CG によるスタイル表現の比較と実験

文理学部情報科学科
5419045 高林 秀

2021 年 7 月 29 日

概要

本稿では、今年度マルチメディア情報処理の第 2 回目課題研究として、「blender」及び「Google Colabratory」における 3DCG の描画をもちいて、配布されたデータ「Lit-Sphire」と [StyleBlit] を使用し CG 画像のスタイル表現の動画化を実験し、それぞれのデータにおける結果を比較・考察するものである。結果として、Lit-Sphire は「光の反射があるようなテクスチャのスタイル表現を得意とし、反対に人工的に描いた絵画のようなテクスチャのスタイル表現は、大きく歪んでしまうため不向き」であり、StyleBlit は Lit-Sphire とは対象的に「人工的に描いた絵画のようなテクスチャのスタイル表現を得意とし、光の反射があるようなテクスチャのスタイル表現は歪んだり、ノイズが現れたりするため不向き」であることが分かった。

1　目的

本稿では、今年度マルチメディア情報処理の第 2 回課題研究として CG スタイル表現を、レンダリング画像の連番データ化し、最終的には動画化をする実験を行う。本実験は、予め配布されたデータである、「Lit-Sphire」と [StyleBlit] を使用して行う。なお、実験環境は CG レンダリングソフト「brender」及び、クラウド上の Python 環境である「GoogleColaboratory」を利用した。なお計算機スペックについては、後述する実験準備の章をご覧頂きたい。

2　使用環境の紹介

まずは、本実験で使用したソフトウェア、サービスについて軽く説明する。

2.1　blender について

blender とは、オープンソースの統合型 3DCG ソフトウェアで、主に 3D モデリング、モーショングラフィックス、アニメーションやシミュレーション、レンダリングやデジタル合成など、3DCG における様々な機能を提供するソフトウェアである。Windows から MacOS、Linux 系の OS など幅広く対応している。主な開発言語は、Python, C, C++。

開発元は BlenderFoundation（Blender 財団）と呼ばれる非営利団体が行っており、この財団は短編コンピュータアニメーション映画の制作も行っている。

blender は、一般的な 3DCG ソフトウェアの中では比較的軽量であり、ライセンス料も無料である。そのためプロ層に限らずアマチュア層や素人でも 3DCG を体験することが可能だ。

blender は、実際の映画制作スタジオでも広く利用されつつあり、近年の代表的な例だとスタジオカラー[*1]の作品「シン・エヴァンゲリオン劇場版:||」の制作にも利用された。

表 1　blender の推奨動作要件

部品	スペック要件
CPU	4 コア以上の 64bit プロセッサ
GPU	VRAM が 4GB 以上のグラフィックプロセッサ
RAM	16GB 以上
ディスプレイ解像度	1920×1080 以上 (Full HD)

- blender 公式ページ：`https://blender.org/`

図 1　blender ロゴ

出典：`https://ja.wikipedia.org/wiki/Blender_Foundation`

2.2　GoogleColaboratory について

GoogleColaboratory（以降 Colab）とは、自身の PC 上に Python の環境構築を行うことなく Python を利用することができる Google のサービスである。Microsoft Edge や、Google Chrome などのウェブブラウザで動作するため、初心者から上級者まで幅広く Python を利用した開発を行うことができる。Colab は、機械学習の普及を目的としたサービスである。

見た目は、JupyetrNoteBook[*2]のウェブブラウザ版だと思って良い。Google アカウントさえあれば誰でも無料で使用でき、機械学習等で CPU 以外のプロセッサを利用したい場合は GPU[*3]や TPU[*4]も無料で利用することが可能だ。

Colab はブラウザがあれば動作するため、スマートフォンやタブレット端末からでも利用することができる。したがって機種に依存せず、複数人と共有して利用することが可能となる。

[*1] 日本のアニメーション制作会社である株式会社カラーの制作スタジオ。アニメーション制作会社ガイナックスの元取締役である庵野秀明氏が 2006 年 5 月に同社取締役を辞任し設立。社名である「カラー」はギリシャ語の「$\chi\alpha\rho\acute{\alpha}$(歓喜)」に由来。

[*2] ブラウザ上で動作する、対話型の Python 実行環境。Anaconda に付属している。

[*3] GPU:Graphics Processing Unit の略。画像処理に特化したプロセッサ。コンピュータが画面に描画する映像の計算処理を主な任務としている。そのため、CPU よりも単純構造でコアを大量に積んでいるため、並列計算に特化している。代表的な GPU には、Nvidea 社の GEFORCE がある。

[*4] TPU：Tensor processing Unit の略。Google が開発した、機械学習に特化した集積回路（ASIC）。GPU より 1 ワットあたりの IOPS が高いが、計算精度は劣る。

図 2　Colab ロゴ

出典：https://www.tcom242242.net/entry/%E3%83%A1%E3%83%A2/colab/%E3%80%90%E5%85%A5%E9%96%80%E3%80%91colaboratory%E3%81%AE%E5%A7%8B%E3%82%81%E6%96%B9/

- Colab ホームページ：https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja

3　関連技術調査

今回配布されてたデータ「Lit-Sphire」と [StyleBlit] について説明する。

なお事前に説明すると、テクスチャとは、「3D オブジェクトンの凹凸や色の画像」のことで、マッピングとは「テクスチャを 3D オブジェクトに貼り付けること」をそれぞれ意味する。

3.1　Lit-Sphire

Lit-Sphire とは、3D シェーディング[*5]の手法において、球状の環境マップ[*6]利用したシェーディングの手法のことである。この手法では、材質と光を表現した球状の絵を利用し、擬似的にライティングを行っている。したがって、事前にライティングの計算を済ませてから、適用先オブジェクトへのライティングを行っている。

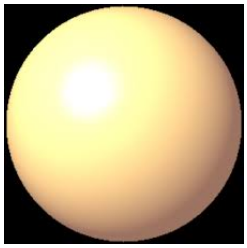


図 3　球状環境マップの一例

出典：https://mevislabdownloads.mevis.de/docs/current/MeVisLab/Standard/Documentation/Publish/ModuleReference/SoGVRLitSphereShading.html

Lit-Sphre は、ユーザーからそのオブジェクトが見える位置であるカメラ座標の法線座標をテクスチャの座標として使用し、法線と環境マップの色の対応付けを行う役割を果たす。テクスチャ画像の RGB 値が「面の向き」を示している。R 値が右側の面の向きを示し、G 値が上側の面の向きを示している。この面の向きと、マテリアル画像の位置関係の対応を計算し、その色を構成画像上に張り付けることで、レンダリングしている。

[*5] 3DCG において、イラスト等を使用し明暗のコントラストを与え、モデルに立体感を与える技術のこと

[*6] 3DCG におけるテクスチャマッピングの手法の 1 つ。3D 上の表面に擬似的な周囲環境の映り込みを再現する手法。

本稿では取り扱わないが、他のシェーディング手法である Lambert シェーディングや、Phong シェーディングとは違い環境マップを使用して直感的に色を成業する事ができるので、NPR[*7]目的として都合が良いシェーダーであるとされる。

ところで、NPR とは反対の目的で「PR : Photorealistic rendering」と呼ばれるものがあるが、これは「物理現象のシュミレーションにより、より現実見のある (写実的な)CG レンダリング」を目的として行われる。

図 4　PR の一例

出典 : https://jp.123rf.com/photo_85627230_%E5%AE%B6%E3%81%AE%E5%86%99%E5%AE%9F%E7%9A%84%E3%81%AA%E3%83%AC%E3%83%B3%E3%83%80%E3%83%AA%E3%83%B3%E3%82%B0%E3%81%AE-3-d-%E5%9B%B3.html

上図のように精密な写実的な CG モデルを構築するには、被写体のモデルも精密である必要がある。対して、NPR は絵画や、イラスト等のあまり現実味のない (非写実的) な画像、CG を構築することを示す。人が描く絵を書くように CG を構築することを指す。

図 5　NPR の一例

出典 : http://modogroup.jp/modo/kits/npr

3.2　StyleBlit

レンダリングには、通常比較的強いマシンスペックが要求されることが多い。しかし、Sty5leBlit と呼ばれるレンダリング手法は、1 コアの CPU でも高品質なレンダリングを提供することのできる技術、アルゴリズムとなっている。

[*7] NPR : Non Photorealistic rendering の略。和訳は「非写実的レンダリング」。

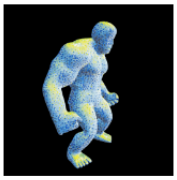
StyleBlit とは、高品質で定型化されたレンダリングを提供できる効率的なスタイル転送アルゴリズムである。この StyleBlit は比較的計算量の少ないゲームやモバイル系のアプリに適したアルゴリズムとなっている。

StyleBlit に関して、藤堂英樹氏のブログ [1] にて非常に分かりやすく紹介されているので、引用する。

> StyleBlit は，Lit-Sphere と同様に，球にデザインされたシェーディングを 3D モデル上に転写できますが，ストロークのような細かい特徴まで合わせて転写することが可能です．

つまり、StyleBlit は Lit-Sphere と同様に球状の環境マップを利用したシェーディングの手法と言うことになる。では、実際のレンダリングにおいて、両者にはどの様な相違点があるのか。同氏のブログ [1] には次のように述べられている。

> 通常の Lit-Sphere だと，下記のようなストロークタッチ込みのマップは法線マップの影響で歪んでしまってうまく転写できませんが，StyleBlit だと歪みなく転写することができます．

スタイル画像	転写結果	スタイル画像	転写結果
			

つまり、Lit-Sphire では歪んでしまうような部分でも、StyleBlit を利用することで鮮明に歪みなくレンダリングをすつことができるという点で、両者は大きく異なっていると言える。

巻末付録に、StyleBlit の開発者ホームページへのリンクを載せる。必要に応じてご参照いただきたい。

■手法解説　以下、同氏のブログに記載されている内容を踏襲しながら StyleBlit の流れを説明する。その前に、3D レンダリングで何かと登場することの多い「法線マップ」とはなにか説明する。

法線マップとは、ポリゴン[*8]に対して、貼り付けることのできるテクスチャの画像データとは別に専用の画像データで割り当てることのできる、法線ベクトルの集合データのことである。法線はポリゴン上の凹凸の影を表現するのに利用され、3D オブジェクトの 1 枚のテクスチャ上に対する、光の入射方向と法線マップ上の法線の方向、すなわち法線ベクトルを元に計算している。

[*8] ポリゴン：曲面を構成する最小単位。英語では「Polygon」。多角形という意味。3DCG において、3 点以上の頂点を結んで得られる多角形データのこと。

図 6　法線マップの一例

出典：https://nodachisoft.com/common/jp/article/jp000050/

StyleBlit では、スタイル転写を実行する時ガイド画像とよばれる法線画像を用いて特徴の類似する部分を探索する。このとき、法線のガイド画像等を入力し、RGB の画像を出力する。以下アルゴルズムの概要を示す。

- 入力一覧
  - C_S：スタイル画像 (RGB)
  - G_S：スタイルガイド画像 (法線画像)
  - G_T：ターゲットガイド画像 (法線画像)
- 出力
  - C_T：ターゲット画像

出力画像 C_T のピクセル p の色値 (RGB)$C_T(p)$ は次に示す手順で算出される。

1. ターゲットガイド画像の法線空間におけるピクセルの色 $G_T(p)$ の特徴に類似するパッチ領域 $p \in \Omega_{T_i}$ を考える。
2. 上記バッチ領域における代表点 q を計算する。
3. 代表点 $ q のターゲットガイド画像の法線 $G_T(q)$ に類似する法線を持つピクセル u をスタイルガイド画像 G_S の法線 $G_S(u)$ から探索する。
4. u を中心としたパッチ領域 $u \in \Omega_{S_i}$ を考える。
5. スタイル画像のピクセルの RGB 値を $C_S(u)$ とし、ターゲット画像側に $C_S(u)$ を以下の形で転写する。

$$C_T(p) = C_S(u + p - q)$$

以下の画像は、開発者が Youtube 上に公開している StyleBlit の概要動画における、MatCap との比較シーンにおける一部場面である。

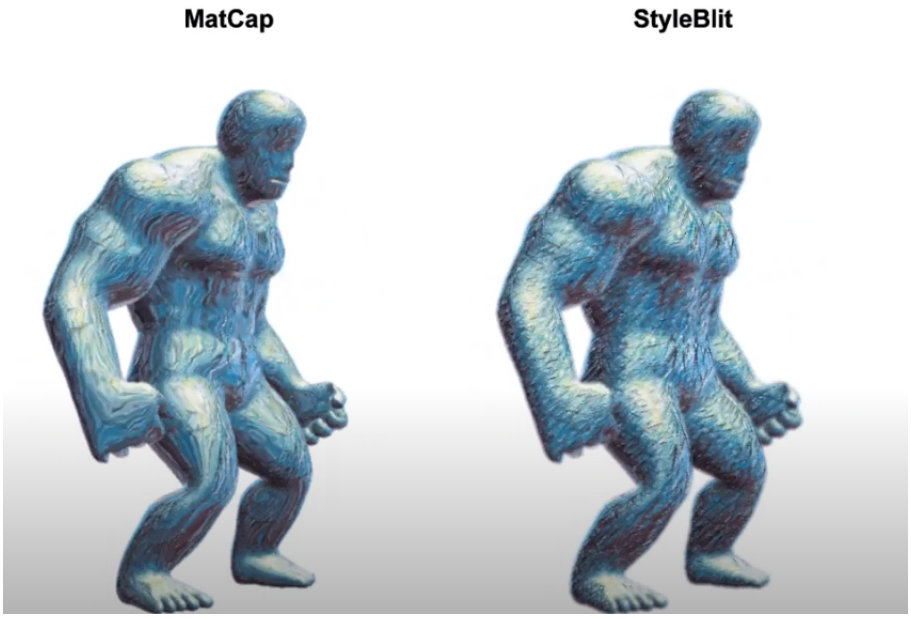


図 7　MatCap と StyleBlit の比較

出典：https://www.youtube.com/watch?v=krQrDhestVA&t=1s

上記を見ると確かに、オブジェクトの表面の歪みが StyleBlit の方が無いことが伺える。

4　実験方法

まず、本実験の目標を改めて説明する。本実験は「Lit-Sphere と StyleBlit のスタイル表現をレンダリング画像の連番データに適用する」ことを目的とする。

4.1　実験準備

■実験環境　今回の実験は以下の環境上で行った。下記に実験時の環境を示す。

- OS：Window10 Home Ver.20H2
- CPU：Intel(R)Core(TM)i7-9700K 8cores @ 3.6GHz
- GPU：Nvidia Geforce RTX2070 OC VRAM 8GB
- RAM：16GB
- brender のバージョン：2.93.1
- Chrome のバージョン： 92.0.4515.107

4.2　実験手順

まず、スタイル表現を動画化する全体的な流れは以下の通りである。

1. blender ファイルより連番画像を制作

2. 配布された Colab ノートブックにて上記画像を 1 枚ずつスタイル化処理
3. スタイル化した画像を動画化

以下具体的な流れを説明する。

1. 配布されたスライド資料にしたがって、blender で連番画像を出力し、結果を ZIP 形式で圧縮し一つにまとめておく。
   （a）blender ソフトを起動。
   （b）blender 内で対象の blender ファイルを開く。
   （c）上部のメニューバーにある「Render Animation」を選択すると、対象のフォルダに連番画像が出力される。
   （d）生成されたフォルダを zip に圧縮し保存しておく。
2. 以下、配布された Colab ノートブックにある「必要なライブラリのインストール」のコードセルを実行する。
3. ライブラリのインストールが完了したら、画像ベースのスタイル化の実装例と補助関数のコードセルをそれぞれ順番に実行する。
4. 必要なファイルを Colab 上にアップロードする。
   - 左のサイドバーからフォルダのアイコンをクリックし、「セッションストレージにアップロード」を選択し、エクスプローラーからファイルを選択する。
5. ファイルのアップロードが完了したら Colab の入力画像の設定にある指示にしたがって、必要なパラメータを指定し、コードセルを実行する。
6. Colab の Lit-Sphere/StyleBlit の結果の確認からそれぞれのレンダリング手法による結果を確認し、考察を行う。
7. Colab の Lit-Sphere/StyleBlit の動画出力結果のコードセルをそれぞれ実行し、動画化を行う。その結果を確認し、考察を行う。

5　実験結果

以下、スタイル化結果のスクリーンショットを示す。

5.1　モデル：suzanne の場合

テクスチャ画像が左で、転写するモデルが右側に示されている。

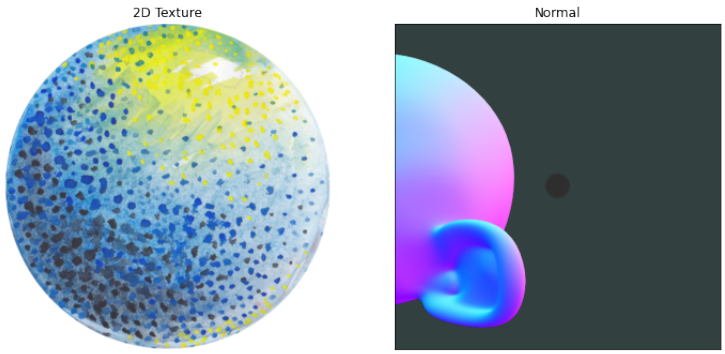


図 8　入力画像 1

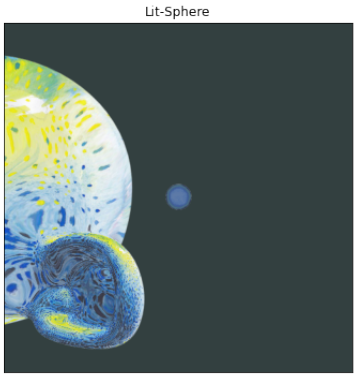


図 9　Lit-Sphire での出力結果画像

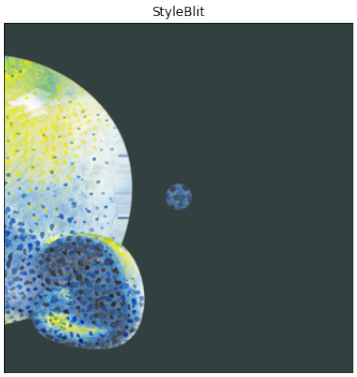


図 10　StyleBlit での出力結果画像

なおこの条件での動画出力結果は、以下のリンクから参照できる。

- URL:`https://drive.google.com/drive/folders/112dBLmxLyQe94MToRBUOHlm75C4zrxKg?usp=sharing`

次に、テクスチャを変更して同様の実験を行った。その時の結果は下記。

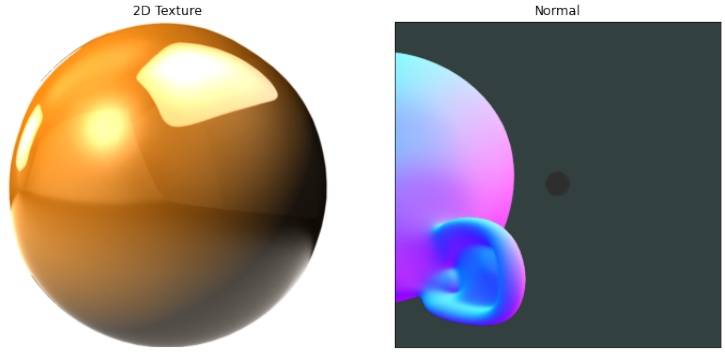


図 11　入力画像 2

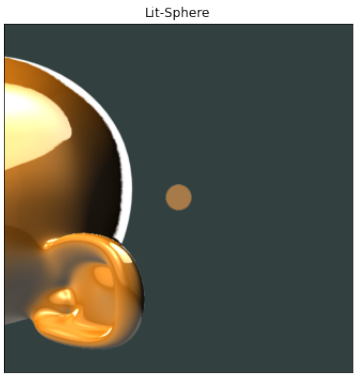


図 12　Lit-Sphire での出力結果画像

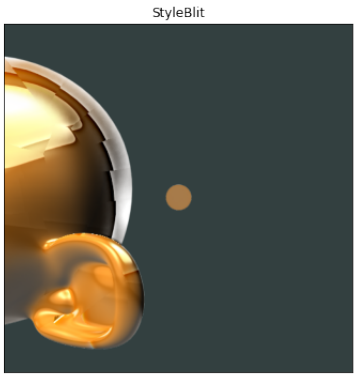


図 13　StyleBlit での出力結果画像

なおこの条件での動画出力結果は、以下のリンクから参照できる。

- URL:`https://drive.google.com/drive/folders/1L7WVmp5V8TFQB9ILghPZwqvu1XI9gEQv?usp=sharing`

5.2　モデル：blobby の場合

また、異なる 3D モデルでも同様の実験を行ってみた。

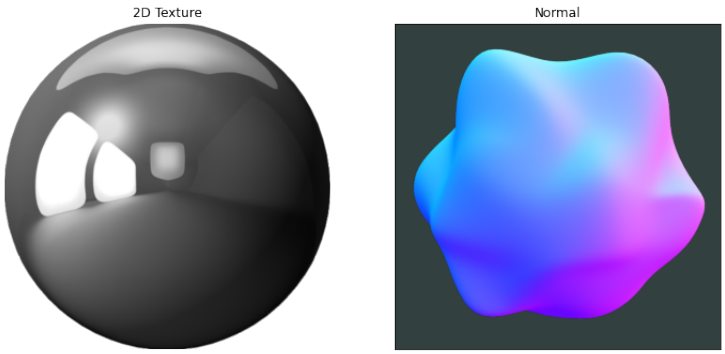


図 14　入力画像 3

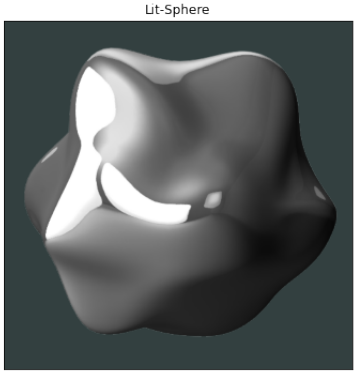


図 15　Lit-Sphire での出力結果画像

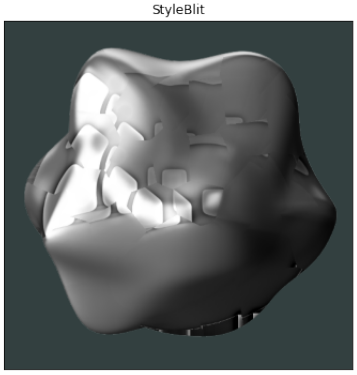


図 16　StyleBlit での出力結果画像

なおこの条件での動画出力結果は、以下のリンクから参照できる。

- URL:`https://drive.google.com/drive/folders/1fics3oMdK64yPkb5z2NB42zhTQyoBngQ?usp=sharing`

加えて、異なるテクスチャでも同様の実験を行ってみた。

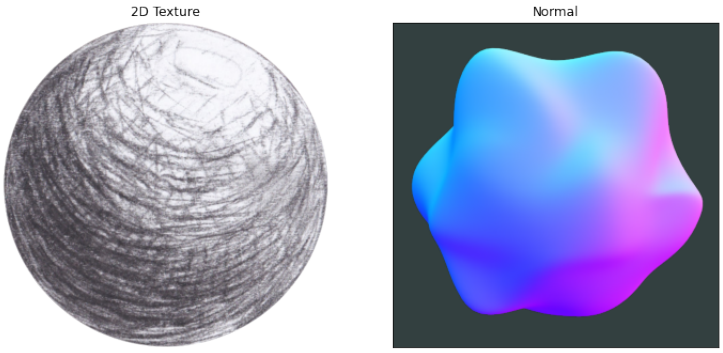


図 17　入力画像 4

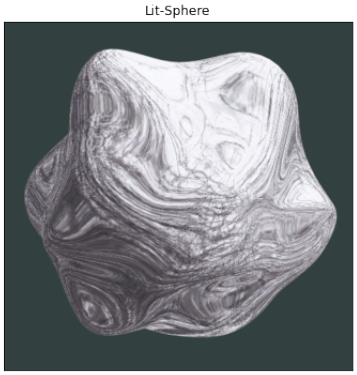


図 18　Lit-Sphire での出力結果画像

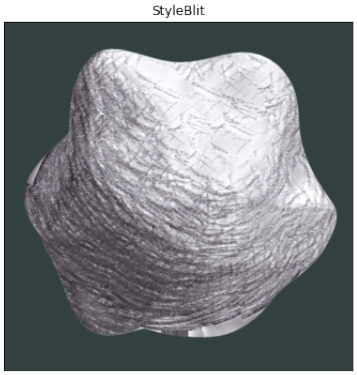


図 19　StyleBlit での出力結果画像

なおこの条件での動画出力結果は、以下のリンクから参照できる。

- URL:https://drive.google.com/drive/folders/1wMASt-w3BbsDIBcSFDGM8_pVyUbzJpUO?usp=sharing

5.3 結果の説明

まず、モデル suzanne の場合を説明する。最初の、入力画像 1 の条件における結果において Lit-Sphire では耳の部分のテクスチャが StyleBlit の結果よりも大きく歪んでいるのが確認できる。 2 つ目のテクスチャの場合 (入力画像 2) では Lit-Sphire の方が比較的忠実に、元のテクスチャを反映しており、StyleBlit の場合では光を反射している部分がすこし乱れて、角ばっているように見える。

次に、モデル blobby の場合を説明する。入力画像 3 の条件における結果では、Lit-Sphire の方はしっかりと滑らかに入力のテクスチャが転写されているが、StyleBlit の場合では、なにかくすんだノイズのようなものが一緒に出力されているのが分かるだろう。入力画像 4 の条件における結果では、Lit-Sphire の方が明らかに StyleBlit の場合よりもテクスチャの歪みが生じている。特に、入力画像のテクスチャ画像では、鉛筆で線を描いたような真っ直ぐな短線のようなテクスチャが Lit-Sphire でレンダリングすると曲がってしまっているのが確認できる。

6 考察

まず、画像における結果の考察をする。結果の説明からも分かるとおり、使用するテクスチャによって Lit-Sphire と StyleBlit のそれぞれの手法に対し、向き不向きがあるように思われる。その裏付けとして、例えば、モデル blobby の場合で、入力画像 3 のテクスチャと入力画像 4 のテクスチャにおける、StyleBlit の結果を比較してほしい。入力画像 3 では色合いは元のテクスチャを忠実に再現しているが、3D モデルの表面になにかノイズのようなものが出現している。これは左側の結果画像である Lit-Sphire の場合と比較した際に明らかに異なっている点である。また、結果の説明でも述べたが、入力画像 4 の場合では、入力のテクスチャの線模様が StyleBlit では忠実に転写されているのに対し、Lit-Sphire では明らかに歪んでしまっている。

このとき、入力画像 3 と入力画像 4 にて使用したテクスチャの大きな違いは、光 (ライティング) の反射があるか否かで、入力画像 3 の方は反射あり、入力画像 4 の方は反射がなく、鉛筆で線を描いたようなテクスチャになっている。以上の結果から、Lit-Sphire は「光の反射があるようなテクスチャのスタイル表現を得意とし、反対に人工的に描いた絵画のようなテクスチャのスタイル表現は、大きく歪んでしまうため不向き」であると推察される。また、StyleBlit は Lit-Sphire とは対象的に「人工的に描いた絵画のようなテクスチャのスタイル表現を得意とし、光の反射があるようなテクスチャのスタイル表現は歪んだり、ノイズが現れたりするため不向き」であると推察される。

次に、動画における結果の考察を行う。動画にすると、先ほど述べた Lit-Sphire と StyleBlit の長所・短所がより分かりやすく現れる形で目に見えた。例えば、入力画像 1 の条件では、画像ではモデルの耳のあたりのテクスチャが Lit-Sphire では大きく歪んでしまっている。これが映像として出力されると、モデルが移動するたびに歪んでいる部分の模様が不自然に変化しているのが確認できた。反対に、StyleBlit での動画は、Lit-Sphire で歪んでいた部分のテクスチャは、モデルが移動しても変化することなく、入力画像のテクスチャを忠実に再現していた。このことからも、先に述べた Lit-Sphire と StyleBlit の長所・短所の裏付けができる

と思う。

7　まとめ

本稿では、スタイル表現の手法である Lit-Sphire と StyleBlit を使用し、実際に Colab 上で 3D オブジェクト、モデルのスタイル表現の実験を行うことで、両者の性能を比較し、その相違点を考察した。結果以下の様なことが分かった。

Lit-Sphire は「光の反射があるようなテクスチャのスタイル表現を得意とし、反対に人工的に描いた絵画のようなテクスチャのスタイル表現は、大きく歪んでしまうため不向き」

StyleBlit は Lit-Sphire とは対象的に「人工的に描いた絵画のようなテクスチャのスタイル表現を得意とし、光の反射があるようなテクスチャのスタイル表現は歪んだり、ノイズが現れたりするため不向き」

今後の事後調査で、何故このような違いが出るのか調べていく。

8　巻末付録

- StyleBlit 開発者ページリンク：`https://dcgi.fel.cvut.cz/home/sykorad/styleblit.html`
- GoogleDrive へのリンク:`https://drive.google.com/drive/folders/1ciY7XHNFSsUBXfiAC9xqw2D2_LYDKVSv?usp=sharing`
- GitHub リポジトリはこちら：`https://github.com/tsyu12345/Multi-report2`

参考文献

[1] 藤堂英樹、『研究開発日誌』より StyleBlit、(2019.7.30)、`http://hideki-todo.com/cgu/blog/article/styleblit/`