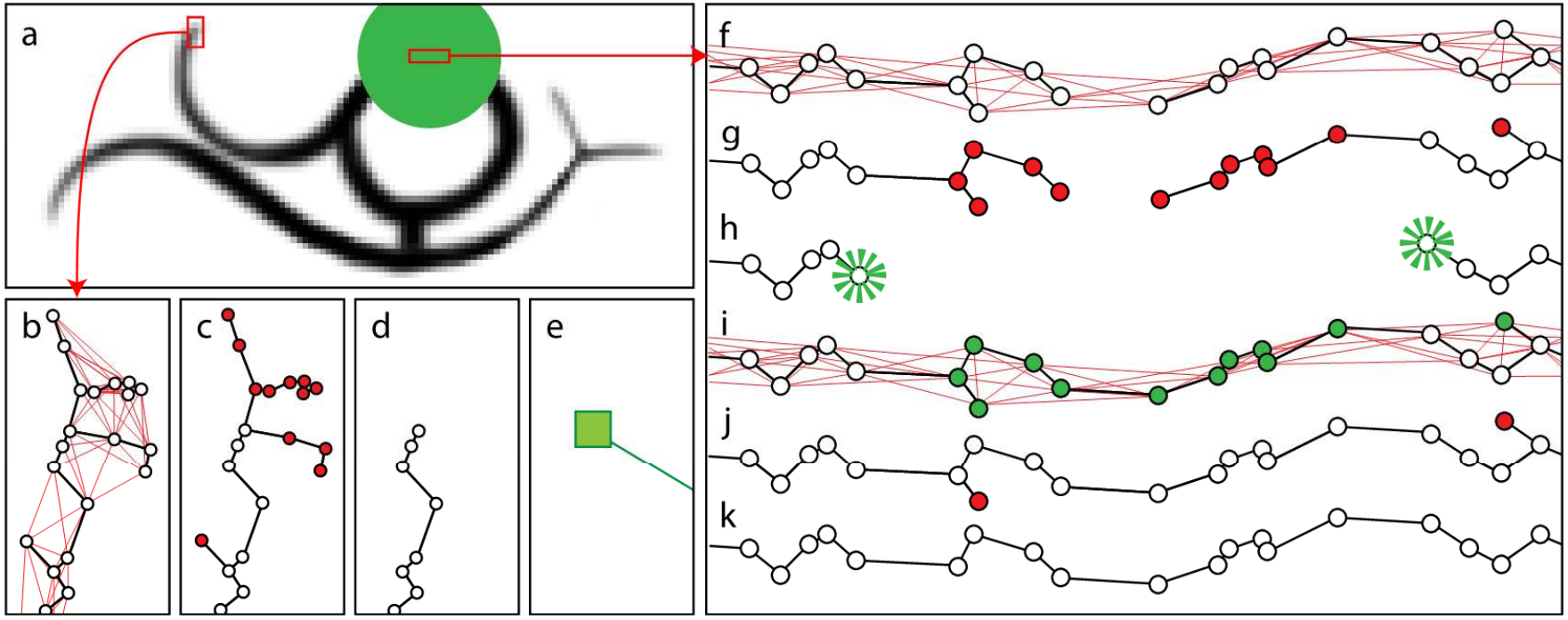
グラフの作成と無駄な接続の除去

- b)　各パーティクルについて
  - ある距離範囲にある他のパーティクルとを繋ぐ
- c)　最小全域木を計算する
  - 接続数をなるべく少なく、全体をカバーする



グラフの作成と無駄な接続の除去

- d)　葉のノードをある距離まで削除
  - 更に無駄な分岐を削除する



グラフの作成と無駄な接続の除去

```
% Make the graph
X = ones(size(x, 1), size(x, 1)) * diag(x);
dX = (X' - X);
Y = ones(size(y, 1), size(y, 1)) * diag(y);
dY = (Y' - Y);
D = dX .* dX + dY .* dY + eye(size(x, 1)) * 1e+10; % Distance matrix
AG = sparse(D .* (D < radius^2)); % Adjacency matrix
[R, C, V] = find(AG);
subplot(3, 3, 5); %line([x(C), x(R)], [-y(C), -y(R)]);
for i = 1:size(R, 1) % This rendering is slow...
    if R(i) < C(i)
        line([x(R(i)), x(C(i))], [-y(R(i)), -y(C(i))]);
    end
end
MST = graphminspantree(AG);
[R, C, V] = find(MST);
subplot(3, 3, 6); %line([x(C), x(R)], [-y(C), -y(R)]);
for i = 1:size(R, 1) % This rendering is slow...
    line([x(R(i)), x(C(i))], [-y(R(i)), -y(C(i))]);
end

% Cull leaves
AG = MST + MST';
for j = 1:iteration
    for i = 1:size(AG, 1)
        if sum(0 < AG(i, :)) <= 1
            AG(i, :) = 0;
            AG(:, i) = 0;
        end
    end
end
```

グラフの作成と無駄な接続の除去

- d)　葉のノードをある距離まで削除
  - 更に無駄な分岐を削除する
- この(d)に対して平滑化を掛ける
  - 論文では以下の式が紹介されている
    ・・・が、簡単な平滑化を実装しました

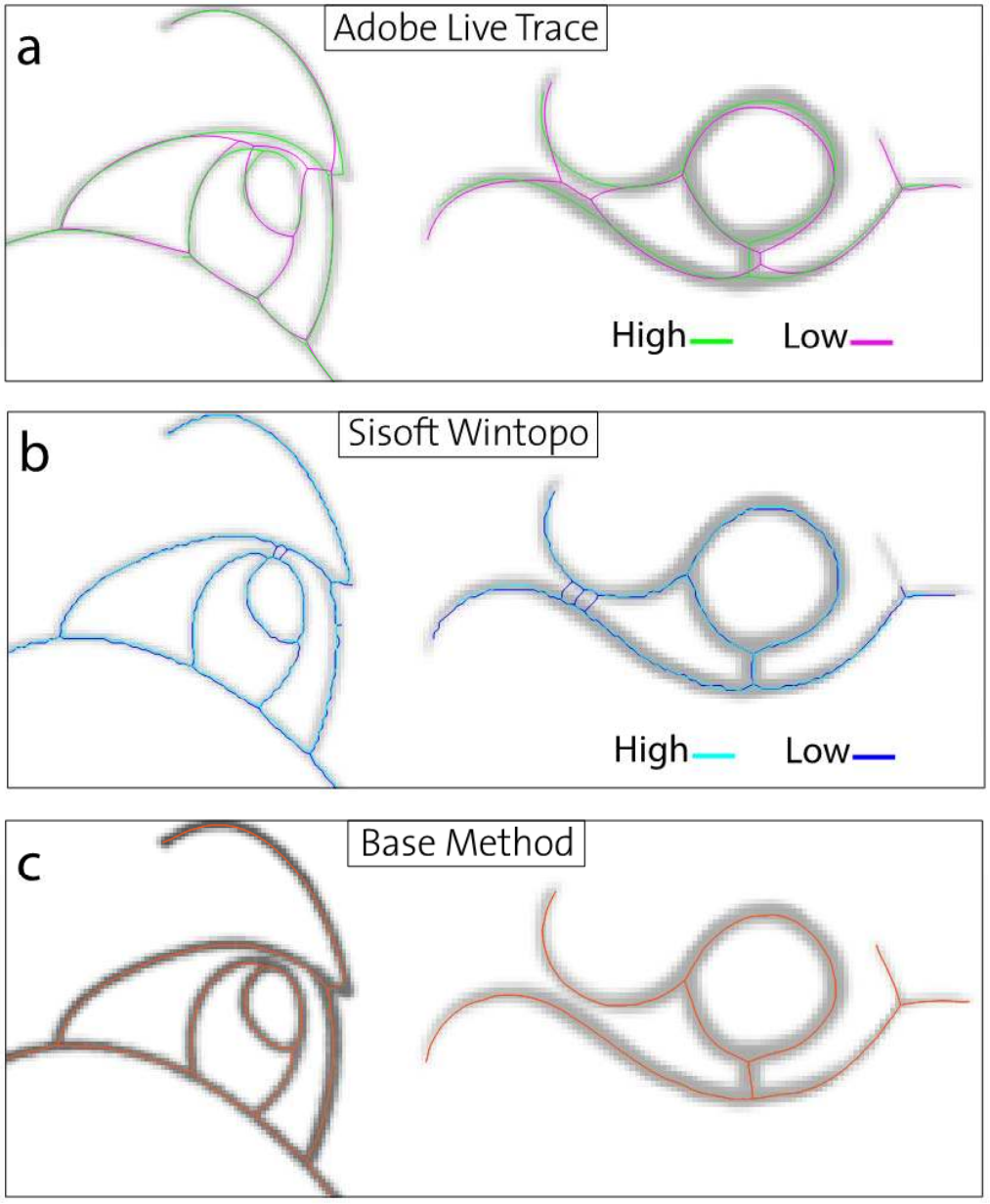
$$\tilde{\mathbf{p}}_\mathbf{i} \leftarrow \mathbf{p}_i + \left((\mathbf{c}_i - \mathbf{p}_i) \cdot \mathbf{n}_i\right)^T \mathbf{n}_i, \text{ with}$$

$$\mathbf{c}_i = \frac{\sum_j w_j \mathbf{p}_j}{\sum_j w_j}, \text{ and } w_j = e^{-\frac{D(\mathbf{p}_i, \mathbf{p}_j)}{2\sigma^2}},$$

グラフの作成と無駄な接続の除去

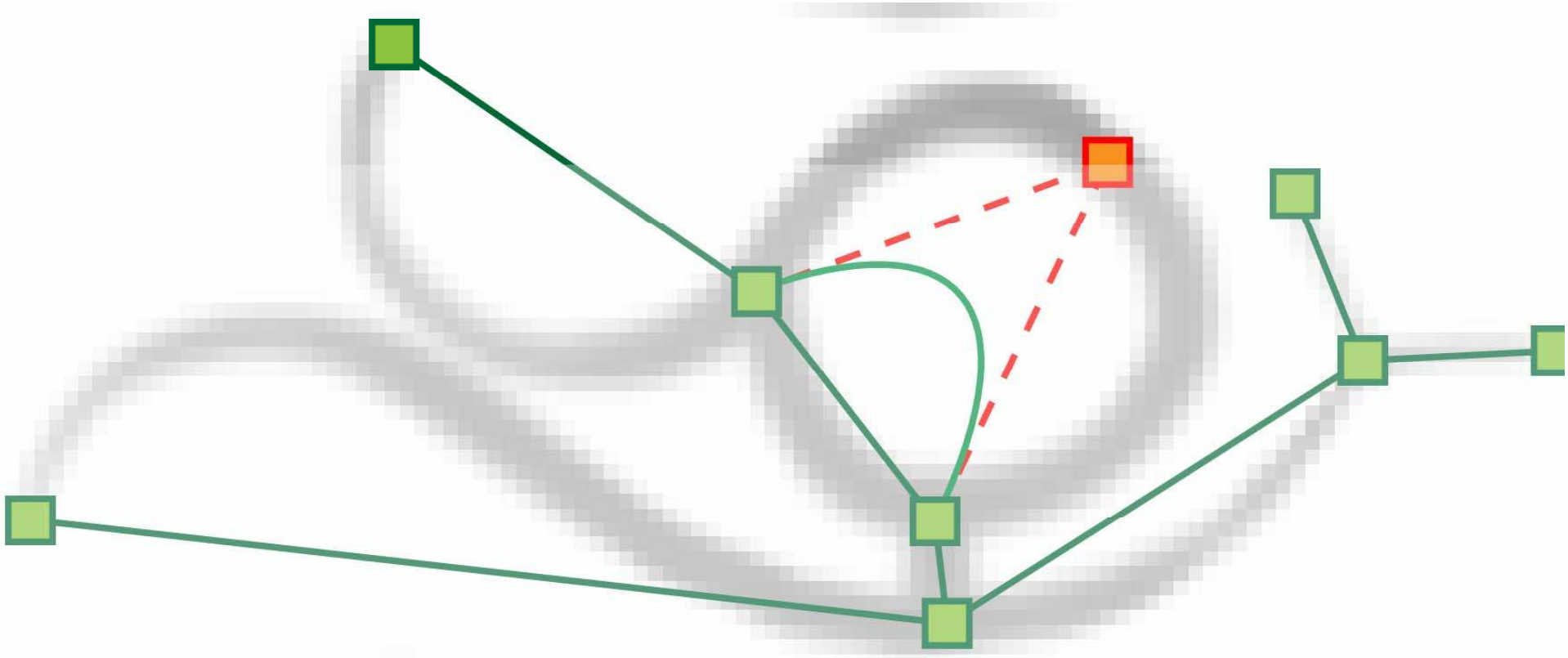
```
% Smooth the graph
for j = 1:smoothIter
    x2 = x;
    y2 = y;
    for i = 1:size(AG, 1)
        if 2 <= sum(0 < AG(i, :))
            F = find(AG(i, :));
            x(i) = mean(x2([F i]));
            y(i) = mean(y2([F i]));
        end
    end
end
```

ここまでの話の既存研究との比較



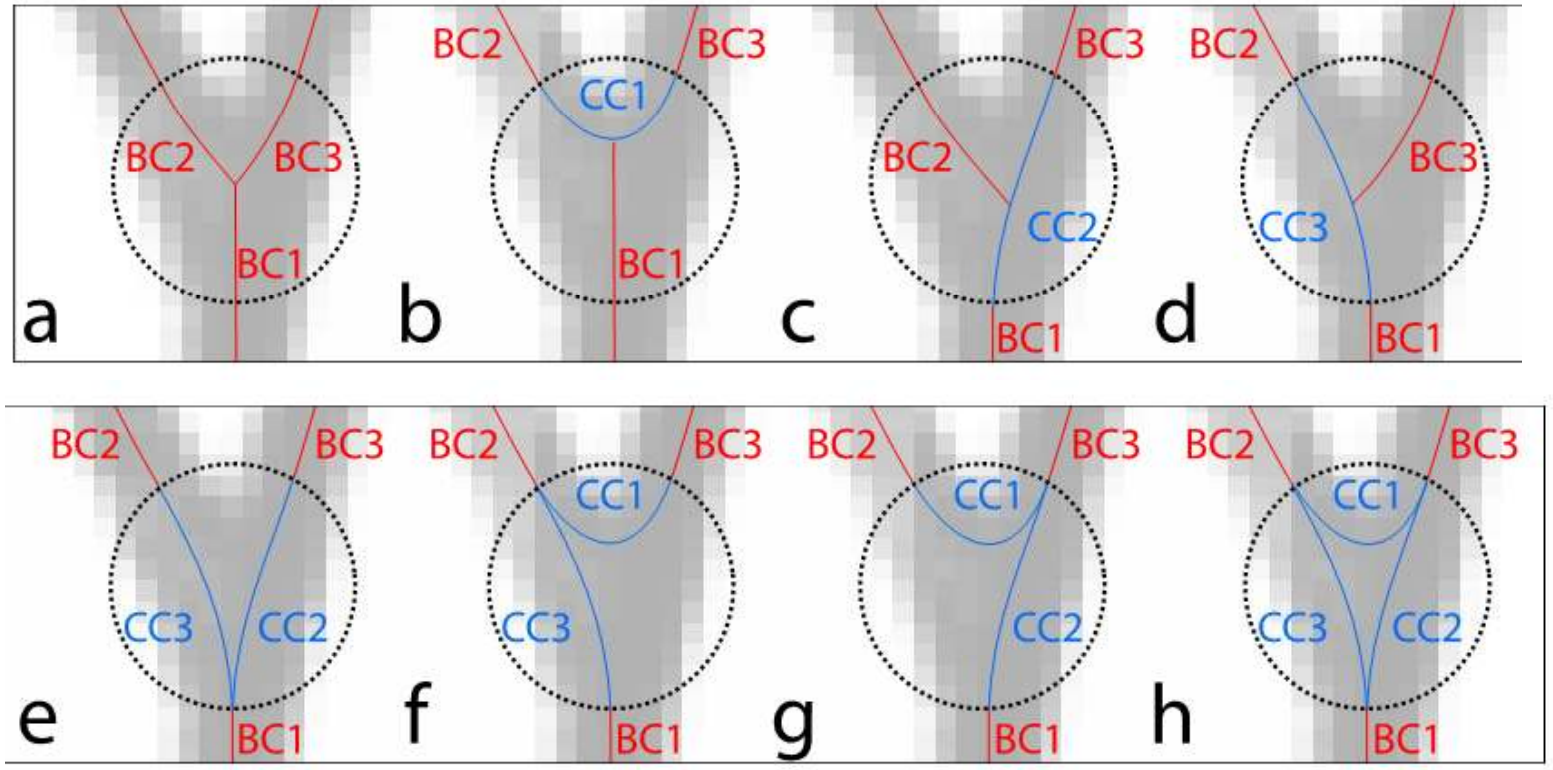
位相構造を観察しつつ、形を整える

- 更に、位相構造を考える
- 先程のグラフのJunctionに注目
  - 接続数が3のノードに注目



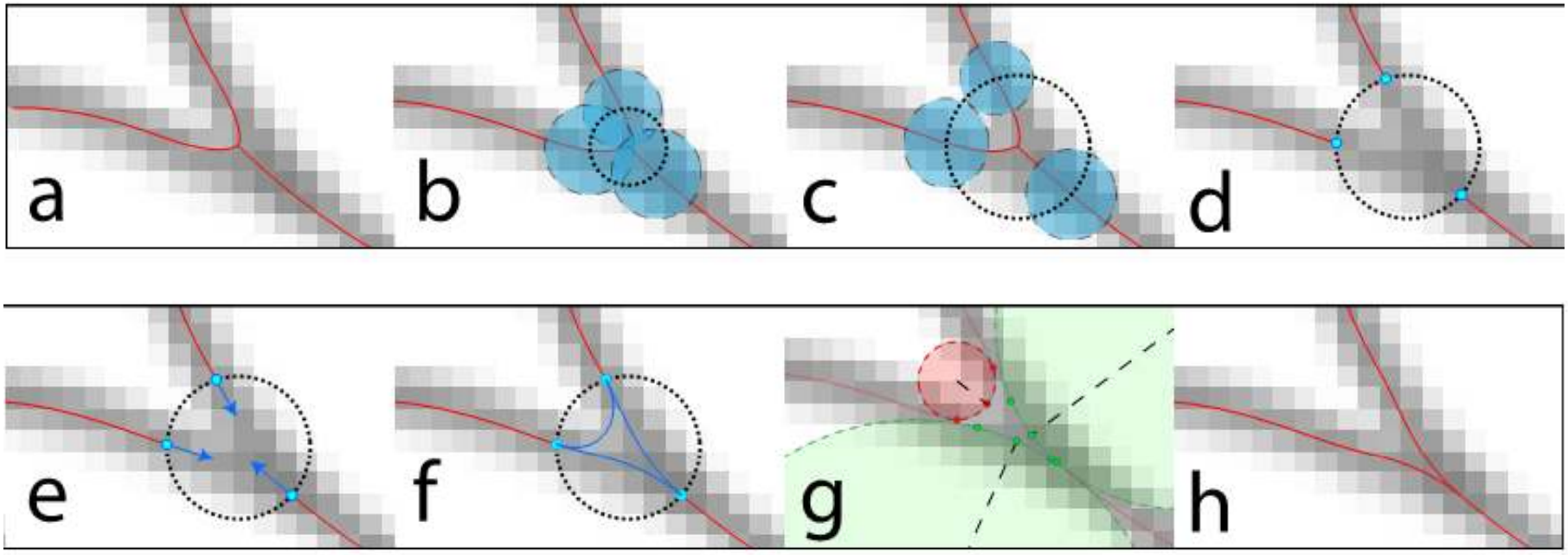
位相構造を観察しつつ、形を整える

- Junctionでありえるパターン



位相構造を観察しつつ、形を整える

- Junctionでありえるパターン
  ・・・を以下のアルゴリズムで観察する
- なるべく曲率の小さなカーブ（緑円）を採用

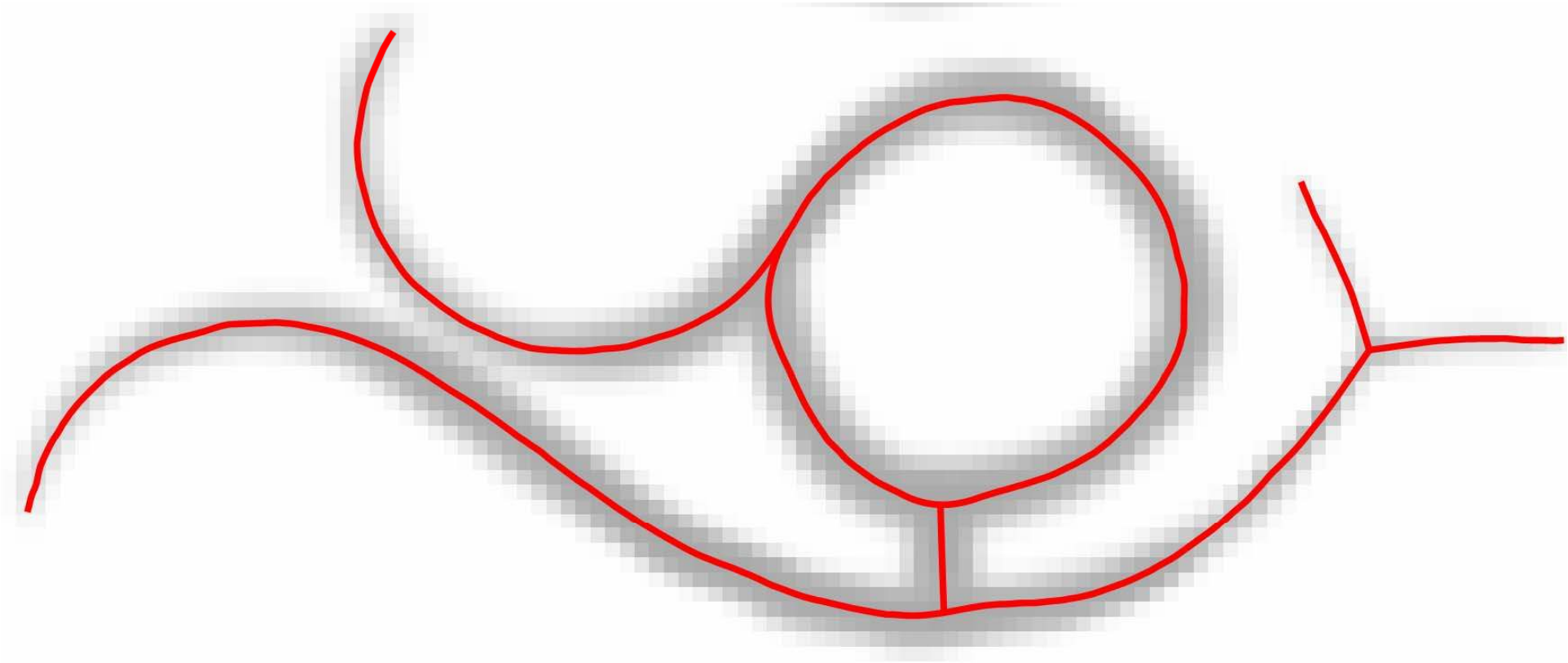


位相構造を観察しつつ、形を整える

```
% Modify junctions
AG2 = AG;
x2 = x;
y2 = y;
for i = 1:size(AG, 1)
    if sum(0 < AG(i, :)) == 3
        r = 3;

        E = [];
        Rmv = [];
        for j = 1:size(R, 1)
            s = sqrt((x(R(j)) - x(i))^2 + (y(R(j)) - y(i))^2) - r;
            t = sqrt((x(C(j)) - x(i))^2 + (y(C(j)) - y(i))^2) - r;
            if s < 0 && 0 < t
                E = [E j];

                % Remove the edges at R(j)-C(j)
                AG2(C(j), R(j)) = 0;
                AG2(R(j), C(j)) = 0;

                Rmv = [Rmv R(j)];
            end
        end

        % Remove the junction
        while 0 < size(Rmv, 2)
            Rmv = [Rmv find(AG2(Rmv(1), :))];
            AG2(Rmv(1), :) = 0;
            AG2(:, Rmv(1)) = 0;
            Rmv = Rmv(1, 2:end);
        end

        CDS = [];
        for j = 1:size(E, 2)
            e1 = E(j);
            e2 = E(mod(j, size(E, 2)) + 1);
            c1 = [x(C(e1)); y(C(e1))];
            r1 = [x(R(e1)); y(R(e1))];
            c2 = [x(C(e2)); y(C(e2))];
            r2 = [x(R(e2)); y(R(e2))];
            cps = [c1, r1 + (r1 - c1) * r * 2, r2 + (r2 - c2) * r * 2, c2];
            cds = bezier([0:0.1:1], cps);
            CDS(:, :, j) = cds;
        end

        minCurvature = 1;
        for j = 1:size(CDS, 3)
            curvature = 1;
            for k = 1:size(CDS, 2) - 2
                v1 = [CDS(1, k, j); CDS(2, k, j)];
                v2 = [CDS(1, k + 1, j); CDS(2, k + 1, j)];
                v3 = [CDS(1, k + 2, j); CDS(2, k + 2, j);]
                v12 = (v2 - v1);
                v12 = v12 / norm(v12);
                v23 = (v3 - v2);
                v23 = v23 / norm(v23);
                curvature = min(curvature, dot(v12, v23));
            end
            if curvature < minCurvature
                minCurvature = curvature;
                minIndex = j;
            end
        end

        for j = 1:size(CDS, 3)
            if j ~= minIndex
                c = C(E(j));
                for k = 1:size(CDS, 2)
                    x2 = [x2; CDS(1, k, j)];
                    y2 = [y2; CDS(2, k, j)];
                    AG2(size(x2, 1), c) = 1;
                    AG2(c, size(x2, 1)) = 1;
                    c = size(x2, 1);
                end
            end
        end
    end
end
```

結果

- 綺麗なベクトル化が可能



数学者の皆様への課題

- スムージング:
  - より効率よい、より質の良いアルゴリズムの開発
- 位相構造部分:
  - 何故このアルゴリズムで上手くいくのか？
- 3次元への拡張
  - 3次元形状においても全く同様の問題が存在