Vol.2022-CVIM-229 No.26
2022/3/10

Soft Rasterizer を用いた 2D to 3D Style Transfer

高石圭人†脇田玲†
慶応義塾大学

1. はじめに

近年、建築、プロダクトデザインなどのプロトタイピングの場面や、映像制作において 3DCG 技術を用いた制作が活発に行われている．映像表現を向上させる目的や、プロダクトのデザイン提案時に 3DCG オブジェクトの形状やテクスチャなどに関するデザインのバリエーションが求められる．このような状況に対して豊富なデザインのバリエーションを生成するにはクリエイターの技術力、工数がかかるという問題点がある．本研究では、3D モデルに対するバリエーション生成を最終的な目的とし、形状バリエーションの要因の1つとなるメッシュのサーフェイスデザインの多様性に注目する．

Gatys ら[1,2]は CNN（Convolutional Neural Network）を用いて、画像の形状情報であるコンテンツ情報と画風であるスタイル情報を分離させ、さまざまスタイルでセマンティックコンテンツをレンダリングする Neural Style Transfer (2D Image to 2D Image Style Transfer) を提案した．そして、その発展として微分可能レンダラーを用いることによって、図1のような画像のスタイル情報を3次元形状オブジェクトの表面形状やテクスチャへ転写する技術である，2D Image to 3D Mesh Style Transfer[3]に関する研究も行われている．

本研究では、微分可能レンダラーの１つである、Soft Rasterizer[5]を用いて、法線マップを媒介とすることによって 2D Image to 3D Mesh Style Transfer を実現し、サーフェイスデザインのバリエーション生成に対する考察を行う．

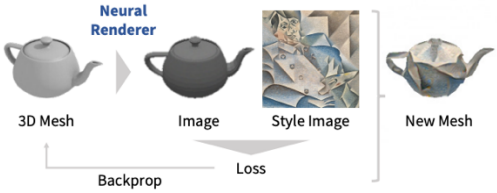

図 1 2D Image to 3D Mesh Style Transfer ([4]より引用)

2. 関連研究

2.1 2D Image to 2D Image Style Transfer

Gatys らによって提案された手法[1]のアーキテクチャを

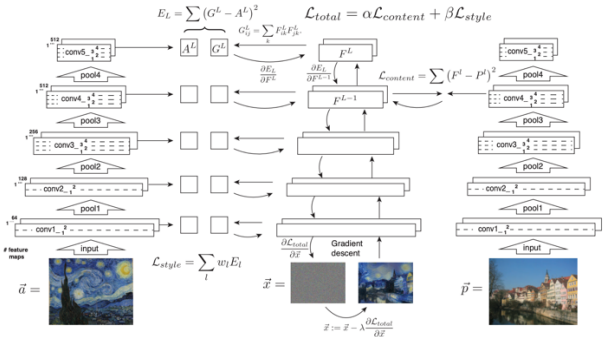

図 2 Neural Style Transfer のアーキテクチャ
([1]より引用)

図 2 に示す．Gatys らは学習済みの VGG[3]などの CNN を含むモデルによって得られた特徴マップを元に Content Loss と Style Loss を定義した．式 1 に示されているそれらの和に対して勾配降下法を用いて最小化を行う．Content Loss と Style Loss は$L_{content}$, L_{style}、それぞれの損失に対する係数はλ_c, λ_sによって示されている．

$$L_{total} = \lambda_c L_{content} + \lambda_s L_{style} \quad (1)$$

$L_{content}$は形状情報を保持するために設計された損失関数であり、式２のように二乗誤差を用いて定義されている．参照としているスタイル画像、コンテンツ画像、最適化対象である目的画像はそれぞれ $\vec{a}, \vec{p}, \vec{x}$、畳み込みの結果得られた特徴マップの次元数のチャンネル番号と、マップ内の座標は i, j、目的画像とコンテンツ画像の l レイヤー目の特徴マップは$F^l_{i,j}$, $P^l_{i,j}$によって示されている．

$$L(\vec{p}, \vec{x}, l)_{content} = \frac{1}{2} \sum_{i,j} (F^l_{i,j} - P^l_{i,j})^2 \quad (2)$$

L_{style}では特徴マップをそのまま用いるのではなく、式３のように特徴マップ間の内積であるグラム行列を用いる．ここでは i, j はチャンネル番号、k はマップ内の座標を示している．

$$G^l_{i,j} = \sum_k \left(F^l_{i,k} \cdot F^l_{j,k} \right) \quad (3)$$

このグラム行列を目的画像とスタイル画像の特徴マップに対して計算を行い、式 4 のように和を求めた値がL_{style}となる．α_nはレイヤーごとの重みや画素数などをまとめた

係数である．

$$L_{\text{style}} = \sum_n \sum_{i,j} \alpha_n \left(G^l_{i,k} - A^l_{j,k}\right)^2 \qquad (4)$$

2.2 微分可能レンダラー

2D Image to 3D Mesh Style Transfer では 2 次元の画像の勾配を 3 次元頂点へ流すために、微分可能レンダラーの 1 つである Kato ら[4]によって提案された Neural 3D Mesh Renderer が用いられている．Computer Graphics におけるレンダリング処理では、三角形ポリゴン内のピクセルを特定するラスタライズという離散的な処理が含まれる．これによって頂点座標でピクセル値を微分すると常に算出される値が0となり、勾配の逆伝搬を妨げるため、レンダリング処理をニューラルネットワークに統合することは難しい問題とされてきた．この問題に対して Neural 3D Mesh Renderer では逆伝搬時の勾配を近似する手法によってレンダリング処理をニューラルネットワークに統合することを実現した．Liu ら[5]によって提案された Soft Rasterizer では，Neural 3D Mesh Renderer のように逆伝搬に用いる勾配を近似するのではなく，レンダリング内のラスタライズ処理自体の近似を行っている．通常のレンダリングにはレンダリングを行うカメラから最も近い物体のみがレンダリングされるように各ピクセルに奥行に関する情報をバッファリングさせ，奥行き比較を実現させる Z-buffering という処理が存在する．しかし，この処理もラスタライズのように微分不可能な処理であるため，ニューラルネットワークに組み込むためには微分可能な表現にしなければならない．図 4 に示されている通り，Soft Rasterizer では微分不可能なラスタライズと Z-buffering を Probability Computation と Aggregate Function を用いることによって微分可能な表現を行っている．

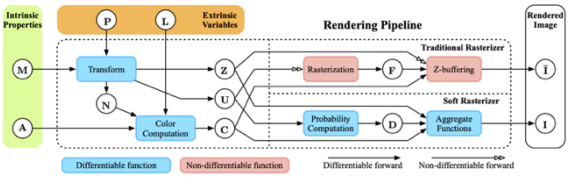

図 4 Soft Rasterizer のアーキテクチャ([4]より引用)

2.3 法線マップ

法線マップとはピクセルの構成要素である R, G, B に対して、View 空間（カメラを原点とした座標系）の法線ベクトルの要素 X, Y, Z を埋め込んだテクスチャであり、図 5 に RGB 画像と法線マップの対応例を示す．図 6 のように法線マップを用いることによって、本来凹凸が少なく、少ない頂点数の 3D モデルに対してでもレンダリング時に擬似的に複雑な法線情報を付与することが可能であるため、クオリティの高い陰影表現を行うことが出来る．そのため、ゲーム CG において処理負荷の軽量化にも用いられている．また、陰を見るとわかるように実際に頂点座標を変化させる技術ではない．

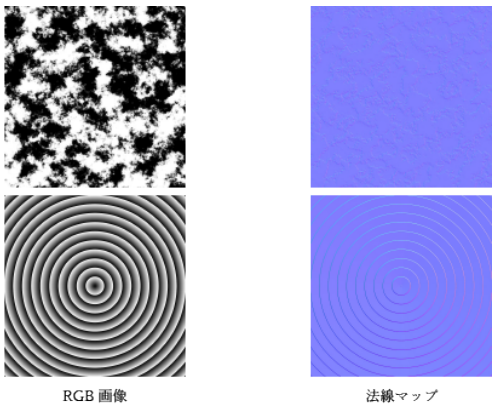

図 5RGB 画像と法線マップの対応例

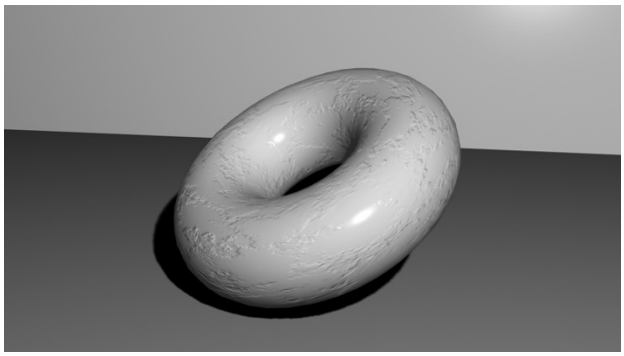

図 6 法線マップ適応例

3. 提案手法

本研究では Soft Rasterizer を用いた 2D Image to 3D Mesh Style Transfer を行う手法を提案する．図 7 に提案手法のアーキテクチャを示す．

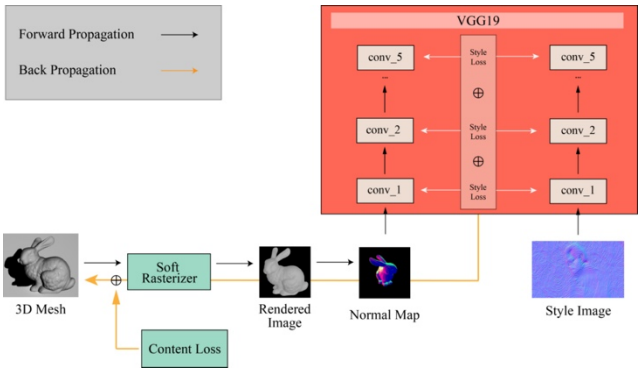

図 7 提案手法アーキテクチャ

3.1 損失関数

式 5 に示されているように、2D Image to 3D Mesh Style Transfer の損失関数は Gatys らの手法と同様に$L_{content}$とL_{style}から構成される．

$$L_{total} = \lambda_c L_{content} + \lambda_s L_{style} \qquad (5)$$

Vol.2022-CVIM-229 No.26
2022/3/10

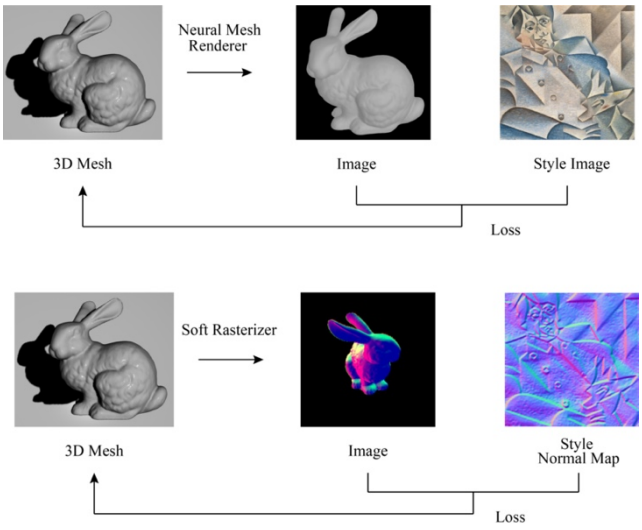

図 8 既存手法との比較
上段が Kato らによる手法、下段が本研究による提案手法

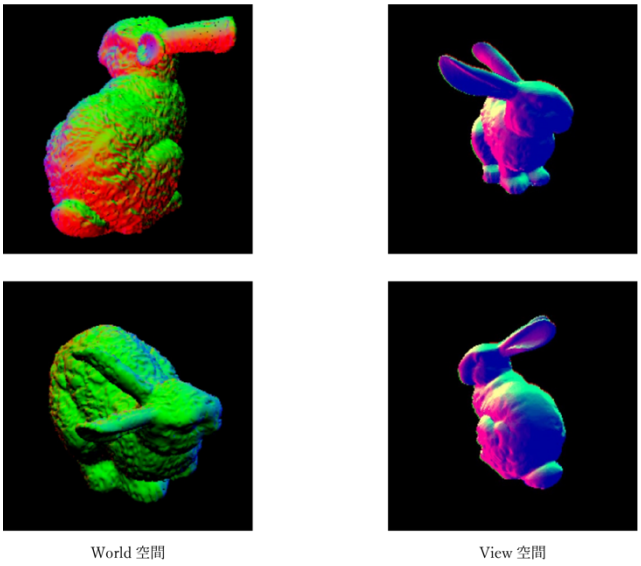

図 9 法線ベクトルの可視化

2D Image to 3D Mesh Style Transfer における$L_{content}$は 3 次元的な形状を保つために入力メッシュのオリジナルの頂点と変形後のメッシュの頂点座標の二乗誤差によって定義されている．$vertices$はオリジナルの頂点座標、$vertices'$は変形後のメッシュの頂点座標を示している．

$$L_{total} = (vertices - vertices')^2 \tag{6}$$

またL_{style}も Gatys らの手法と同じように、メッシュをレンダリングした画像を VGG19 に入力し、得られた特徴マップ間から算出されたグラム行列と参照スタイル画像との二乗誤差によって定義されている．

3.2 法線マップを用いた Style Transfer

本研究による提案手法と Kato らの手法との比較を図 8 に示す．Kato らはカラー画像を用いることで 2D Image to 3D Mesh Style Transfer を実現している．しかし、カラー情報のみではスタイル画像から細かい凹凸表現を抽出することが難しい．このような問題に対して、Liu ら[6]の手法では複数のライトによるライティングによってサーフェイスの詳細な陰影情報を利用している．しかし、使用するライブラリの制約上、複数のライトを用いたレンダリングが不可能な場合もあるため、本研究では法線マップを用いて 2D Image to 3D Mesh Style Transfer を実現する．メッシュの中心座標を World 空間の原点と一致させることによって World 空間での法線ベクトルは容易に取得することが出来る．しかし、図 9 にて示されているように World 空間での法線ベクトルと View 空間での法線ベクトルは異なる．法線マップに格納されている法線情報は View 座標系のベクトルであるため、View 空間でのメッシュの法線情報を算出する必要がある．View 空間でのメッシュの法線情報は式 7-10 のようにカメラからの距離である Depth を含むベクトルを用いることによって算出することができる．

$$pos_{view} = (\mathrm{x}, \mathrm{y}, depth) \tag{7}$$

$$\mathrm{dx} = (pos_{view} + (\Delta x, 0.0, 0.0) - pos_{view}) \tag{8}$$

$$\mathrm{dy} = (pos_{view} + (0.0, \Delta y, 0.0) - pos_{view}) \tag{9}$$

$$normal_{view} = cross(dx, dy) \tag{10}$$

4. 実験

図 10 に示されているメッシュを利用し、スタイル画像は[4]の実験内で使用されている画像と、自作画像を使用し、本研究によって提案された 2D Image to 3D Mesh Style Transfer に対する実験を行った．Style Loss に関する計算を行うために VGG19 内の畳み込み層 conv1_1, conv2_1, conv3_1, conv4_1, conv5_1 から特徴量を抽出した．また、損失関数に関するハイパーパラメータには$\lambda_c = 5.0e1$, $\lambda_s = 5.0e2$、レンダリングする際の画像サイズは 512 x 512、Batch Size = 4、Optimizer は Adam[7]($\beta_1 = 0.9$, $\beta_2 = 0.99$)を採用し、学習を 15epoch 行った．学習には NVIDIA GeForce RTX3080 を利用した．

4.1 定性評価

図 11 に 2D Image to 3D Mesh Style Transfer を行った結果を示す．結果画像では凹凸の状態をより的確に表現するために法線ベクトルの方向を色情報としてレンダリングしている．1 行目のスタイル画像の流線のような表現、2行目のスタイル画像の細かな格子、3行目のスタイル画像の大きな三角形を用いた表現など、直感的に汲み取ることができるスタイル情報をメッシュのサーフェイスに対して転写が実現されていることが確認出来る．

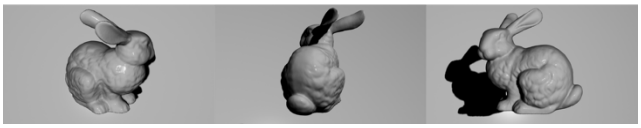

図 10 2D Image to 3D Mesh Style Transfer を敵応する Mesh(頂点数：34817)

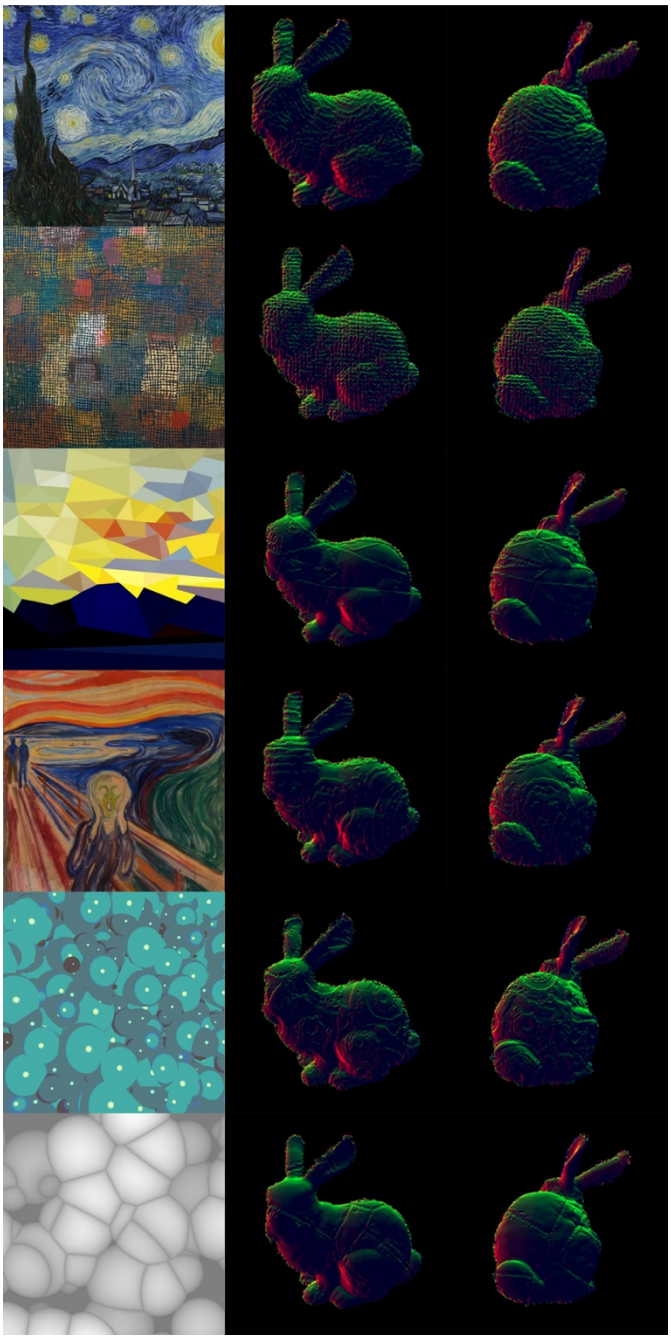

図 11 図 10 のメッシュに対して Image to 3D Mesh Style Transfer を敵応した結果．1 列目がスタイル画像、対応する行がスタイライズされたメッシュ

5. 結論

本研究では、複雑なスタイル情報であっても、深度情報から復元された法線情報を媒介とすることでメッシュのサーフェイスに対してスタイルを転写し、サーフェイスデザインのバリエーション生成を達成した．15 epoch(約 30 秒程度)の学習によって比較的高速かつ的確な生成結果を得ることが出来るため、デザイナーに対する 3D プロトタイピングツールとして今後の活用も期待出来る．また、今後の課題としてはスタイルを表現する凹凸の深さに対する制御、スタイルを表現するパターンの繰り返しの周期に対する制御を可能にすることがあげられる．

6. 参考文献

[1] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
[2] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style." *arXiv preprint arXiv:1508.06576* 2015.
[3] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* 2014.
[4] Kato, Hiroharu, Yoshitaka Ushiku, and Tatsuya Harada. "Neural 3d mesh renderer." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.
[5] Liu, Shichen, et al. "Soft rasterizer: A differentiable renderer for image-based 3d reasoning." *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019.
[6] Liu, Hsueh-Ti Derek, Michael Tao, and Alec Jacobson. "Paparazzi: surface editing by way of multi-view image processing." *ACM Trans. Graph*. 37.6 2018: 221-1.
[7] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* 2014.