Silicon Studio
Ideas × Art × Technology

次世代 NPR の輪郭線表現に迫る！

リアルタイムベクトルストローク生成手法 StrokeGen を徹底解剖

CEDEC2024 08.21

シリコンスタジオ株式会社
研究開発室
川口龍樹

撮影OK　SNS投稿OK

後日
資料公開

0 はじめに

Silicon Studio
Ideas × Art × Technology

川口　龍樹　Tatsuki KAWAGUCHI

テクノロジー事業本部　技術統括部　研究開発室

2017年シリコンスタジオ株式会社に新卒入社
リアルタイムレンダリングにおける技術の研究開発に従事

CEDEC 2021 では「リアルタイムレイトレーシング時代を生き抜くためのデノイザー開発入門」を講演

近年は NPR の研究開発に携わり、中でも StrokeGen 技術の実用化に取り組んでいる

0 StrokeGen とは

Silicon Studio
Ideas × Art × Technology

論文

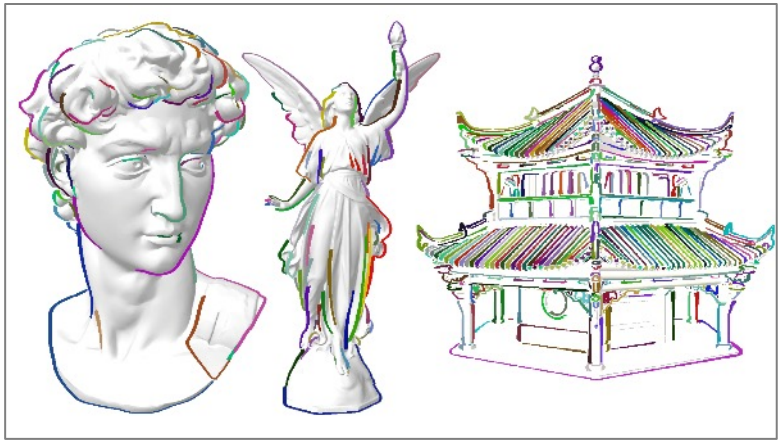
GPU-Driven Real-Time Mesh Contour Vectorization (EGSR 2022)
[Wangziwei Jiang, Guiqing Li, Yongwei Nie, Chuhua Xian]

論文のプロトタイプ実装

StrokeGen: Realtime Contour Curve Generation
https://github.com/JiangWZW/Realtime-GPU-Contour-Curves-from-3D-Mesh

- 既存のCPUベースオフライン手法よりも最大800倍高速
  - Pencil+4, Line Art, Freestyle, Active Strokes
- FHD 描画で 300k△ を 1ms で処理

StrokeGen はゲーム NPR における新しい手法となり得るのか？



Generated strokes with different colors.

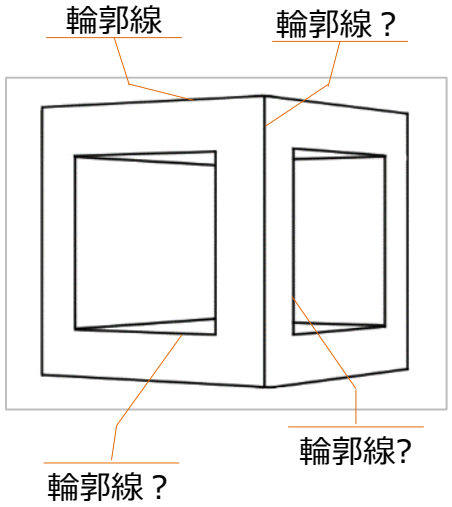
0 はじめに

Silicon Studio
Ideas × Art × Technology

1. 輪郭線とは？
2. StrokeGen 概要
3. StrokeGen アルゴリズム解説
4. StrokeGen 実装解説
5. デモ・結果紹介
6. 課題と今後の展望

Silicon Studio
Ideas × Art × Technology

1 輪郭線とは？



Silhouette （シルエットライン）もたまにみる

↕

輪郭線 ＝ Contour or **Outline ＝ アウトライン**

同じ？

輪郭線を取り巻く用語は文脈や現場でバラバラ

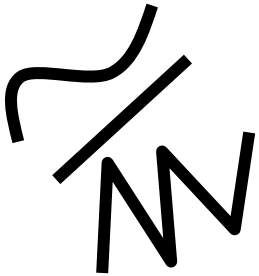
イラストの線を指して「ここが輪郭線、ここはそれ以外の線」と分類できるようなものではない
StrokeGen の扱う **(Mesh) Contour** は後で定義するが
講演の中で輪郭線を表す用語は明確に**区別して扱わない**

1 輪郭線とは？

Silicon Studio
Ideas × Art × Technology

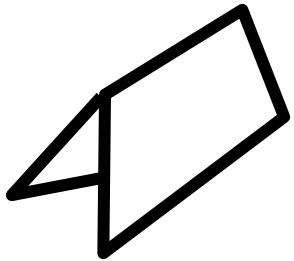
Line
ライン

あらゆる種類の線全般
直線、曲線、線分・・・



Edge
エッジ

線の意味で使うこともある
形状と関連している
（辺、縁）



Stroke
ストローク

線を描く一筋を表す
動きのニュアンスがある
（筆跡）

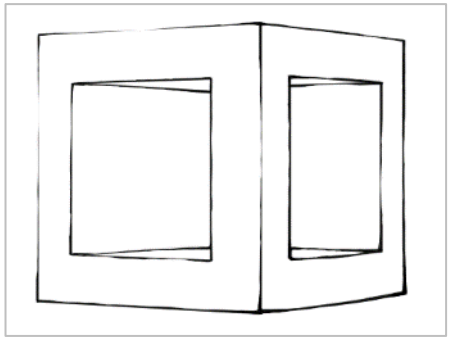
本公演における**ベクトルストローク**とは、輪郭線を構成する**個々の線**であり、
始点から終点への**繋がりを持った点列**（ベクトルデータ）
パラメトリックな曲線ではない

1 イラストにおける輪郭線

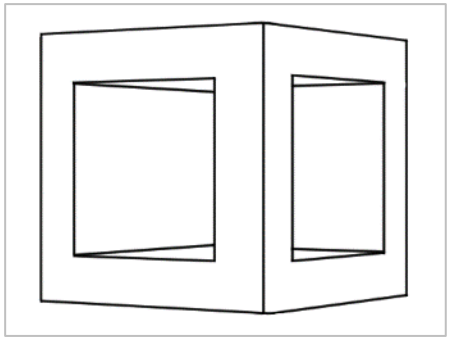
イラストにおいては輪郭線という言葉はあまり使わず、
線だけの絵を**線画**と言いその中に輪郭やそれ以外の線が混在している

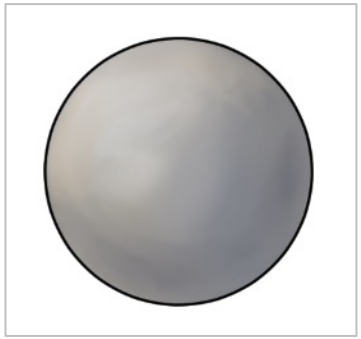
奥行き・明るさ・動きなど作者が表現したい**効果を線画に仕込む**テクニックがある
（テクニックに限らず筆圧や手ぶれによって線画に**味**が付いている）

・線に**強弱や色**を付ける
・あえて**形状を無視**した線を描く



均一な線は無機質な印象
時間的な整合性をつけるためのアニメの線画に近い

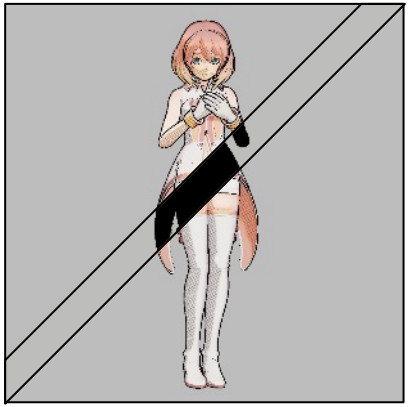




1 ゲームにおける輪郭線

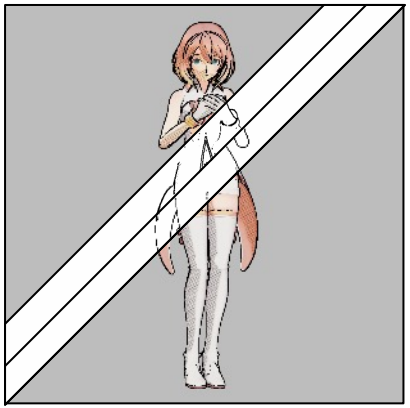
Silicon Studio
Ideas × Art × Technology

Geometry-based
（反転法・押し出し法・Inverted hull）



裏返して拡大したメッシュを通常のメッシュに重ねることで輪郭線のように見せる手法
シンプルで利用しやすいが、線が割れたり描けない線があったりと制限も多い

Image-based
（Sobel fileter, Laplacian filter）



画像処理フィルタを利用して輪郭線を描画する手法
様々な種類の線を精細に描くことが出来るが、画面全体の効果なので期待する絵を得るための調整が効きにくい

その他
テクスチャ書き込み・ジオメトリ配置等

テクスチャやジオメトリなどの3Dシーンオブジェクトとして輪郭線を配置する手法

これらの手法の利点
- 実装が容易
- 処理が高速
- 破綻が少ない

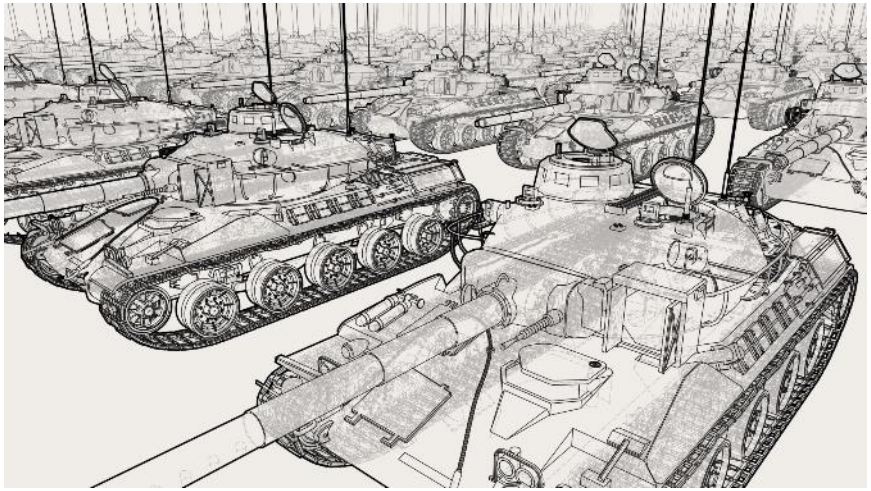
これらの手法の欠点
- 線幅の制御性が低い
- 元々の形状に拘束される
- 手法ごと描画できない種類の線がある

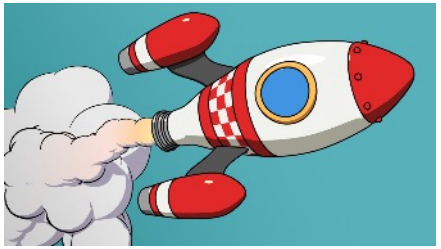
1 アニメの 3DCG における輪郭線

Silicon Studio
Ideas × Art × Technology

オフラインの輪郭線レンダリングはベクトルデータを扱うことができる
コンポの工程で線に多彩な演出を加える
そもそものアニメの線画が動きを考慮して演出は控えめ

フレーム単位で調整していいのであればイラストに近い線画も表現可能？



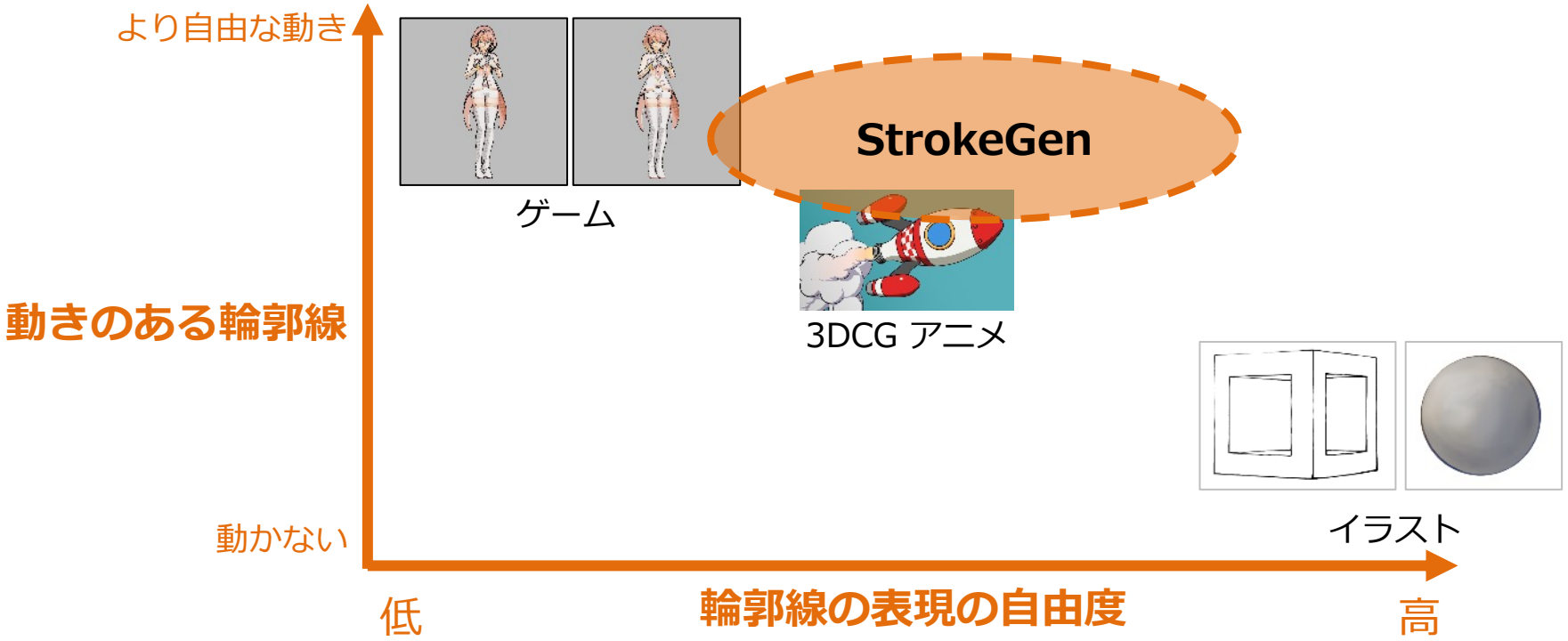


Pencil+
Advanced Nonphotorealistic Renderer

PSOFT Pencil+ 4 for 3ds Max | PSOFT WEBSITE

1 輪郭線の表現

Silicon Studio
Ideas × Art × Technology



StrokeGen はゲームで使える輪郭線描画を
アニメやイラストの表現力に近づけることができる技術

2 StrokeGen 概要

Silicon Studio
Ideas × Art × Technology

1 輪郭線とは？

2 StrokeGen 概要

3 StrokeGen アルゴリズム解説

4 StrokeGen 実装解説

5 デモ・結果紹介

6 課題と今後の展望

2 StrokeGen 概要

Silicon Studio
Ideas × Art × Technology

StrokeGen 論文の概要

GPU-Driven Real-Time Mesh Contour Vectorization (EGSR 2022)
[Wangziwei Jiang, Guiqing Li , Yongwei Nie, Chuhua Xian]

「3Dメッシュから**輪郭線のベクトルストローク**を**リアルタイム生成**するシステム」を提案

StrokeGen 論文の Contribution

ベクトル化の一連の処理をすべて GPU による並列化

- CPU⇔GPU 間のデータのやりとりを無くすことで効率化
- 繋がりを持つデータ構造の並列処理
- **Potrace の並列化**

StrokeGen の描く輪郭線

ベクトルストロークなので 3DCG アニメの線に近しい手法＋**リアルタイム**
ベクトルストロークの描画方法は論文の内容ではない

2 StrokeGen 概要 – 本公演について

Silicon Studio
Ideas × Art × Technology

本公演の動機

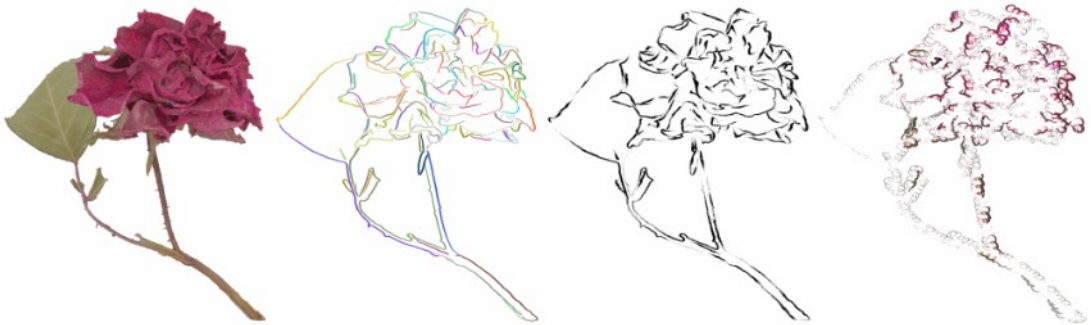
リアルタイムベクトルストローク生成を実現したという手法に
従来のゲームの輪郭線では為し得なかった表現の**可能性**と**魅力**を感じ
ここ1年ほど調査と検証を行った結果を本公演で紹介

本公演の概要

- StrokeGen のアルゴリズム解説&結果紹介
- StrokeGen へ行った改造実装の紹介
- ベクトルストロークでどのような表現が可能か検討
- StrokeGen の考察

2 StrokeGen 概要 – StrokeGen の輪郭線

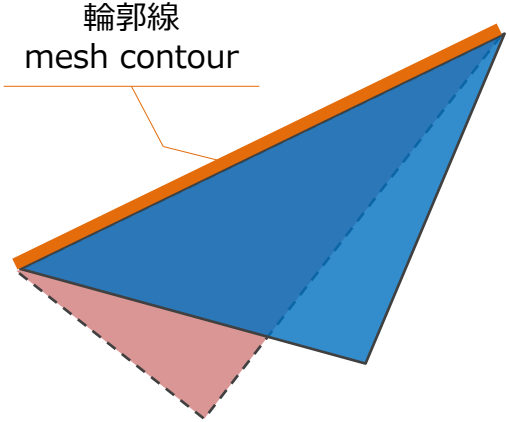
Silicon Studio
Ideas × Art × Technology

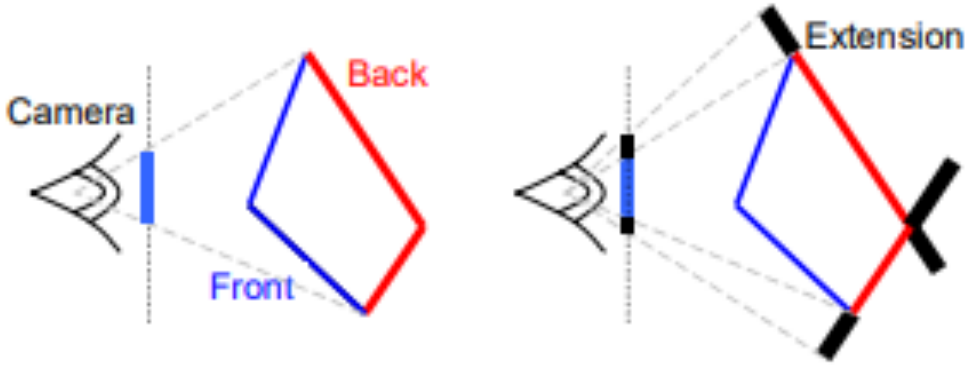


StrokeGen 概要 - 輪郭線の定義

メッシュの輪郭線（Mesh Contour）
ある視点から物体を見たとき、**一つの辺を共有する二つの面が逆向き**ならば
その辺は**輪郭線（輪郭エッジ）**である

輪郭線という言葉の広さと多少齟齬はあるが CG においてはわりと一般的な定義
押し出し法もこの定義における輪郭線を可視化していると見なせる





Hardware Support for Non-photorealistic Rendering [Raskar 2004]

2 StrokeGen 概要

Silicon Studio
Ideas × Art × Technology

入力:
三角形メッシュ

出力:
ベクトルストローク

Preprocessing
メッシュの隣接情報や
ジオメトリ情報を収集

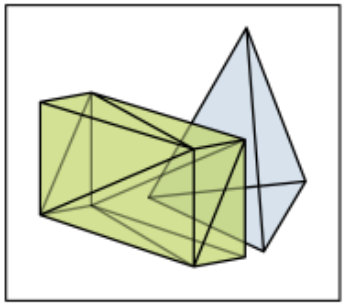
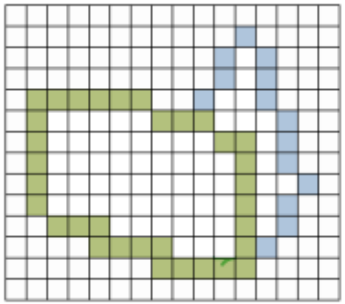
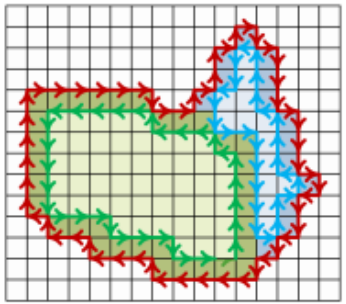
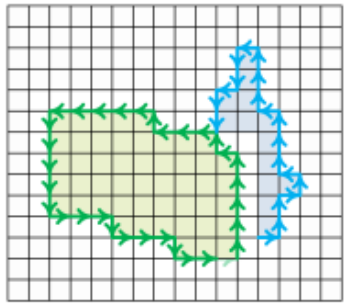
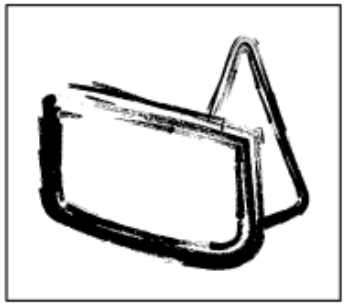
Rasaterization
輪郭エッジをラスタライズ

Vectorization
ラスタライズされた
ピクセル境界をトレース
してループをベクトル化

Stroke Generation
ベクトル化されたループ
からストロークを抽出

Stylization
ベクトルストロークを
任意の方法で描画

毎フレーム処理

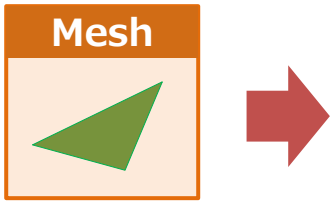
3 StrokeGen アルゴリズム解説

Silicon Studio
Ideas × Art × Technology

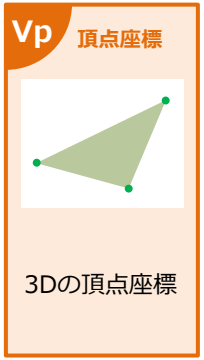
必要なデータの収集

Silicon Studio
Ideas × Art × Technology

入力：輪郭線描画の対象とする（単一スタティック[*1]）メッシュ
出力：各種データを収めた GPU バッファ
実行時に一度だけ処理[*2*3]
基本的に Vertex/Index Buffer から取り出すだけ
隣接情報は自前で計算する必要がある



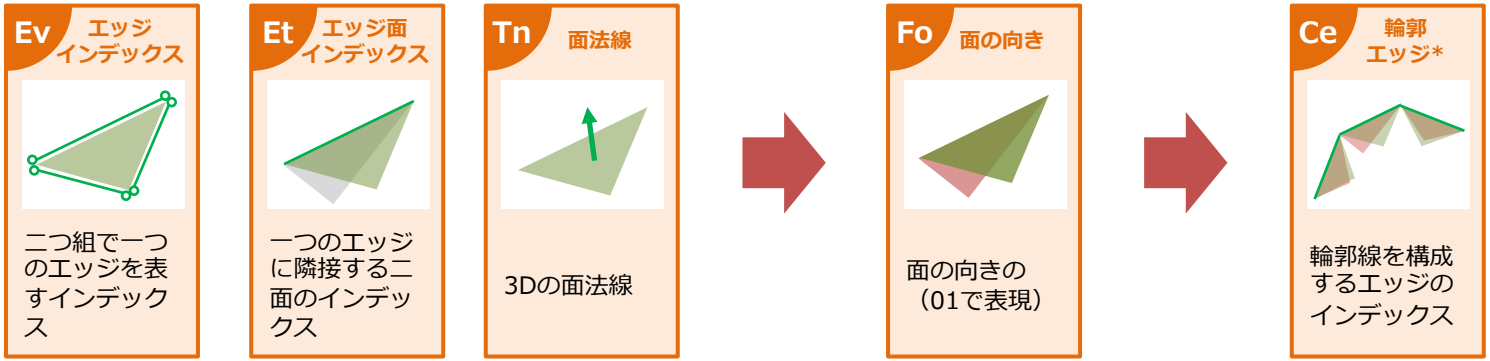
VB / IB



*1*2 公開されている StrokeGen の仕様。アルゴリズムの制約ではない
*2 公開実装には Asset 化して使い回すような処理も確認できる（動作未検証）
*1*3 動的更新への対応は後述

輪郭エッジの抽出

ビューポイントと法線から面の向きを計算
メッシュ輪郭線の定義から輪郭線を構成するエッジだけを取り出す
（GPU で毎フレーム処理）
処理の対象を輪郭エッジだけにすることで以降の処理を大幅に軽減できる*1



*1 面の数 N_f に対して輪郭エッジの数は $N_f^{0.8}$ 程度になる [McGuire 2004]
e.g., 10K tris -> 1.6K edges、100K tris -> 10K edges
* 輪郭エッジのようなサイズが不定なデータは、
大きなバッファを用意して必要な量を毎フレーム計算して利用する

輪郭エッジのラスタライズ

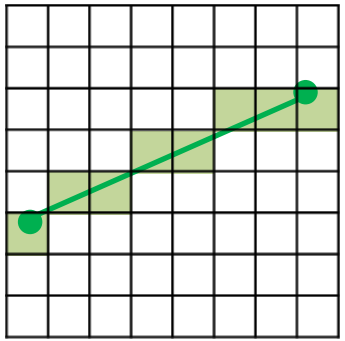
2パスの**並列化 Bresenham** を利用して輪郭エッジをラスタライズ
入力：輪郭エッジ　　**出力**：輪郭フラグメントとジオメトリ属性

Fragment conting pass

- ラスタライズする輪郭エッジのフラグメントの数を数える

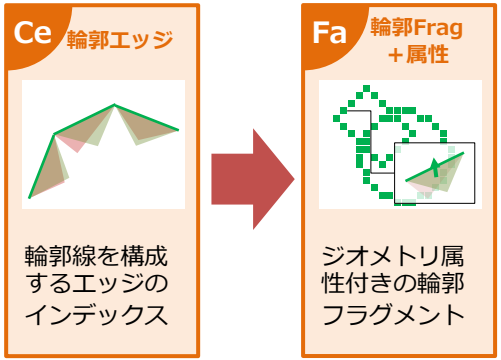
Fragment generation pass

- エッジの開始点・終点とフラグメント数からその座標を計算
- 各フラグメントはそのジオメトリ属性等を保持
  - ピクセル座標、デプス、法線、**エッジベクトル**[*1] を補間



1. フラグメントの数を数える
   開始点から終了点までの長さを解像度で分割
2. 得られたフラグメントの番号（ここでは0~7）で
   開始点終了点の位置を補間して座標を決める
3. 得られた座標がフラグメントのピクセル座標になる
   元となるジオメトリの属性も取得する

[*1] エッジに隣接する面から、表向きの面の winding order（面を定義する頂点の順序）でエッジの向きを決める



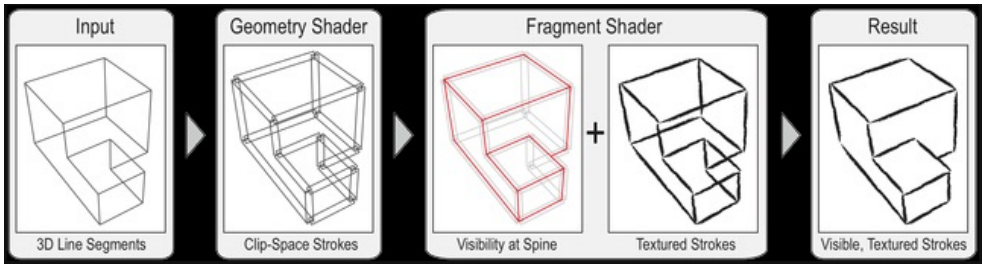
輪郭ピクセルの生成

輪郭エッジのラスタライズは可視性を考慮していない
入力：輪郭フラグメント＋属性　　**出力**：可視の輪郭ピクセル

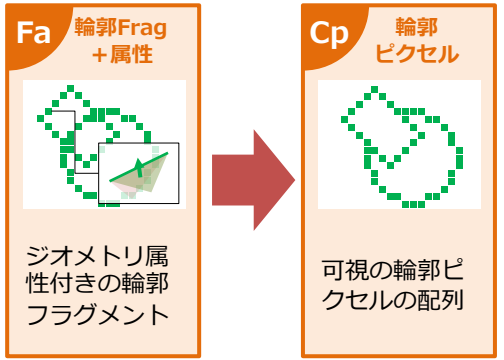
輪郭線の可視性は Challenging Problem

- Two fast methods for high-quality line visibility [Cole 2010]
- Computing smooth surface contours with accurate topology [Be’nard 2014]

StrokeGen では Soft と Hard の**2段階デプステスト**で可視性を評価



ラインの可視性について論じている一例
Two Fast Methods for High-Quality Line Visibility (princeton.edu)



輪郭ピクセルの生成（可視判定）

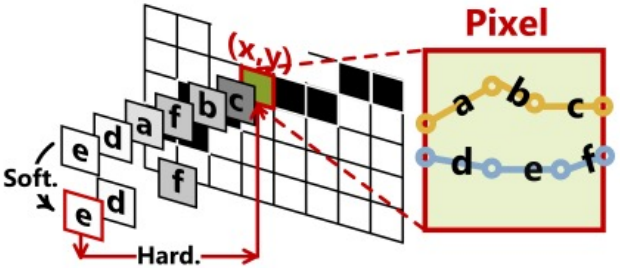
Soft Depth-Test Pass

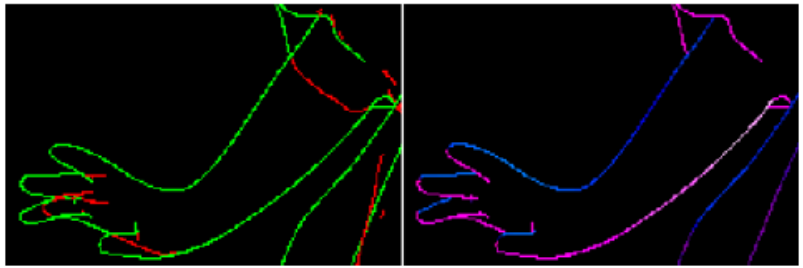
- シーンのデプスは事前に描画されているとする
- 輪郭属性に保存したデプスとシーンのデプス（3x3）を比較
  - フラグメントが 3x3 デプスのうち二つ以上の前面にあるならテストをパス

  → pseudo-visible fragment (pv-frag)

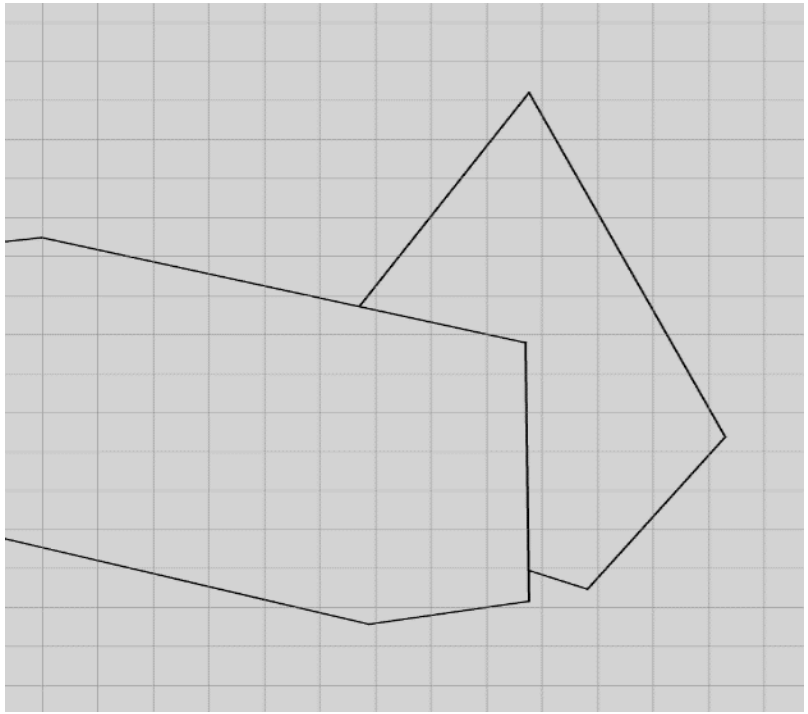
Hardware Depth-Test Pass

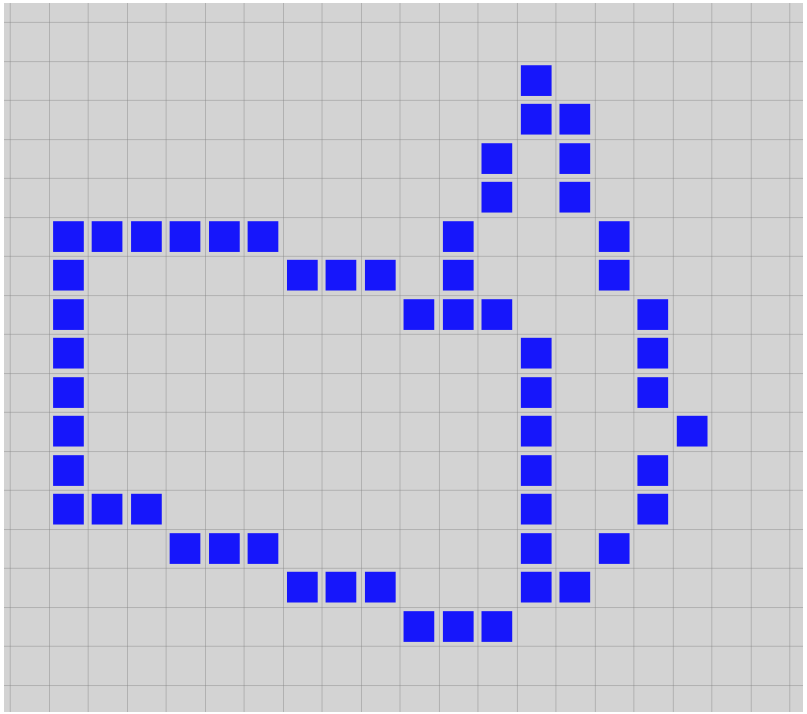
- pv-frag を1ピクセルサイズの点としてラスタライズ描画
- HW のデプステストで最前のもの一つだけが選ばれる
- 輪郭属性の bit を色にエンコードして書き出す → GBuffer → 輪郭ピクセルバッファ





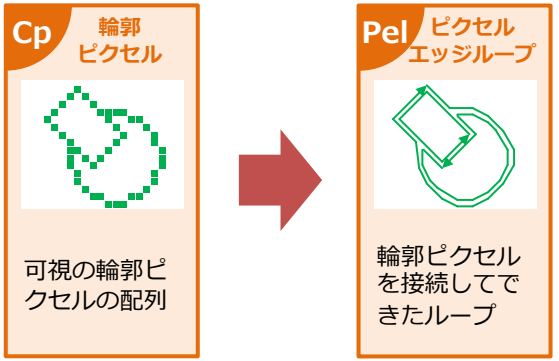
ラスタライズ





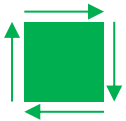
輪郭ピクセルのベクトル化

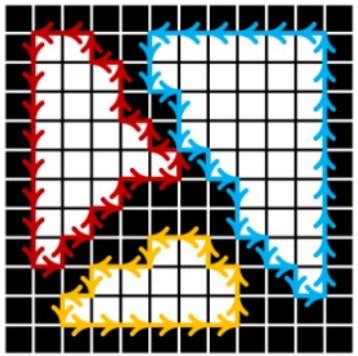
輪郭線を形成するピクセル群（輪郭ピクセル）が得られている
→ピクセル間の繋がりを求める
入力：輪郭ピクセル　　**出力**：ピクセルエッジループ



ピクセルエッジ

ピクセル矩形に向き付きのエッジを定義したもの
二値画像のピクセルエッジを辿るとループになっている


ピクセルエッジ



Contour Chaining

- **Potrace** を並列化して Chaining Process に採用

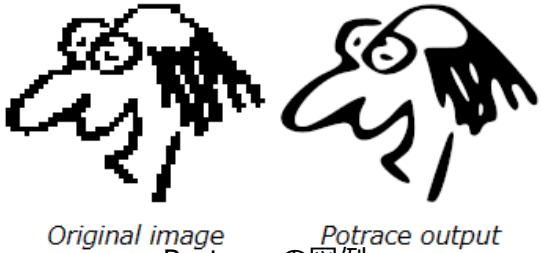
Potrace

Potrace

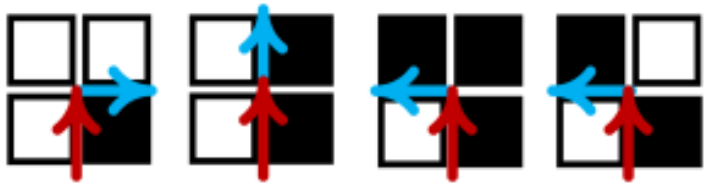
Potrace: a polygon-based tracing algorithm [Selinger 2003]

Potrace (sourceforge.net)

- 二値画像の境界をベクトル化するソフトウェア・手法
- 2x2ピクセルのパターンに当てはめて近傍ピクセルエッジ同士の繋がりを求める



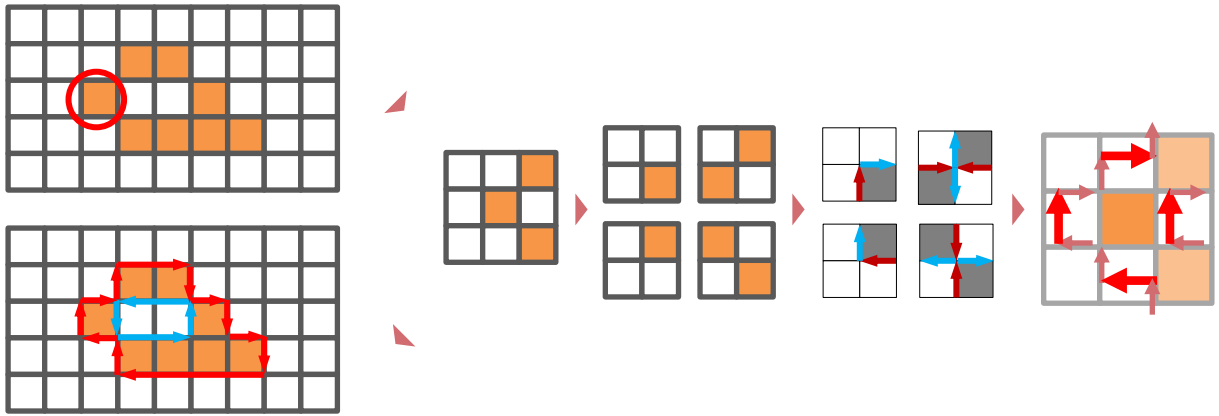
Portace の図例



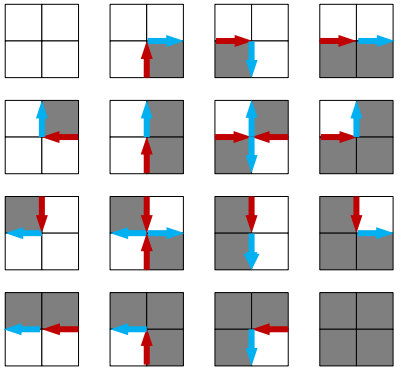
並列化 Potrace

並列化 Potrace（この論文が提案する手法）

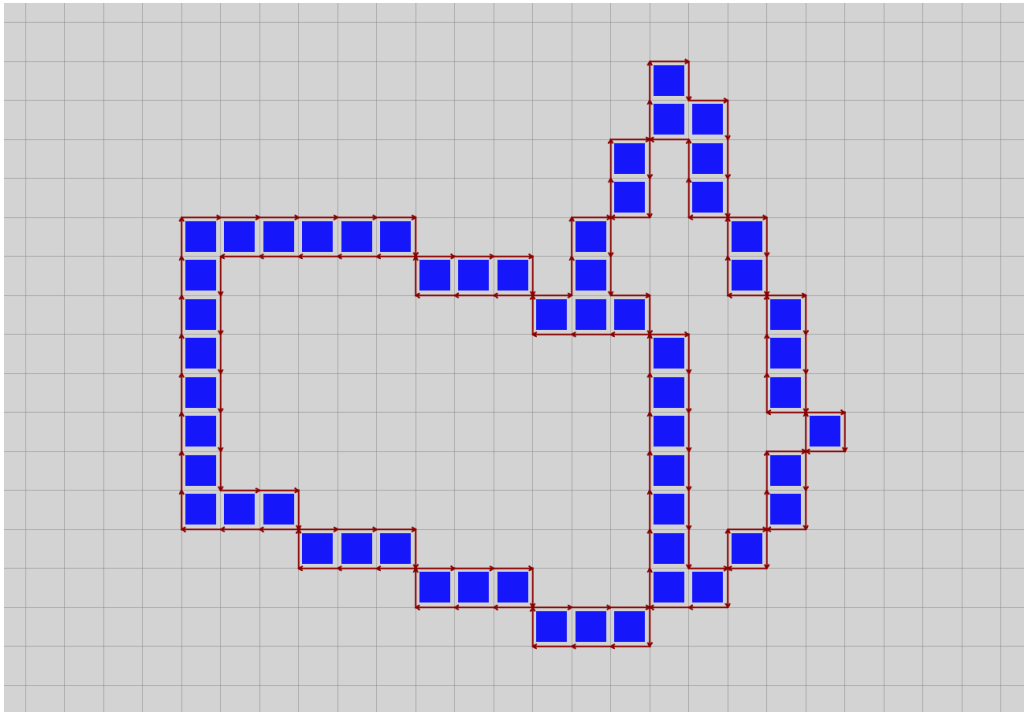
- 個々の輪郭ピクセルを並列に 3x3（2x2 を四つ）範囲を見て処理する
- 最終的にすべてのピクセルエッジについてその前後の接続情報が得られる
  - 繋がりを辿るとループになっている（**ピクセルエッジループ**）
  - ピクセルのリスト→エッジのリスト
- 元々の Potrace はカーブフィッティングなども含むが、StrokeGen はあくまで inspired としてループの作成まで



2x2のピクセルエッジパターン
（16種）



Potrace



シリアライズ

Potrace によって得られたピクセルエッジは、前後情報を辿ればループを為すが、
接続順に関する情報がないため描画に使えるベクトルデータとはいえない
→ピクセルエッジループをリニアな（接続したエッジ同士が連続した）配列に整列したい

Edge loop flatting（シリアライズ）

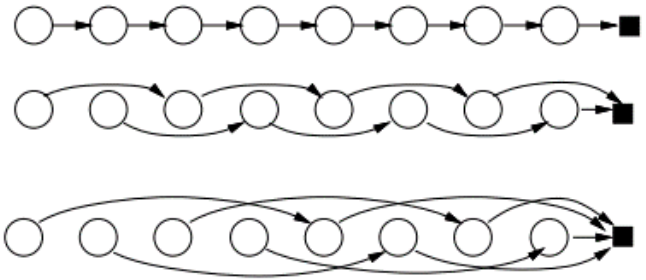
- バラバラな列（循環リンクリスト）をランキングする（順番に並び替える）問題
- 逐次処理は簡単だが並列化は複雑（**Parallel List Ranking Problem**[*1]）
- StrokeGen では問題解決を Loop breaking と List Ranking の二処理で行う

[*1] The Complexity of Parallel Computations [Wyllie 1979] で提起されたリンクリストにおける各ノードのランク割り当てを並列に行う問題
リストの先頭ほど小さいランクをもつように割り振る

リストランキング

Wyllie’s algorithm（Parallel List Ranking）

- 接続関係があるノード列に並列処理で番号を割り振る手法
- **Pointer jumping** ともいう
- 反復処理で接続関係を次→次の次→次の次の次→…と
  ポインタをジャンプするように繋ぎ直しながらランクを更新して行く
- StrokeGen ではシードの割り振りとランキングの二つに使う
- Wyllie の手法以降、より効率的な手法も提案されているが
  ここでは大元のシンプルな手法が採用されている



ReMiMo93.pdf (cmu.edu)

Wyllie’s Algorithm for list ranking

```
Procedure Wyllie:
    In Parallel rank[!] := 1;                          /* initialize rank */
    In Parallel while succ[head] ≠ nil do
        if succ[!] ≠ nil do
            rank[!] := rank[!] + rank[succ[!]];        /* update rank */
            succ[!] := succ[succ[!]];
        end if
    end in parallel
end Wyllie
```

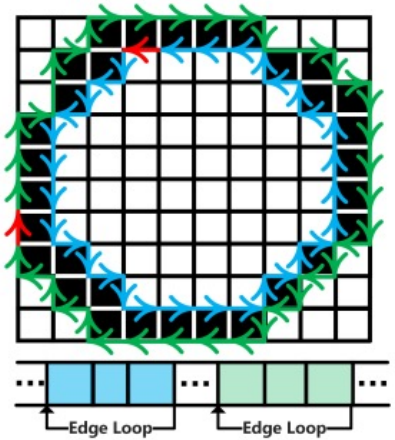
リストランキング

Loop breaking（シードの割り振り）

- ピクセルエッジループ中のシード（**ループの切れ目**）を見つける処理
- ID[*1] が最大のピクセルエッジをループの先頭とする
- Wyllie’s algorithm をループ内で最大の ID の探索に利用[*2]

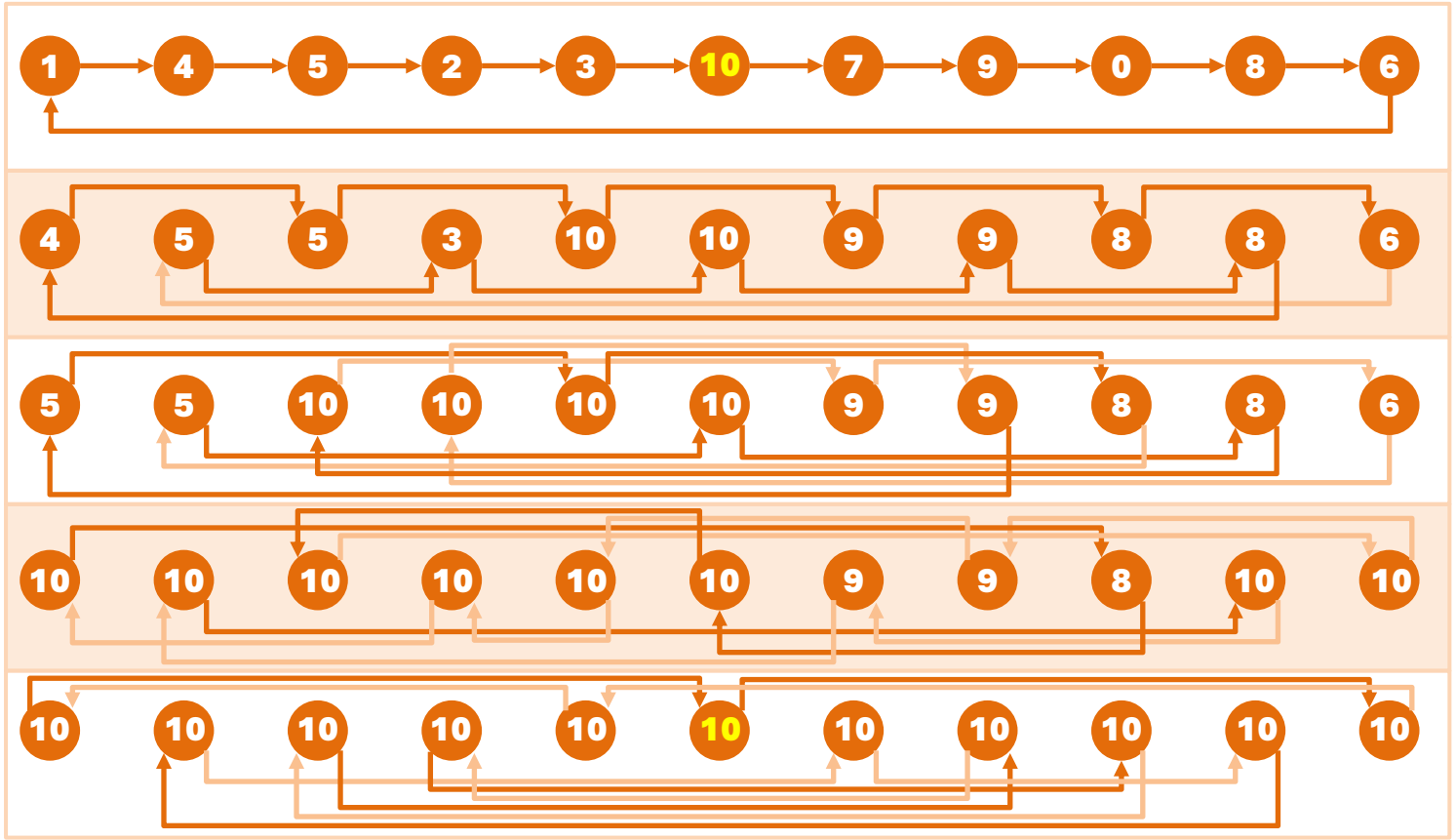
List ranking

- Wyllie’s algorithm で循環リンクリストをランク付けする
- ランク順で並び替えたエッジループの循環リンクリストを新たに作る
  - 一つのリニアな配列に複数のシリアライズされたループが並ぶ



[*1] スクリーンスペースでユニークな ID（モートンコード）を各ピクセルエッジに割り振る
[*2] 循環リストに対する Pointer jumping の過程でランクを加算する代わりに ID を max で更新すると、ループ全体にループ中最大の ID が割り振られる。
その最大 ID と元の ID が一致するピクセルを開始点として選ぶ

Loop breaking



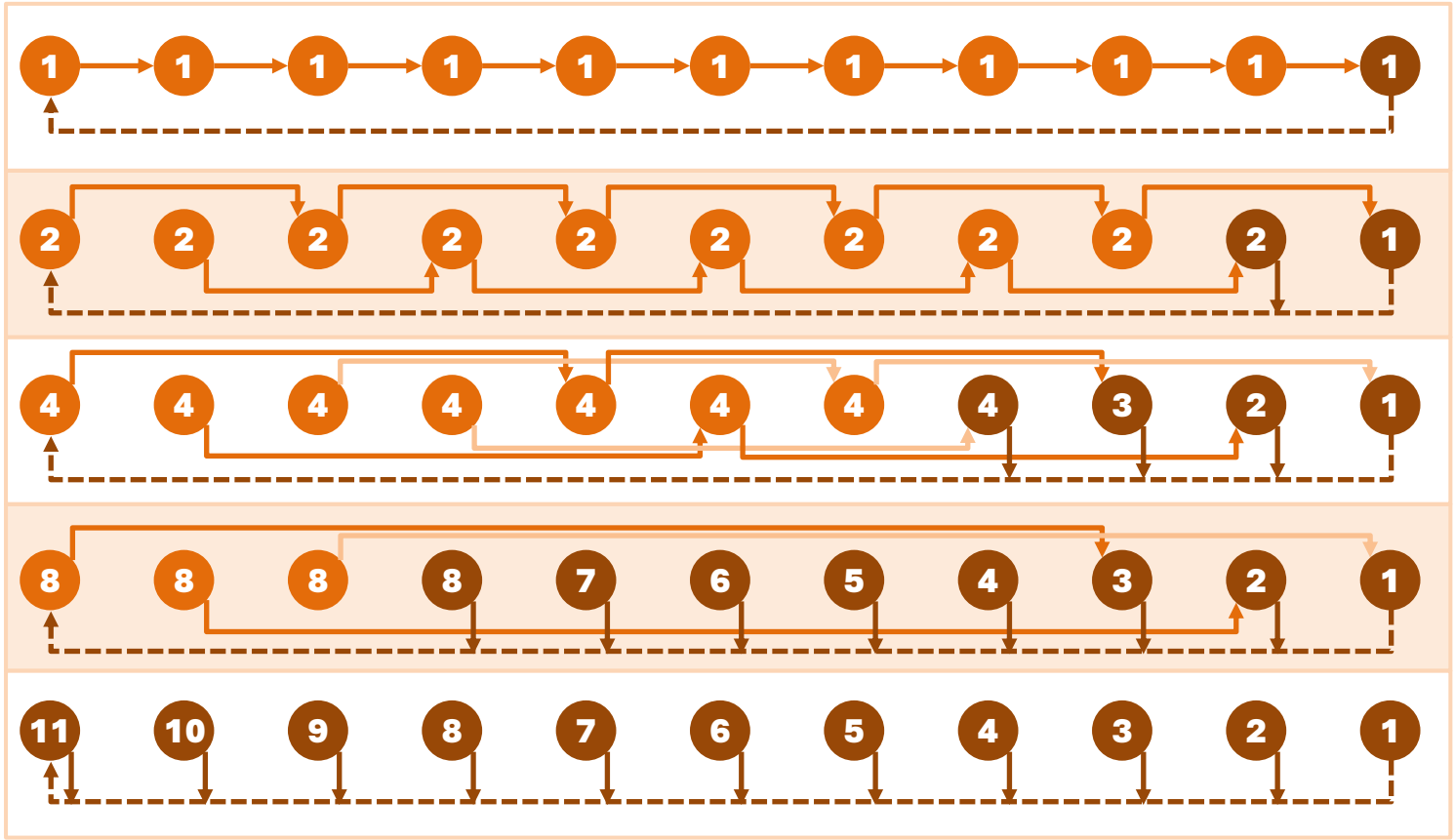
ノード毎に適当にユニークな値を割り振る

後続ノードの値と Max をとり、その後続ノードへ繋ぎ直す

末尾情報の無いループなのでずっと繋ぎ直し続ける

開始時の値と、最終的に割り振られた値を比較して、同じ値ならそこがループ中最大の値

リストランキング

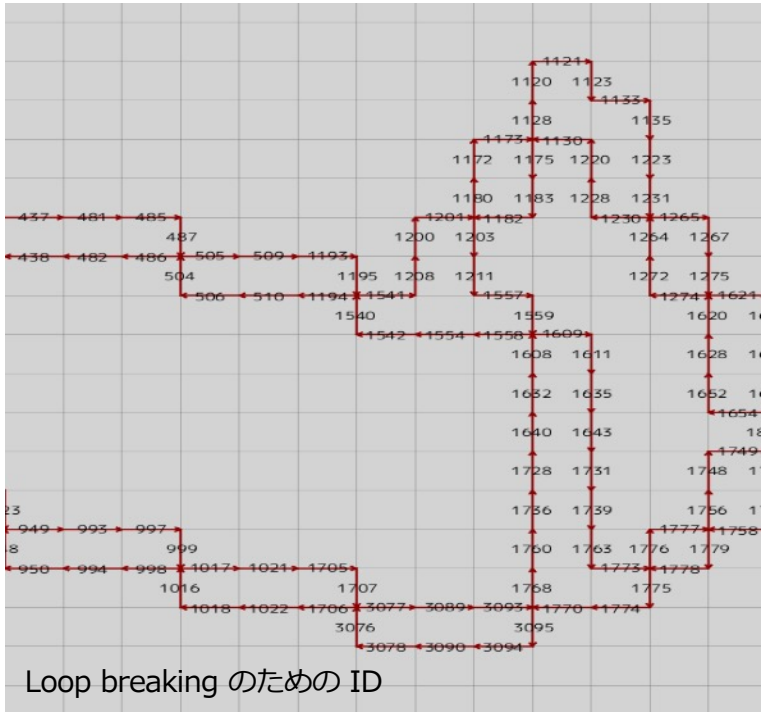


各ノードのランクを1で初期化する

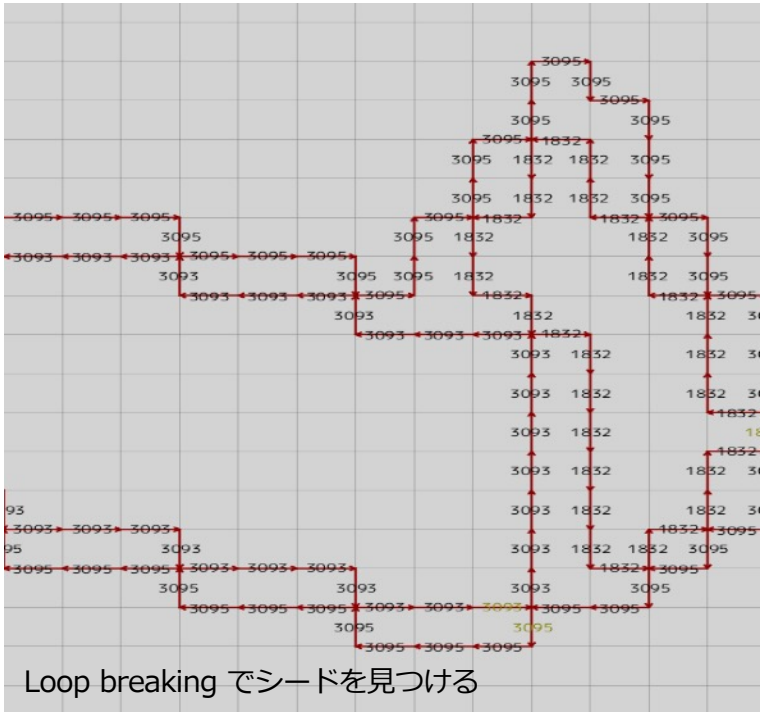
後続ノードのランクを加算して、その後続ノードへ繋ぎ直す

末尾（ここではループのシード）なら繋ぎ直しはしない

StrokeGen アルゴリズム解説 – 輪郭ピクセルのベクトル化
リストランキング&シリアライズ

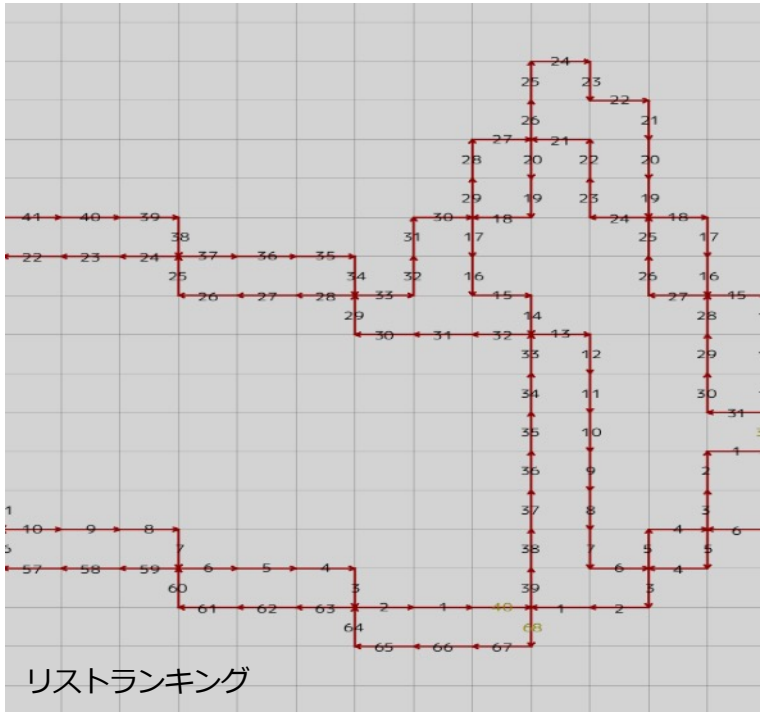


Loop breaking のための ID

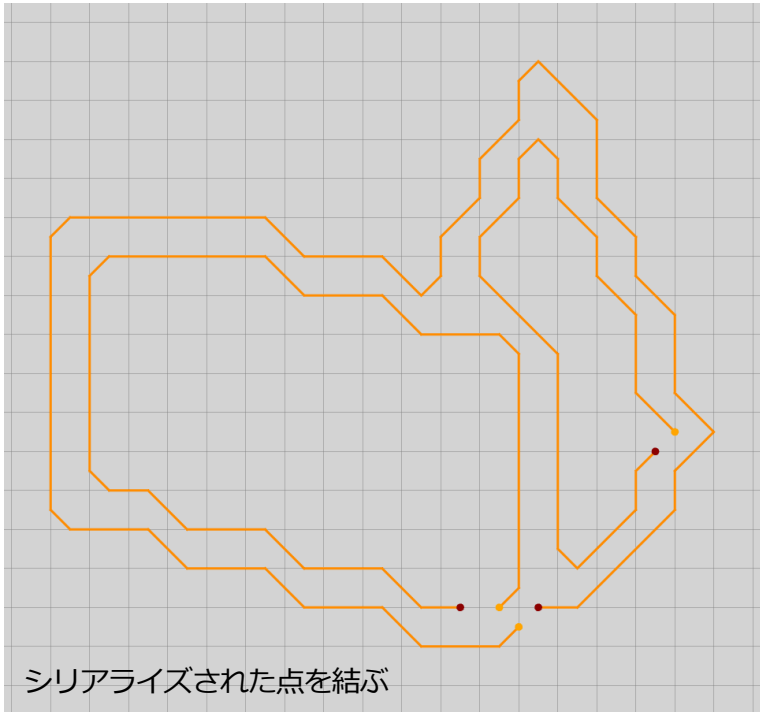


Loop breaking でシードを見つける

StrokeGen アルゴリズム解説 – 輪郭ピクセルのベクトル化
リストランキング&シリアライズ



リストランキング



シリアライズされた点を結ぶ

ベクトルストロークの生成

シリアライズされたピクセルエッジループが得られている
→描画するストロークを生成する

フィルタリング
- エッジループへ1次元の畳み込みとして空間フィルタを適用
- ラスタライズによってガタついた点列を均す効果がある（角も取れてしまう）

フィッティング
- ループの局所的な形状で曲線フィッティングを行い各点における接線ベクトルを推定

カリング
- エッジループの中からストロークとして不要な部分を削除する

セグメンテーション
- Segmented scan で改めて描画するためのベクトルストロークを抽出する

ベクトルストロークの生成

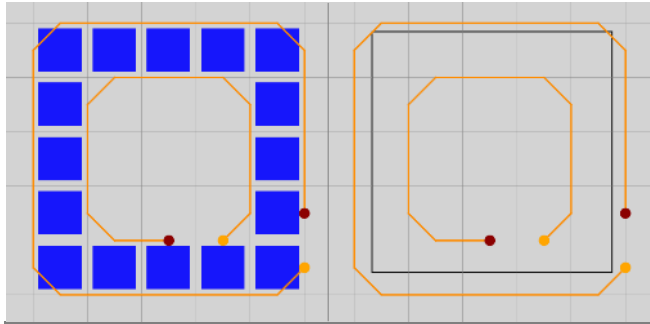
- Potrace は領域境界をベクトル化する手法なので、
  ピクセルエッジループは輪郭線に対して両側（二重）の線を示す
- ピクセルエッジはループになっているが、輪郭線は必ずしもループではない

現在得られているピクセルエッジループは**描画するべき輪郭線になっていない**

↓

描画するためのストロークを生成する

- 期待する輪郭線に対応するピクセルエッジを選ぶ
- 可視性の変化する（線が交差したり途切れたりする）
  箇所をストロークの開始終了にする



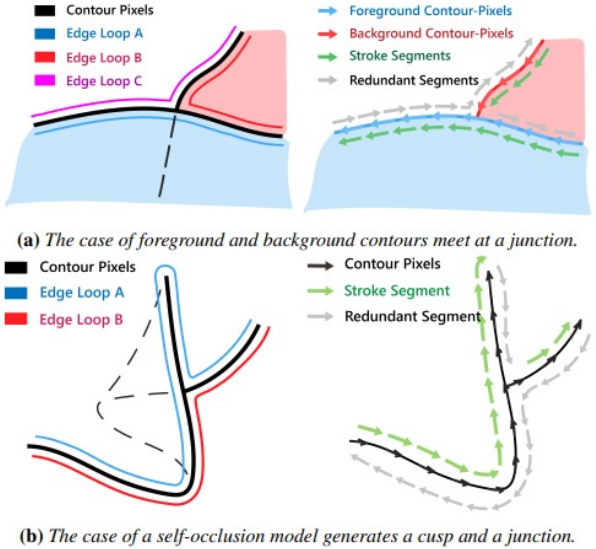
黒の線をベクトル化したいのにラスタライズの内外にそれぞれループが得られている

オリエンテーションカリング

輪郭線に対して二重になっている線から、描画するための一本のストロークを抽出する
入力：ピクセルエッジループ　　**出力**：ストロークセグメント*1

Orientation culling

- フィッティングで推定したループの向き（**接線ベクトル**）と輪郭ピクセルの向き（**エッジベクトル**）で不要な点を削除
- 注目点とその周囲数点で向きが一致するものを採用していく
- ちょうど可視性が変化する点でストロークが途切れる
- 論文が提案するヒューリスティックな方法



*1 ストロークごとにセグメント化（配列にデータの境界を表す情報が付随）された輪郭線データ＝ベクトルストロークセグメントの区切りと座標からなる一次元の配列で表現される

セグメンテーション

Orientation culling で抽出された点列はピクセルエッジループの一部
ループの途切れる位置（もしくはループの開始点）をセグメント区切りとみなして
Segmentated scan[*1] を行うと順序付きの点列が得られる
→**ベクトルストローク**

[*1] 入力配列がセグメント化されているときに、各セグメントごとに Prefix sum（シーケンスをスキャンして加算していく演算）を行う処理
StrokeGen 中の実装は Efficient Parallel Scan Algorithms for GPUs [Sengupta 2008]

input numbers	1	2	3	4	5	6	...
prefix sums	1	3	6	10	15	21	...

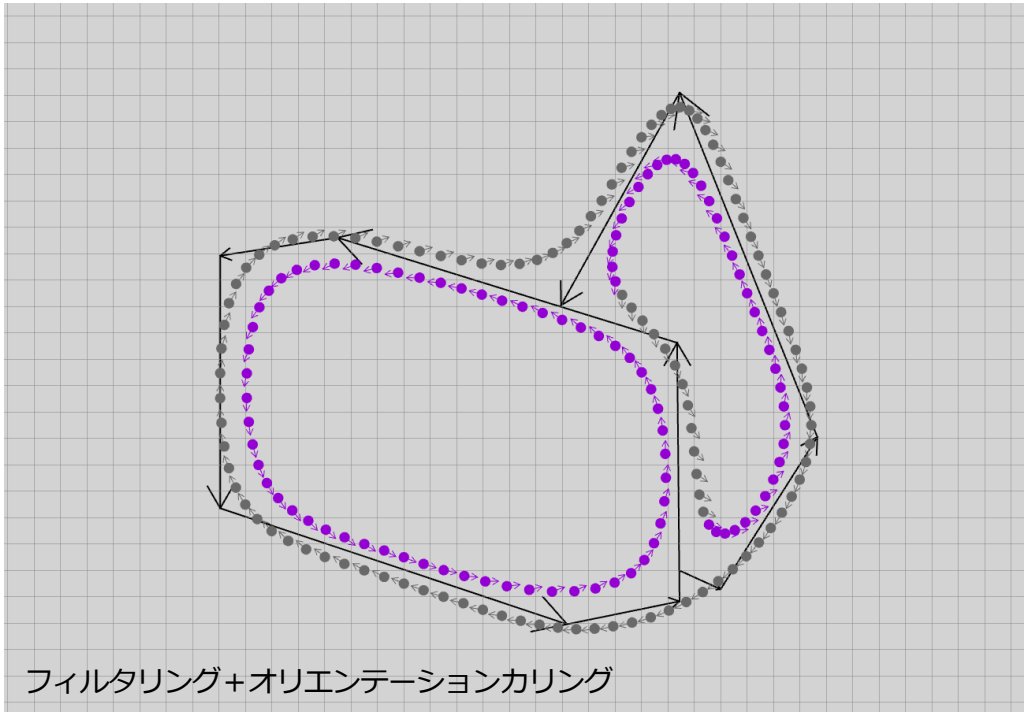
prefix sum: input numbers の値を先頭から順に加算した値

1	2	3	4	5	6	input
1	0	0	1	0	1	flag bits
1	3	6	4	9	6	segmented scan +

Segmented scan:
flag がセグメントの区切り
セグメント毎の Prefix sum

ベクトルストロークの生成

Silicon Studio
Ideas × Art × Technology



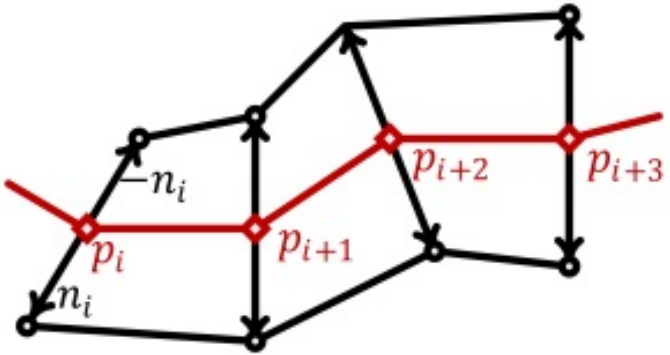
ストロークレンダリング

ここまでで得られたベクトルストロークは繋がりを持った点列

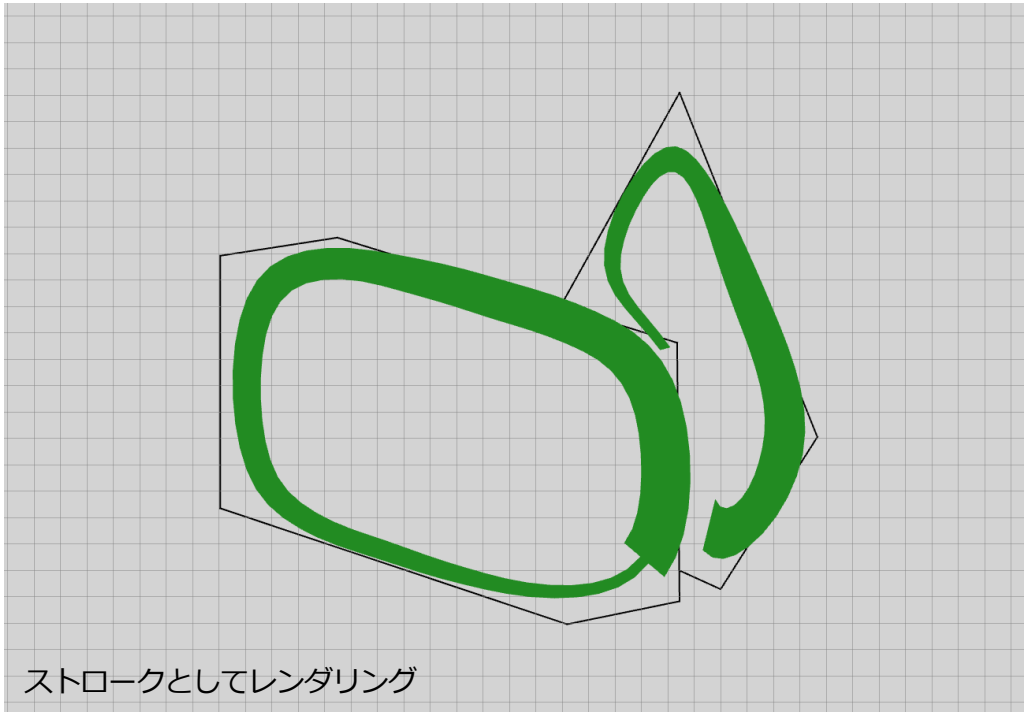
- 描画の方法は**任意に実装可能**

論文では点列から**ポリメッシュ**を生成してパス描画

- 接線→法線として法線方向への押し出し量を線幅パラメータとして用意
- UVは、押し出しした上下の位置からV、
  ストローク頂点の番号とストローク長からUを計算



ストロークレンダリング



ストロークとしてレンダリング

4 StrokeGen 実装解説

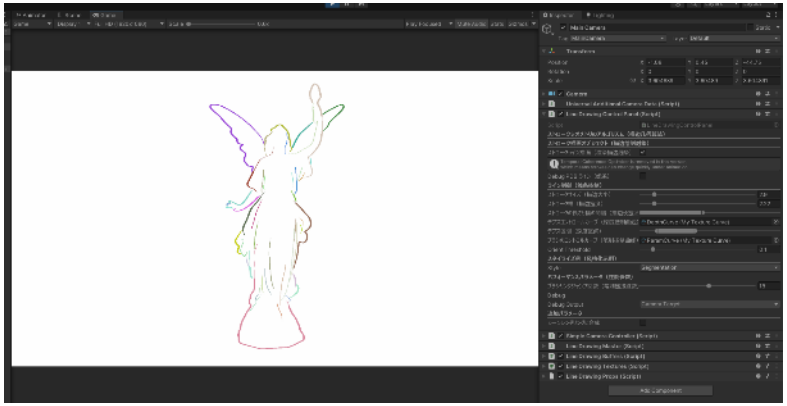
Silicon Studio
Ideas × Art × Technology

仕様

Silicon Studio
Ideas × Art × Technology

StrokeGen

https://github.com/JiangWZW/Realtime-GPU-Contour-Curves-from-3D-Mesh



- Unity Editor 2021.2.11f1
- DirectX 12 & HLSL Shader Model 6.0*1
- Universal RP 12.1.4
- Third Party: Odin Inspector

使い方

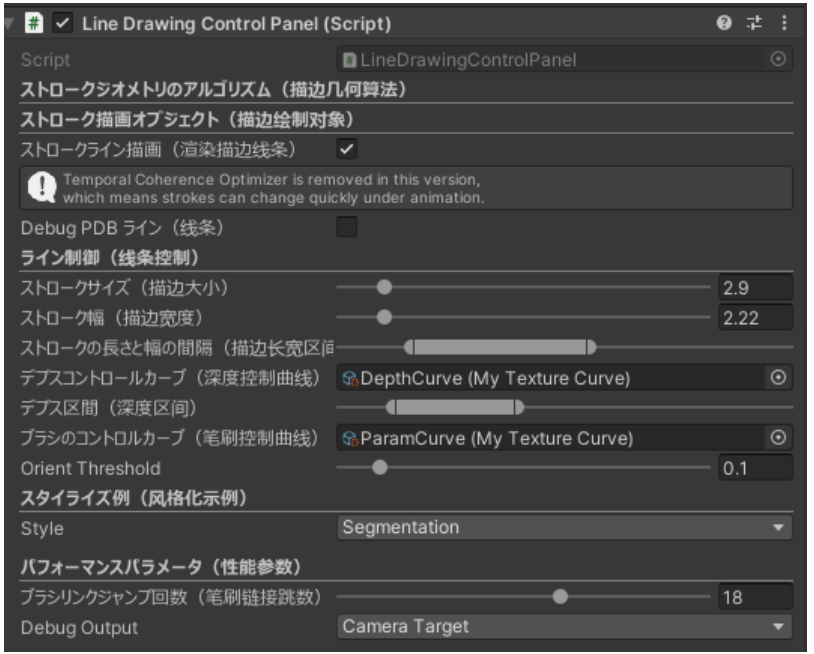
- カメラに **Line Drawing Controle Panel**、
  輪郭を描画したいオブジェクトに **Line Drawing Object**、
  Render Feature に **Contour Processor Feature** をアサイン
  (公開されているプロジェクトは以上設定済み)
- Game を Play すると輪郭線が描画される

*1 Segmented scan の並列処理に Wave Intrinsics を利用している

Silicon Studio
Ideas × Art × Technology

Line Drawing Controle Panel

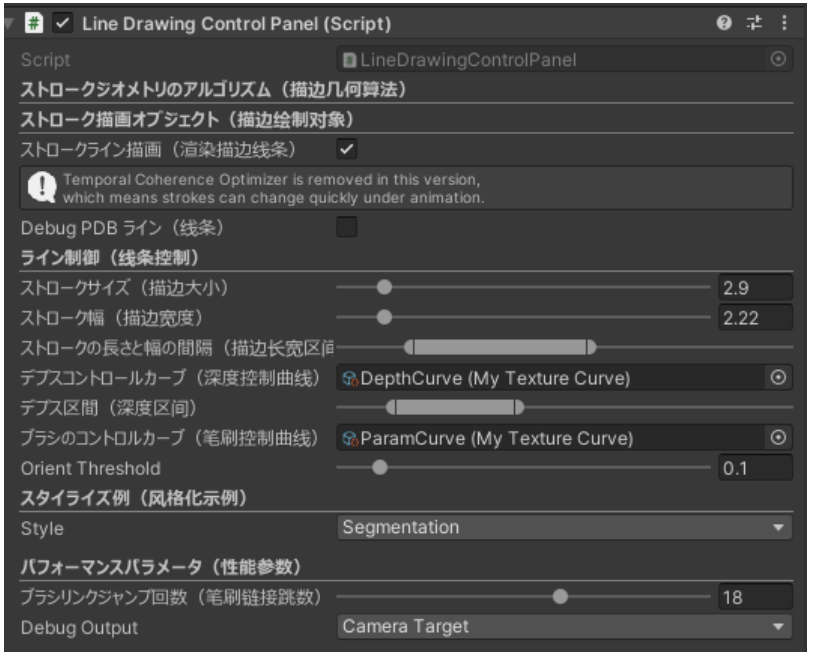
ストロークライン描画：ストロークを描画を有効化
Debug PDB ライン：
　（利用不可）Temporal coherence 最適化を有効化
ストロークサイズ：ストロークの大きさにかかるスケール係数
ストローク幅：ストロークの幅（画面水平方向）
ストロークの長さと幅の間隔：
　ストロークの最小最大のスケール係数
　実装では最小しか使われていない模様
デプスコントロールカーブ：
　深度に応じてスケール（線幅）を調整するためのカーブ
デプス区間：
　デプスコントロールカーブを評価する際の深度の最大最小値
ブラシのコントロールカーブ：
　ストロークのスケールを調整するためのカーブ
　ストロークの位置でカーブを評価して入り抜きを調整できる
Orient Threshold：
　オリエンテーションカリング判定を調整する閾値



Silicon Studio
Ideas × Art × Technology

Line Drawing Controle Panel

Style：描画スタイル
　Segmentation：ストロークのセグメントを可視化
　UV：ストロークのUVを可視化
　Textured：ストロークをブラシテクスチャで描画
ブラシテクスチャ：
　ストロークの描画に使うブラシテクスチャ
Brush Count：
　ブラシの番号で乱数を変えてストロークのオフセットが変化する
　多重にブラシを重ねる機能ではなさそう
Brush Stretching：
　ストロークに沿ったブラシテクスチャの伸縮の調整
ブラシリンクジャンプ回数：
　Pointer jumping のイテレーション回数
Debug Output：デバッグ表示
　Camera Target：最終的な描画結果
　Debug Texture 0,1：
　　シェーダ中で該当するマクロを有効化すると
　　デバッグ情報を可視化できる
　Contour GBuffer：輪郭線のジオメトリアトリビュート可視化



応用実装

以上が論文のアルゴリズムの内容
実際に StrokeGen のプロジェクトを動かし、いくつかの改造を行ったのでその紹介

1. Odin Inspector 依存の解消
2. アニメーションへの対応
3. メッシュ毎のパラメータ
4. Crease Edge への対応
5. スタンプ描画
6. ストロークレンダリング表現の追加

Odin Inspector 依存の解消

StrokeGen は Odin Inspector を利用している
Odin Inspector and Serializer | Improve your workflow in Unity
UI (Inspector) 拡張の有料プラグイン

パラメータの注釈等に活用しているだけでアルゴリズム自体に関わるものではない

主に C# Attribute を使った Custom Inspector の機能を利用しているので、
該当箇所を消していけばパラメータ調整が面倒にはなるが動作するようになる
今回は Odin Inspector を模倣した自作の Custom Inspector を実装した

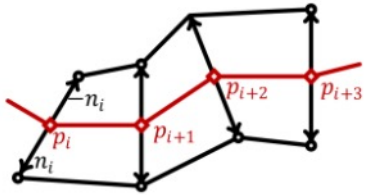
```
// [Title(“描边几何算法”)] // Odin
[MyTitle(“ストロークジオメトリのアルゴリズム”)] // 自作
public (int value, int propid) RenderPass =>
(
    ContourRenderingPass.RenderBrushPathPass,
    Shader.PropertyToID("_RenderMode")
);
```

スタンプ描画

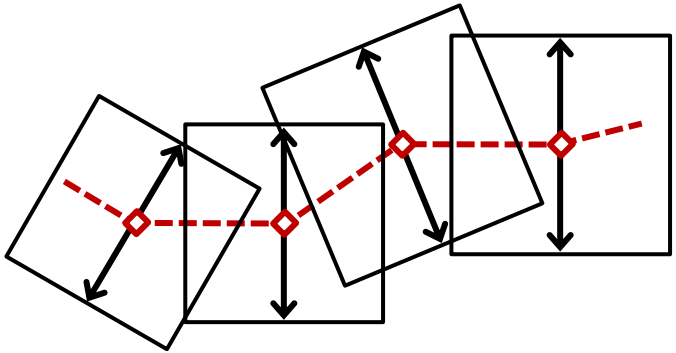
ストロークをイラストソフトにあるようなスタンプブラシへ対応
（公開されている実装に一部やろうとしているコードがあったのでそれを修正）

点列からクアッドメッシュを生成して描画

- 線幅パラメータがそのままクアッドのサイズ
  - 幅とは別パラメータでストレッチしたクアッドも描画可能
- UVは四角形上にそのまま定義される
- ストロークの方向に合わせてスタンプを回転するかどうかのレーキ設定も用意



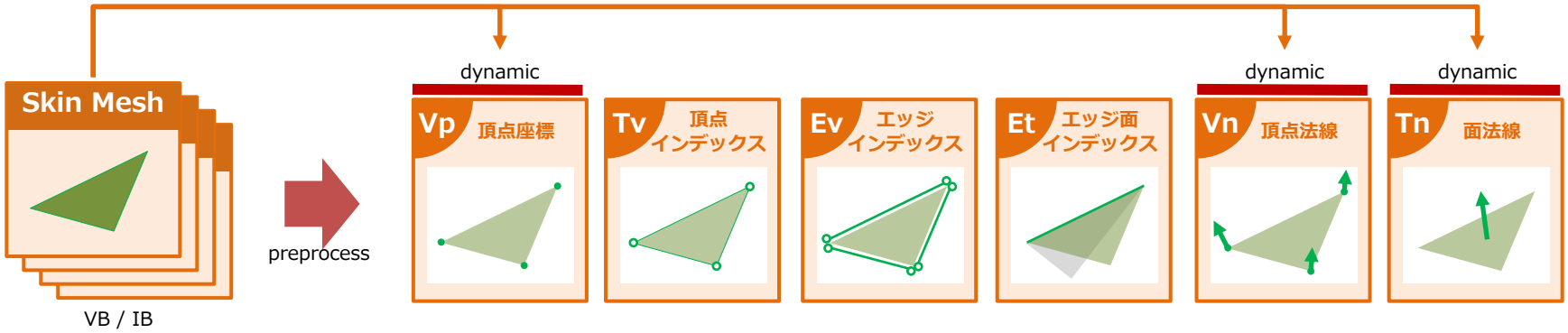
元々あったパス描画



アニメーションへの対応

公開されている StrokeGen はシングルスタティックメッシュのみサポート
アルゴリズム上の制限ではないのでキャラクターアニメーションに対応するように改造

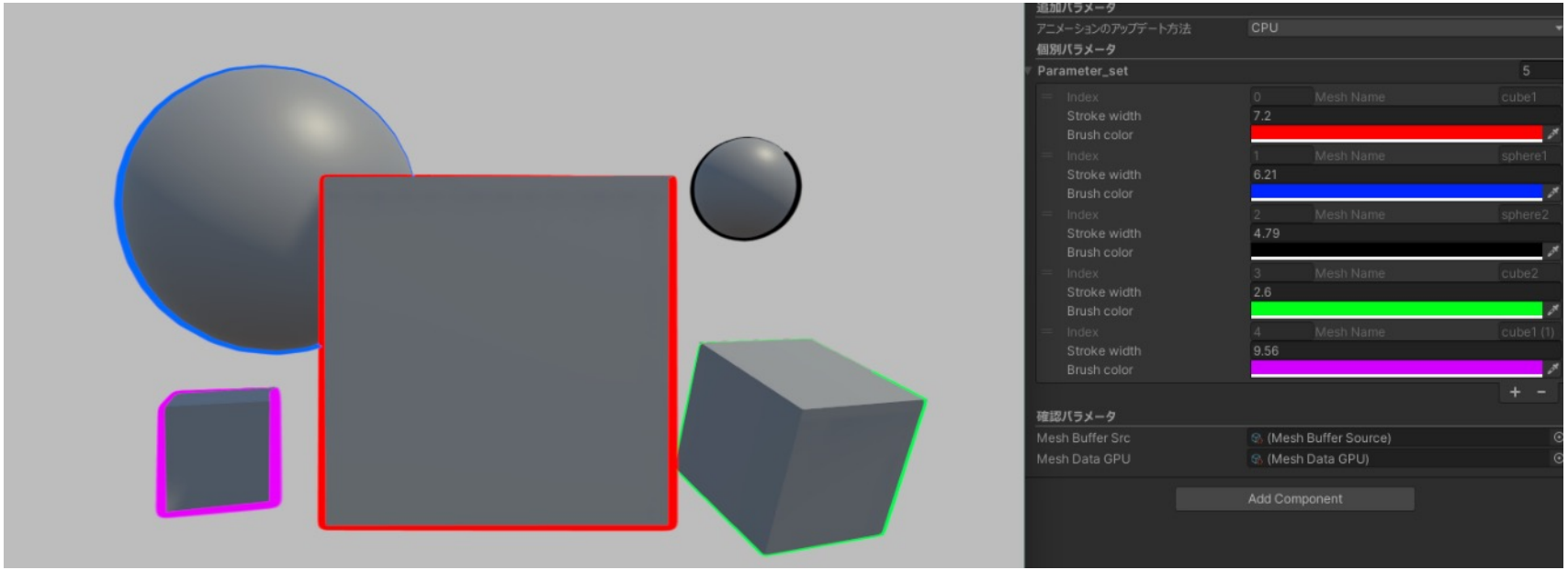
- 親オブジェクトを指定すると以下の子メッシュも全て対象にする
  - Preprocess で複数メッシュから各種データを収集
  - 全てのメッシュのデータは**一つの大きなバッファ**に順次格納
- SkinnedMeshRenderer の VertexBuffer から情報を動的に ComputeShader で更新
  - トポロジーが変わるようなアニメーションでなければ、頂点座標と法線のみ更新



メッシュ毎のパラメータ

複数メッシュ対応に併せて、メッシュ毎に異なるパラメータを適用できるように改造

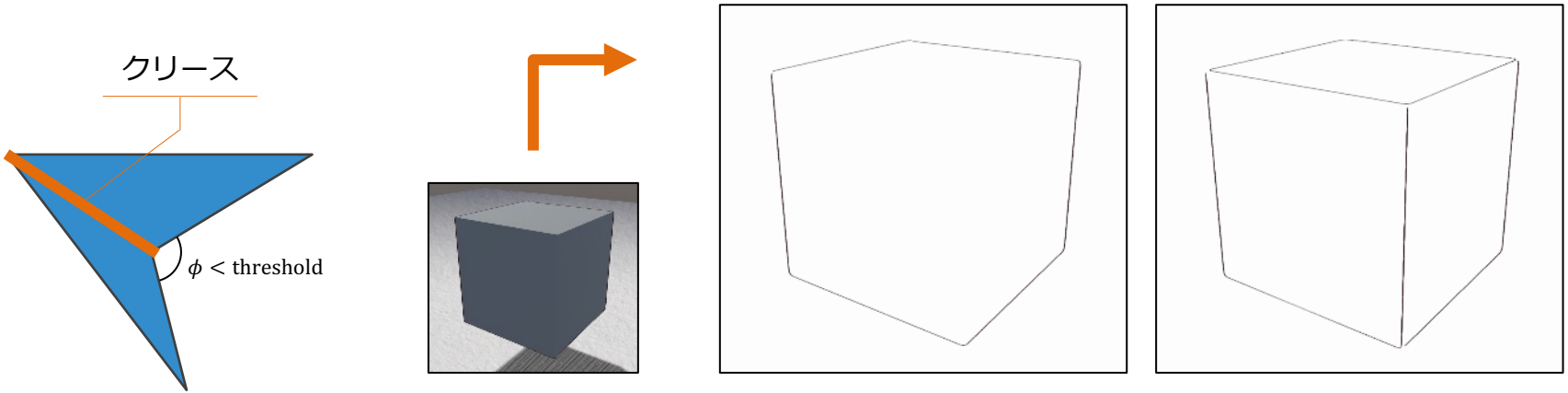
ラスタライズ時の属性情報にメッシュの ID を付与して、
ストロークレンダリング時にメッシュの ID から対応するパラメータにアクセスする



Crease Edge への対応

StrokeGen の輪郭線の定義にはクリースエッジが含まれない

Crease Edge
オブジェクトの内側における折り目になっている線
一つの辺を共有する二つの面のなす角度が閾値以下のとき Crease とする
閾値はユーザーパラメータとして用意
対象となるエッジが決まれば輪郭エッジに混ぜて処理できる
同じようにマテリアル境界のエッジを対象とすることで境界線描画にも対応



ストロークレンダリング表現の追加

ベクトルストロークならでは表現を作成（デモ・結果の中で紹介）

- ストロークへのテクスチャマッピング
- 色や線幅の制御
- 元の形状からズレた線の描画
- ストローク単体の表現
- 光源に応答するような表現
- 動きに応答するような表現

5 デモ・結果紹介

Silicon Studio
Ideas × Art × Technology

5 デモ・結果紹介

Silicon Studio
Ideas × Art × Technology

利用アセットについて

- ユニティちゃん（© Unity Technologies Japan/UCL）
  UNITY-CHAN! OFFICIAL WEBSITE
- ミライ小町（法人による学術発表・技術研究用途）
  ミライ小町 Mirai Komachi Official Page

結果紹介

Silicon Studio
Ideas × Art × Technology

StrokeGen の結果（デフォルト）

結果紹介

Silicon Studio
Ideas × Art × Technology

StrokeGen の結果（アニメーション）

結果紹介

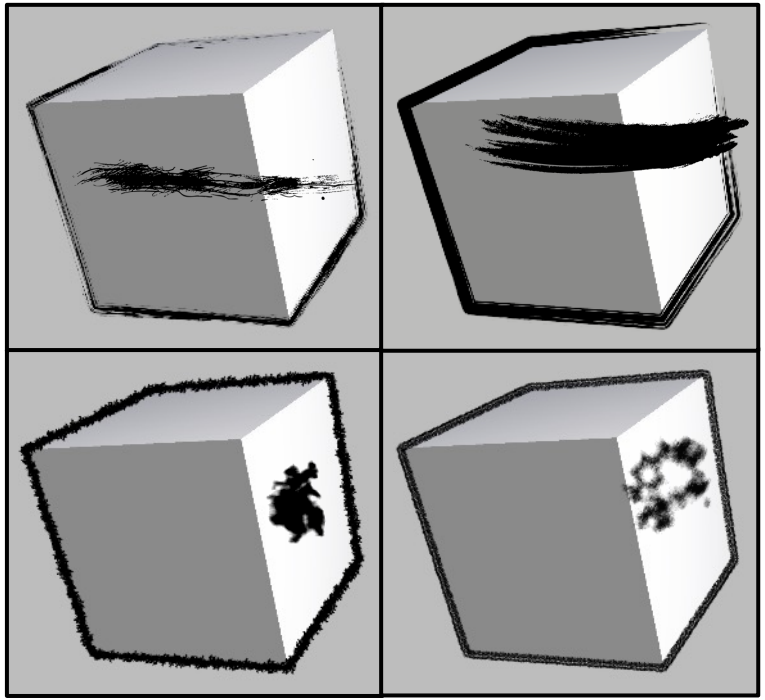
Silicon Studio
Ideas × Art × Technology

StrokeGen の結果（アニメーション）+改良

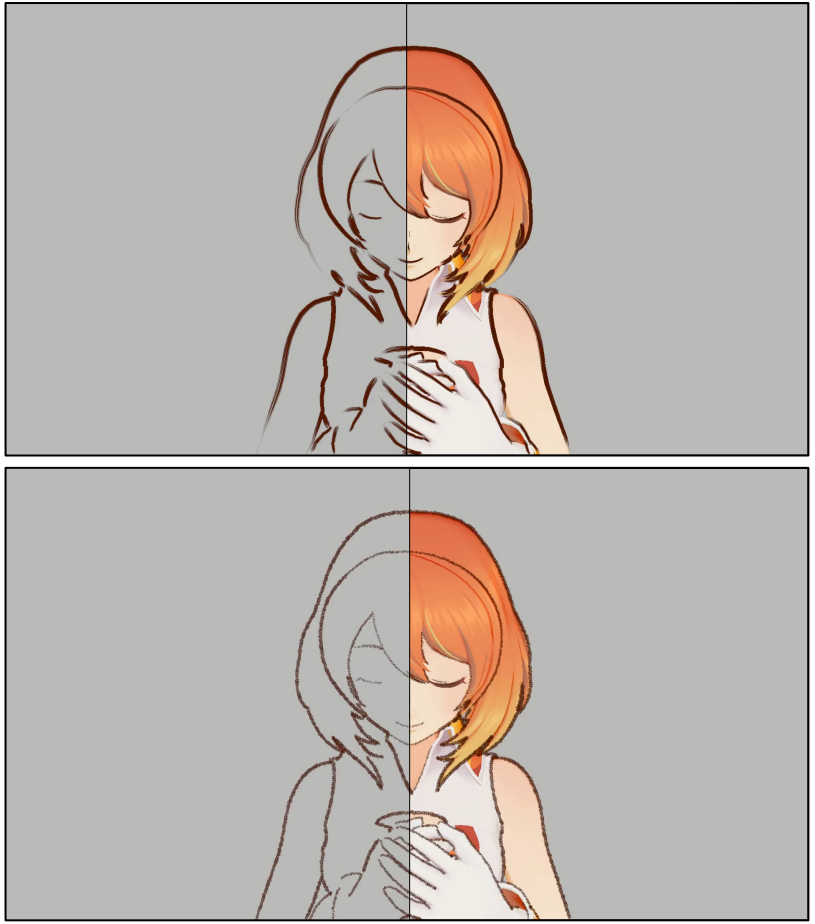
線の対象をマテリアルの境界に変えるなどの調整で安定性を向上させるテスト
元の StrokeGen の輪郭線の定義ではなくまだ検証段階

結果紹介

パス・スタンプテクスチャマッピング

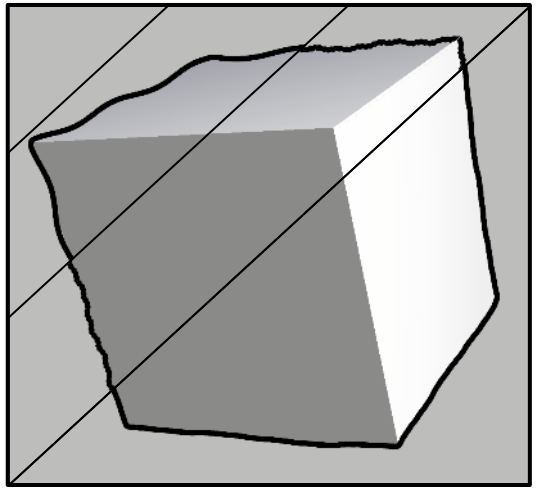


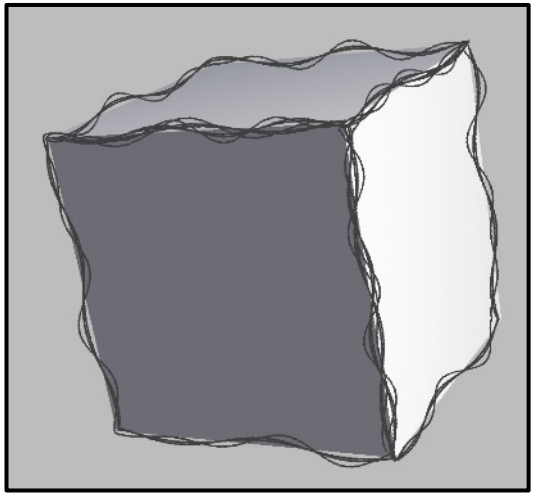
マテリアルの境界のエッジにテクスチャマッピング
パス（上）とスタンプ（下）



結果紹介

線のゆらぎ・多重線

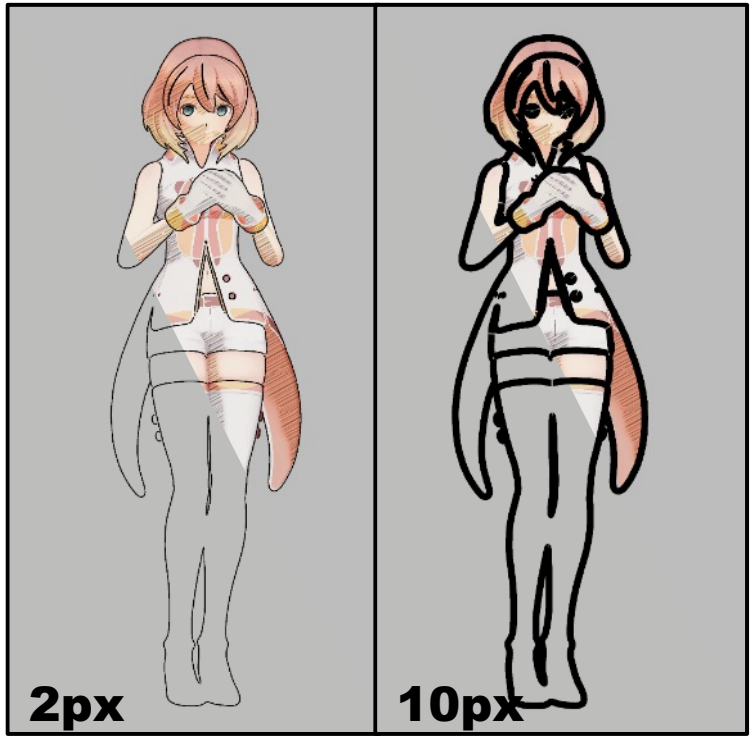




線のゆらぎやラフスケッチ風の表現
細かい形状の制御が難しくアニメーションに適用したときのちらつきも激しいが、
ゲーム輪郭線の従来法には難しい元々の形状を逸脱した線が容易に得られる

結果紹介

ピクセルレベルの線幅制御・太い線

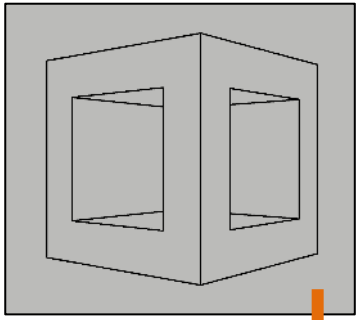


従来法（特にポストプロセス）が苦手な太い線も
ピクセルレベルで調節可能
太いストロークのデプス処理が甘いので
自身のメッシュとの判定によるちらつきが多少ある

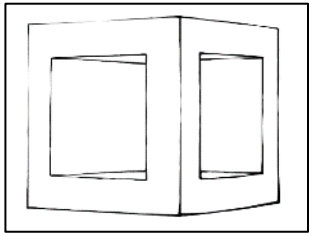
結果紹介

Silicon Studio
Ideas × Art × Technology

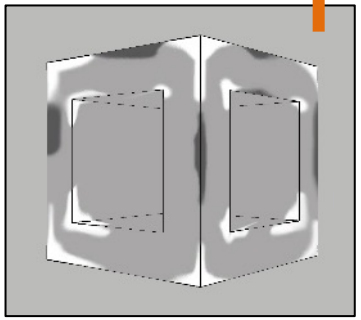
線幅による線画の味付け



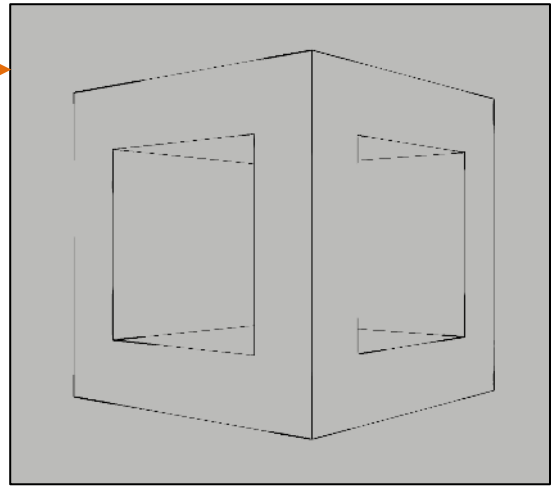
均一な線

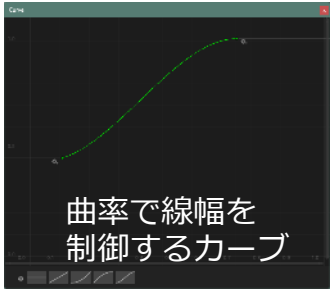


参考手書き



線幅マップ

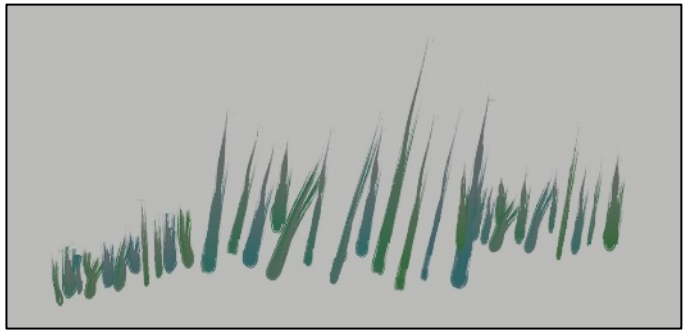




線幅を制御するテクスチャや曲率に基づいて
イラストのような線を演出
曲率は複雑な形状で潰れやすく制御が難しい
テクスチャ（や頂点カラー）での線幅制御は従来法
でもよく用いられるため技術を転用しやすい

結果紹介

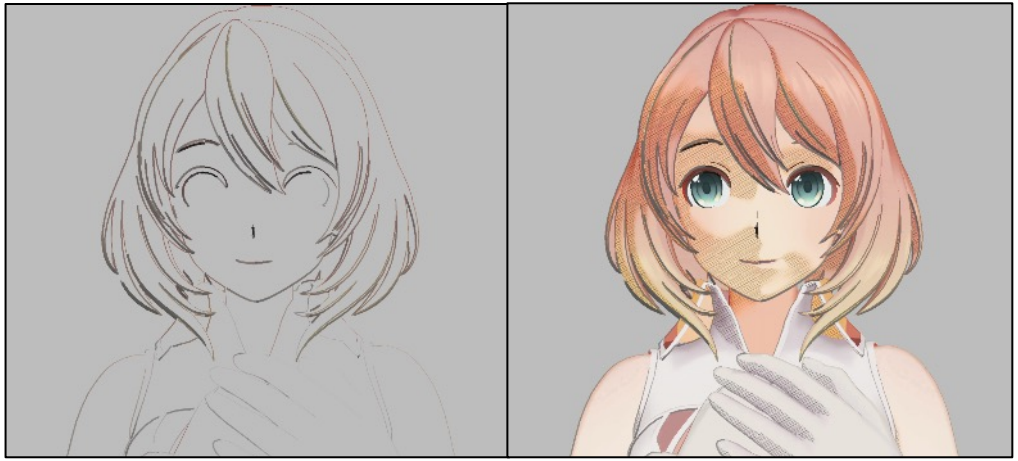
アウトラインのみで描く表現

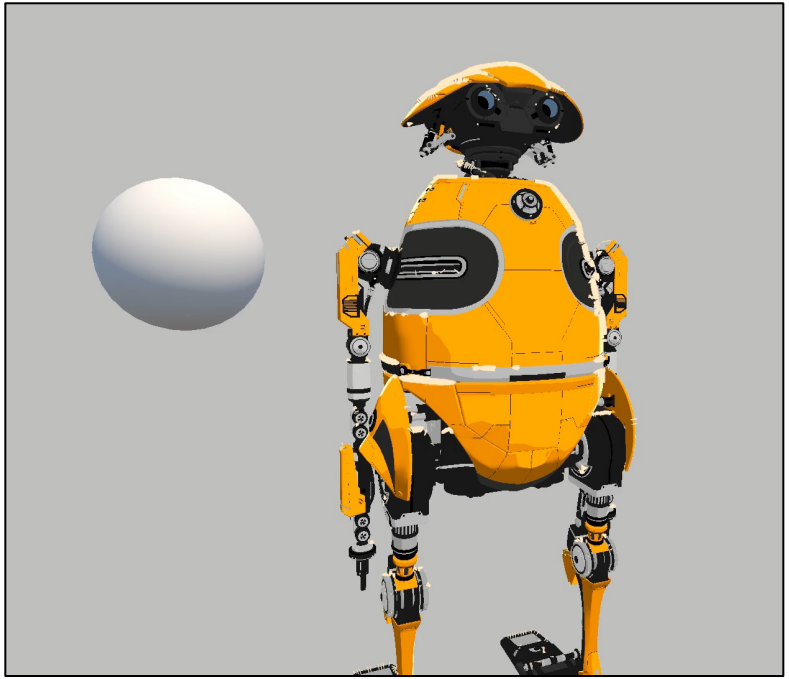


草むらや髪の毛の房をストロークレンダリングで表現
現状の StrokeGen は一本の線を描くのが苦手なので、無理矢理板ポリを置くような感じで実装
線をジオメトリやテクスチャなどの配置物で表現する従来法にストローク表現を統合できる可能性がある

結果紹介

光源に応答する表現

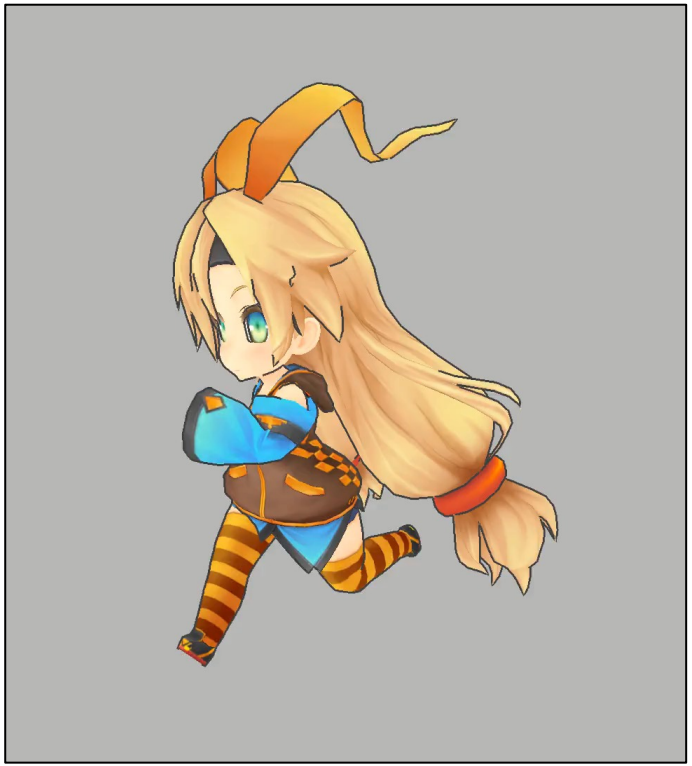


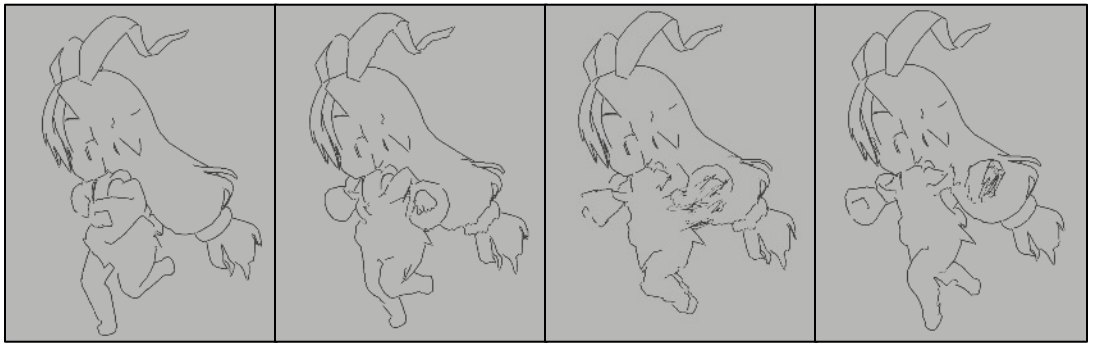


線にライティングの効果を施す演出
（左2つ）画面右上に光源を置き、光が当たるところは色味があり線が細くなる演出
（右）リムライトをストロークで描画

結果紹介

モーションに応答する表現





モーションベクトルに基づいてストロークにゆがみ効果を与えると、いわゆる**おばけブラー（Smear frame）**の表現ができる
複雑な形状では綺麗に出にくく、位置や形状の制御も難しいが従来法では困難な表現

結果紹介

ベクトルストロークならでは表現を作成

- ストロークへのテクスチャマッピング
- 色や線幅の制御
- 元の形状からズレた線の描画
- ストローク単体の表現
- 光源に応答するような表現
- 動きに応答するような表現

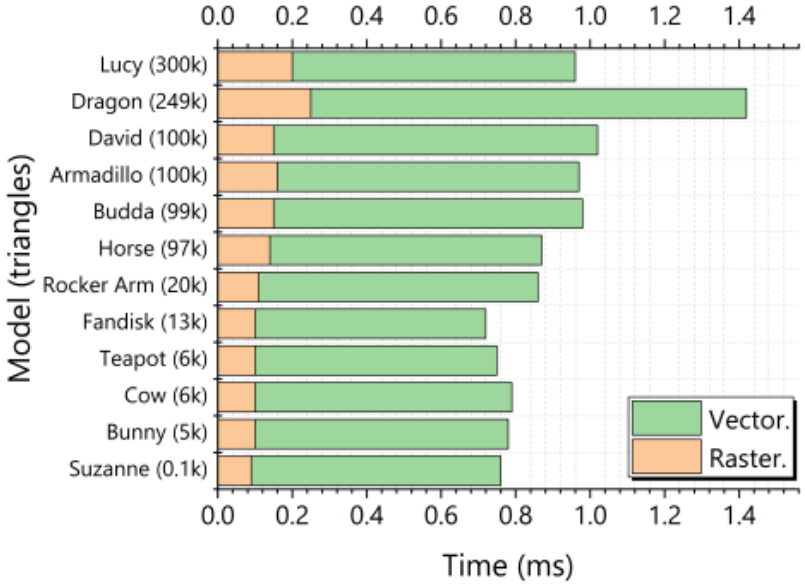
従来法でも工夫次第で可能な表現もあるが、
ベクトルストロークであることで可能な表現が（今回実装した以外にも）多彩にある
線を線として**独立したパラメータ群**として実装・調整できるのも利点

StrokeGen の本質はストローク生成だが実用化を考えると**どんな表現ができるか**が重要
今回はアーティストを挟んでいないのでまだまだ可能性を秘めている

パフォーマンス

Silicon Studio
Ideas × Art × Technology

論文中のパフォーマンス評価



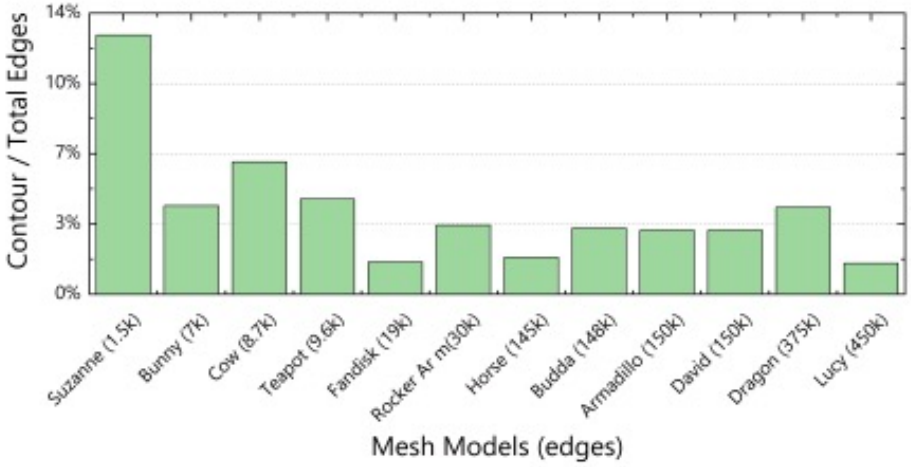
Intel i7-7700HQ 2.8 GHz of 8 GB RAM
NVIDIA GTX 1070
1980x1080

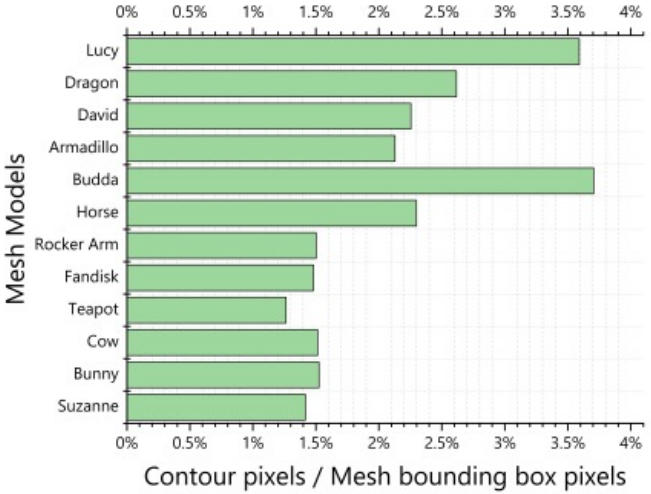
Rastarization と Vectorization+StrokeGeneration に分けてパフォーマンスを計測
Rastarization は複雑な形状（輪郭線が多くなる）やメッシュ自体の大きさに左右される
Vectorization+StrokeGeneration の方が重い
全体を通してリアルタイムレンダリングのバジェットに収まる程度

パフォーマンス

Silicon Studio
Ideas × Art × Technology

論文中のパフォーマンス評価



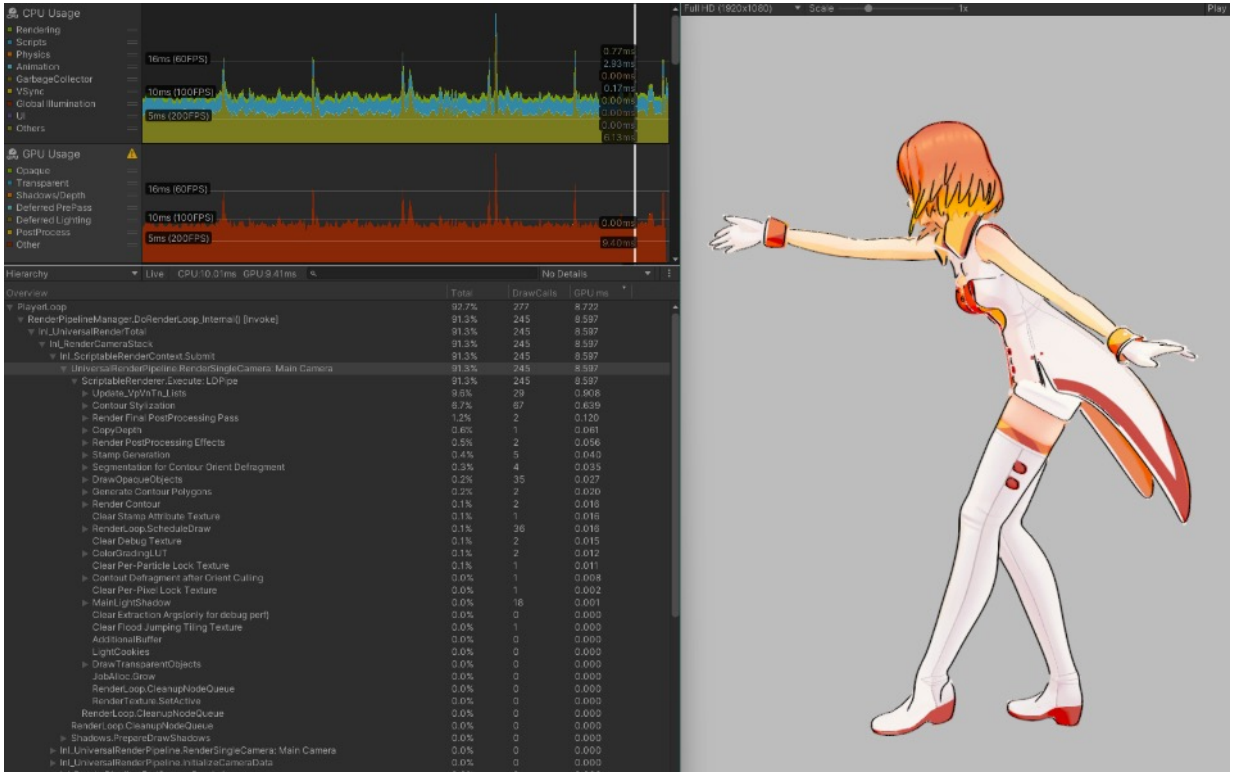


アルゴリズムを解説してきた通り処理は非常に多く本当にリアルタイムに適しているのか怪しく感じるが、
輪郭の疎性によるパフォーマンスのメリットが大きいと論文中で説明されている
メッシュ全体のエッジの内、輪郭エッジはたかだか数%で、
最終的に処理の中心になる輪郭ピクセルはレンダリング画素中たった4%程度しかない

パフォーマンス

Silicon Studio
Ideas × Art × Technology

実際に動かしてみたパフォーマンス



AMD Ryzen 7 4800H 2.9 GHz of 16 GB RAM
NVIDIA RTX 2060 (Laptop)
1980x1080

Update_VpVnTn_List (0.91ms):
- Preprocess (アニメーション対応)

Contour Stylization (0.64ms):
- Vectorization + Stroke Generation

Stamp Generation (0.04ms):
- Rastarization

Segmentation for Contour Orient Defragment (0.04ms):
- Stroke Generation

Generate Contour Polygons (0.02ms):
- Stylization

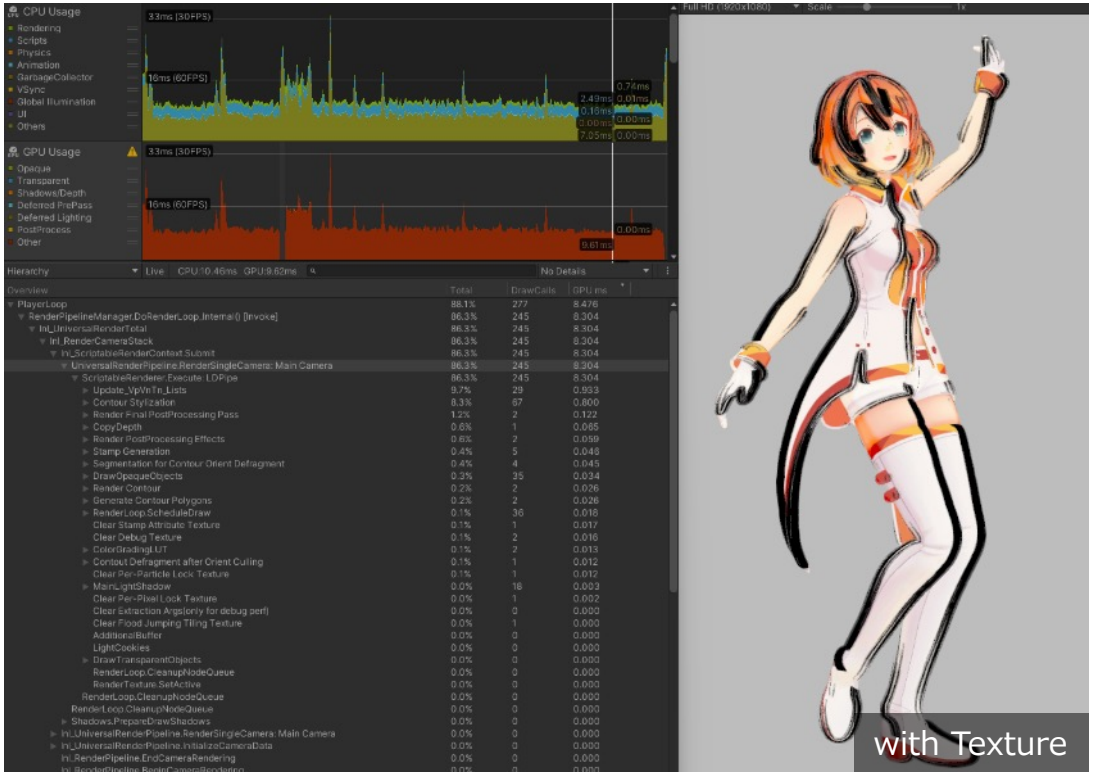
Contour Defragment after Oriennt Culling (0.01ms):
- Stroke Generation

Total 1.66ms

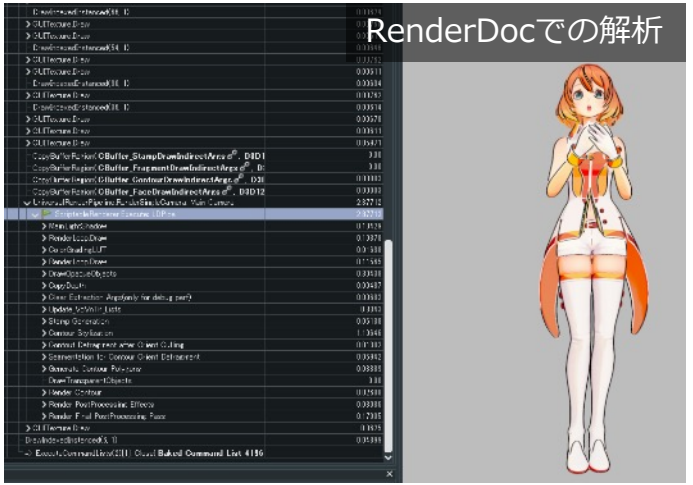
パフォーマンス

Silicon Studio
Ideas × Art × Technology

実際に動かしてみたパフォーマンス



AMD Ryzen 7 4800H 2.9 GHz
of 16 GB RAM
NVIDIA RTX 2060 (Laptop)
1980x1080



パフォーマンス考察

- 全体を通してリアルタイムレンダリングの処理として十分な速度が得られている
  必要なエッジのみに絞って処理を行うこと、**Indirect 命令**で動的に処理量を制御していること
  などが貢献している

- アニメーション対応の最適化が不十分
  元々のシングルスタティックメッシュの実装をあまり破壊しないように組み込むために、
  複数メッシュを**一つの大きなバッファ**で管理するようにしたことがよくないと考えられる

- ストロークの太さやテクスチャマッピングよるパフォーマンスへの影響はあまり見られない

- プロファイラには現れないが**事前処理のせいで起動が遅い**（Play から動き出すまで数秒）
  事前処理結果を保存して使い回せる仕組みが必要

- もっと**複雑で規模の大きなシーン**でどのような結果になるかはさらなる検証が必要
  手法自体を大規模なシーンへ適用できるように拡張する必要がある

6 課題と今後の展望

Silicon Studio
Ideas × Art × Technology

1 輪郭線とは？

2 StrokeGen 概要

3 StrokeGen アルゴリズム解説

4 StrokeGen 実装解説

5 デモ・結果紹介

6 課題と今後の展望

6 課題と今後の展望

Silicon Studio
Ideas × Art × Technology

StrokeGen はゲーム NPR における新しい手法となり得るのか？

従来のリアルタイム手法では為し得なかった表現が可能になり非常に**期待が持てる**手法
但し、実用化には課題も多いのでそのまま**従来の手法に代替するものではない**

特に時間的な安定性は大きな課題
StrokeGen **そのものの安定性**の課題と**スタイライズ表現の時間的一貫性**の課題がある

大規模なシーンへの適用はまだ達成していないため、
キャラクターと背景で**従来法と共存して使い分ける**ような仕組みを考えなければならない

0フレーム

1フレーム

2フレーム

StrokeGen の時間的安定性

Silicon Studio
Ideas × Art × Technology

StrokeGen は時間的な安定性が低い（僅かな形状の変化で結果が変化しやすい）

StrokeGen の採用しているアルゴリズムの中核技術である **Potrace** が一つの要因
Potrace は二値領域の**境界線をベクトル化**するアルゴリズム

StrokeGen ではラスタライズしたエッジを Potrace の入力にしており、
輪郭線の境界線をベクトル化することで、**二重のベクトル輪郭線候補**を得ている
最終的なストロークを決定するための Orientation Culling が不安定

解決案
境界線ではなく線を**線としてベクトル化**する手法に置き換える
Orientation Culling を使わずに**二重の境界線を解消**する

スタイライズ表現の時間的一貫性

Silicon Studio
Ideas × Art × Technology

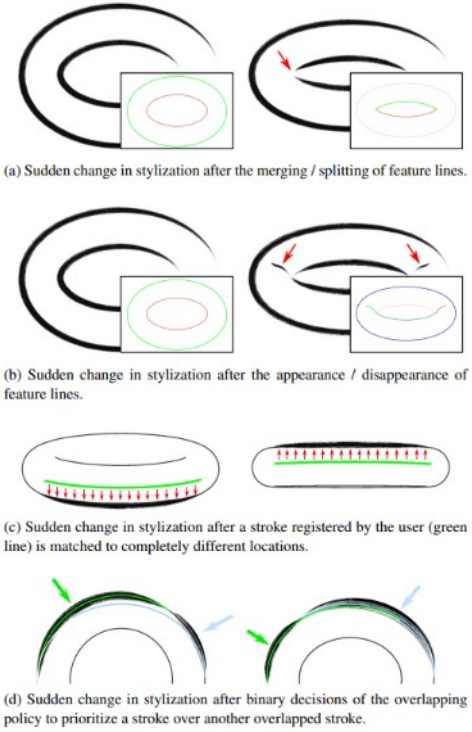
輪郭線は簡単に**トポロジーが変化**する
スタイライズ表現を時間的に一貫して描くことは非常に困難

均一な線やオブジェクトに固定された線の演出*1 ならば問題にはなりにくいが、
入り抜きの位置やテクスチャマッピングの箇所が時間で適当に動いてチラついてしまう

StrokeGen における線の開始位置は
可視性が変化する場所*2**か****ループ中の適当な位置***3

解決案
フレーム間で線の変化を追跡して線の見た目の激しい変化を抑制する
線の Temporal Coherence に関する研究はいくつもあり、
StrokeGen への適用を調査する必要がある

*1 頂点カラーやテクスチャに線の強弱の情報を埋め込むような手法を指す
*2 線が途切れたり重なったりする場所
*3 Loop breaking におけるモートンコードで決まる



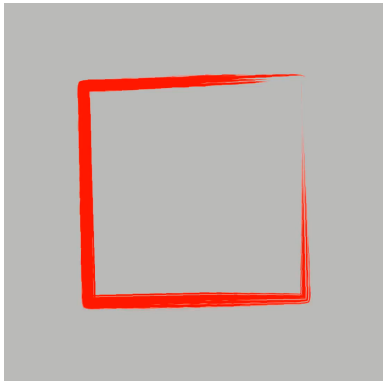
Temporally Coherent and Artistically Intended Stylization of Feature Lines Extracted from 3D Models [Cardona 2016]

リアルタイムならではの時間的な課題

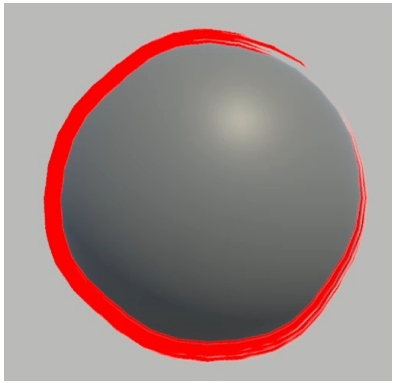
StrokeGen は前述の手法自体の持つ不安定さに加え、
リアルタイム用途であるため**時間的に細かい調整ができない**課題がある

ゲーム用途では**自由なカメラ**に対して、**意図した表現が安定して**得られなければならない
細かすぎない粒度で**アーティストの要求を反映する仕組み**が必要
（細かい演出が可能なカットシーンであってもフレーム単位の調整などはしたくない）

どのような**表現**を、どの程度の**自由度**で、どのような**方法**で調整できるべきか、
ツール的なアプローチも重要になってくる



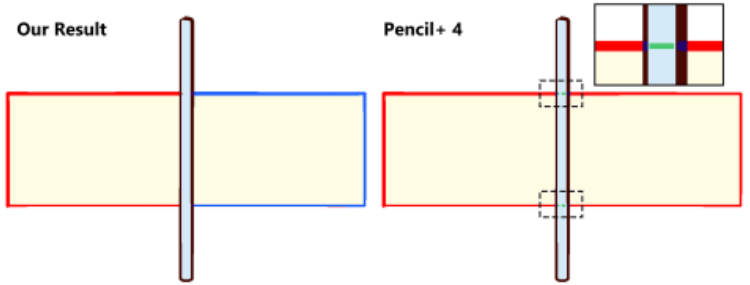
ストロークの角度によって
分断を行うことで、
入り抜きの位置を固定しや
すくする例



マウスポインタの位置を
ガイドに入り抜きの位置
を制御する例

論文中の指摘

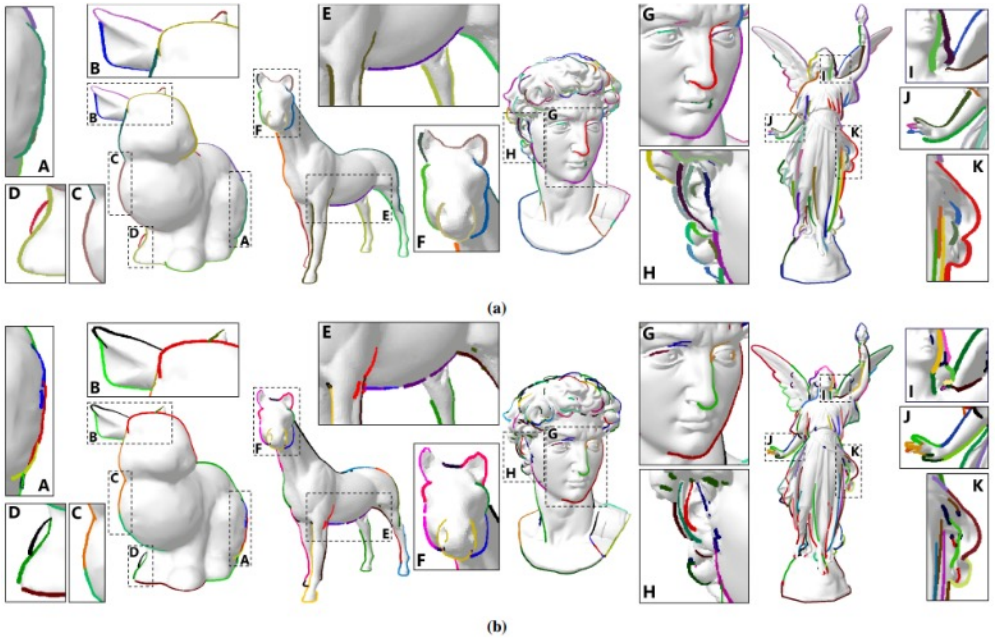
Silicon Studio



ベクトル化処理がスクリーンスペースであるため、
実際は繋がっていない細かい輪郭線が繋がったり、
見えない部分で繋がっている線が途切れてしまう

StrokeGen

Pencil+4



6.2 今後の展望

現在はどのように実用化できるかを研究開発中
本公演の動画・SS にも StrokeGen のアイディアをベースに再構築した新しい手法の結果が含まれる

アルゴリズム面の改善
- 安定性の**大幅な**向上
- 従来法との共存・置換
- 更なる高速化
  （大規模シーンへの適用）

表現面の改善
- アップスケーリング
- アンチエイリアシング
- **ニーズの調査** → 表現の拡充
- アートをサポートするための
  ツールとしての機能拡充

論文の著者もさらに発展させた手法を開発している模様
（著者のXより。Blender上で動作させているようなのでゲーム用途からは離れているかも？）

とても面白い技術なので今後の発展の動向に要注目！

Ideas × Art × Technology

技術力・表現力・発想力を兼ね備えたCGソリューションプロバイダー

ブランドステートメント

私たちシリコンスタジオは、自社開発による数々のミドルウェアを有し、CGの黎明期から今日に至るまでCG関連事業に取り組み、技術力（Technology）、表現力（Art）、発想力（Ideas）の研鑽を積み重ねてきました。これら3つの力を高い次元で融合させ、CGが持つ可能性を最大限に発揮させられること、それが私たちの強みです。

コンピューターグラフィックス（CG）は無限の可能性を秘めています。
映像・エンターテインメント分野では、AI活用やリアルタイムレイトレーシング、グローバルイルミネーションなどの革新的な技術とデバイスの進化が表現の幅を拡げ、美しくリアリティ溢れる描画が深い臨場感をもたらしています。
一方、自動車、製造、土木・建築といった産業分野では、デジタルトランスフォーメーション（DX）への取り組みが進む中、自動運転や製品検査における機械学習向け教師データやシミュレーター開発、デジタルツインなどにおいてCG・ゲームエンジンが活用され、成果を出し始めているところです。
また、5Gのような高速大容量で低遅延を実現するネットワーク環境やクラウドの活用は、ゲーム、SNS、動画配信との融合や数千人単位でのマルチプレイ、VR・AR・MRといったxRやメタバースをはじめとした新たなコミュニケーション手段を作り出し、今後もユーザーエクスペリエンス（UX）にさらなる変革をもたらすでしょう。

私たちはCG業界をリードするソリューションプロバイダーとして、技術力・表現力・発想力でこの変革に挑戦し続けることにより、ゲーム・エンターテインメント、非ゲーム・エンターテインメントの産業分野を問わず、お客さまの課題解決はもちろん、付加価値のあるアウトプットの提供をお約束いたします。

Silicon Studio
Ideas × Art × Technology

Ideas × Art × Technology

技術力・表現力・発想力を兼ね備えたCGソリューションプロバイダー