Impact Factor:					
	ISRA (India) = 4.971	SIS (USA) = 0.912	ICV (Poland) = 6.630
	ISI (Dubai, UAE) = 0.829	РИНЦ (Russia) = 0.126	PIF (India) = 1.940
	GIF (Australia) = 0.564	ESJI (KZ) = 8.997	IBI (India) = 4.260
	JIF = 1.500	SJIF (Morocco) = 5.667	OAJI (USA) = 0.350

SOI: 1.1/TAS DOI: 10.15863/TAS

International Scientific Journal

Theoretical & Applied Science

p-ISSN: 2308-4944 (print) **e-ISSN:** 2409-0085 (online)

Year: 2021 **Issue:** 02 **Volume:** 94

Published: 14.02.2021 **http://T-Science.org**

QR – Issue QR – Article

Andrey Andreevich Kadomskij
Peter the Great St. Petersburg Polytechnic University
Researcher
Saint-Petersburg, Russia

Oleg Yurievich Sabinin
Peter the Great St. Petersburg Polytechnic University
PhD in Computer Science
Saint-Petersburg, Russia

STUDY OF THE POSSIBILITY OF STYLIZATION OF MATERIALS OF THREE-DIMENSIONAL MODELS BASED ON THE APPARATUS OF ARTIFICIAL NEURAL NETWORKS

Abstract: *The purpose of this work is to create a program that can create stylized materials for three-dimensional models, allowing a 3D artist to automate the process of creating materials.*

Key words: *texture, material, three-dimensional model, stylization.*

Language: *Russian*

Citation: Kadomskij, A. A., & Sabinin, O. Y. (2021). Study of the possibility of stylization of materials of three-dimensional models based on the apparatus of artificial neural networks. *ISJ Theoretical & Applied Science, 02 (94),* 165-176.

Soi: http://s-o-i.org/1.1/TAS-02-94-40 ***Doi***: crossref https://dx.doi.org/10.15863/TAS.2021.02.94.40

Scopus ASCC: *1700.*

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТИ СТИЛИЗАЦИИ МАТЕРИАЛОВ ТРЕХМЕРНЫХ МОДЕЛЕЙ, ОСНОВАННОЙ НА АППАРАТЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ.

Аннотация: *Целью данной работы является создание программы, способной создавать стилизованные материалы для трехмерных моделей, позволяющей 3D художнику автоматизировать процесс создания материалов.*

Ключевые слова: *текстура, материал, трехмерная модель, стилизация.*

Введение

В этой статье демонстрируется решение задачи стилизации трёхмерных моделей, какие этапы в этом процессе уже заменяются на решения с использованием алгоритмов нейронных сетей (Neutral Networks – NN), почему выбор делается именно в их пользу (материалы, DeepBRDF, SVDRDF). Рассматриваются также методы, использующиеся для выравнивания финального (законченного) стиля (постпроцессинг и super sampling от Nvidia). Будет рассмотрена генерация, сетка модели и текстуры при создании игровой модели (а также решение компании Netease Games, с помощью которого по одной входной фотографии можно сгенерировать выходного персонажа, стилизованного под человека на фото) и в конце показано решение для графики от TensorFlow – TensorFlow Graphics оценка его применимости для решения поставленной задачи.

Сегодня в создании трёхмерных моделей и визуализации (в основном для игр и фильмов) используют специальные программы (Substance, Quixel, Marmoset, небольшие плагины для больших графических пакетов, таких как Maya, ZBrush, Blender), в которых специалист – 3D художник – может с помощью различных

инструментов раскрасить модель и после этого выгрузить текстуры и материалы (assets), которые впоследствии будут использованы в игре (фильме или чём-то подобном). На создание одной модели таким методом тратится как много времени, так и много ресурсов. Модель с текстурами получается статической, а сегодня, особенно в играх, часто необходимо динамичное изменение некоторых свойств игровой (т. е. задействованной в игре) модели для придания сцене большей реалистичности. Надо сказать, что такая необходимость уже давно стояла перед игровой индустрией – речь пойдёт в основном о ней, в остальных областях проблемы либо аналогичны, либо слишком специфичны.

Но прежде, чем перейти к описанию того, как именно в игровой индустрии подошли к решению проблемы, надо рассказать, как происходит render – отрисовка финального кадра. Точное понимание всего процесса rendering (рендеринга) позволит чётче увидеть проблему и понять, почему индустрия двинулась в таком-то (ниже описывается, в каком же именно) направлении для решения.

Render – это процесс отрисовки кадра на экране. В играх существуют разные механики (Deferred, Forward, Forward+, Clustered Forward и т. п. Их отличия не играют важной роли для задач, но суть заключается в том, что для создания финального кадра создаётся большое количество промежуточных кадров, на каждом из которых хранится только часть необходимой информации (что позволяет добавление максимально большого количества источников света в сцену), и уже из этой информации, её анализа и сопоставления с другими данными получается новое, изменённое или стилизованное изображение (рис. 1).

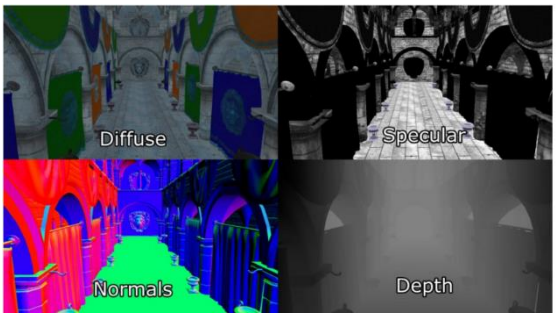

Рисунок 1 – Карты материала.

В «собираемой» таким образом картинке много внимания уделяется текстурированию, ведь текстура объекта состоит из самых разнообразных материалов и нескольких слоёв, в буквальном смысле отражающих свойства окружающей сцены (количество источников света и его интенсивность, огня, теней и многого другого – что может рассеиваться, преломляться, отражаться и т. д.). В статье будет описано, каким именно образом пользователь может влиять на стилизацию (изменения конечной текстуры в зависимости от новых или изменившихся данных) и как в этом процессе могут быть полезны алгоритмы Neutral Networks (NN).

Речь коснётся типов рендера (для разных типов используют разные наборы алгоритмов), а также методов генерации обучающих данных в контексте компьютерной графики на (рис. 2).

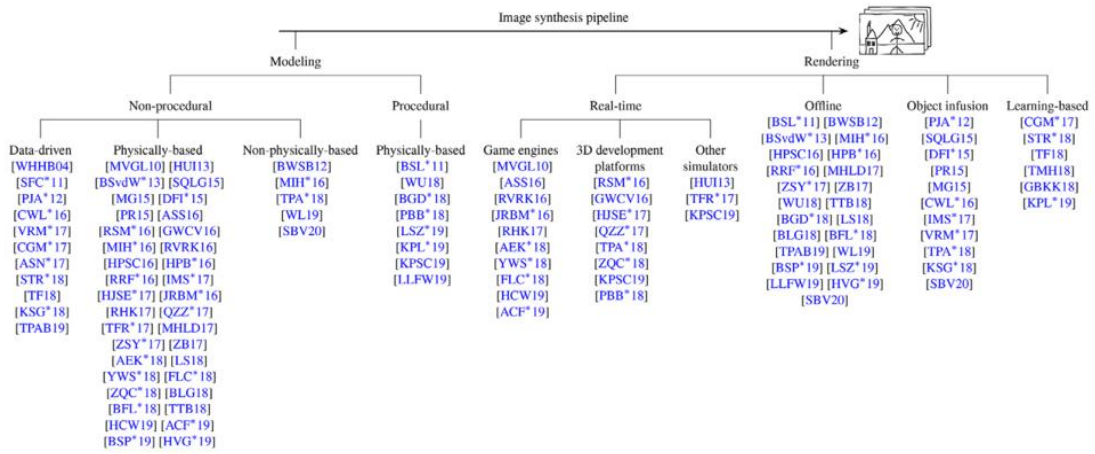

Рисунок 2 – Методы генерации обучающих данных в контексте компьютерной графики [12, с. 6].

Речь пойдёт преимущественно про real-time render (пересчёт которого происходит непосредственно во время выполнения программы), противоположности offline (когда все вычисления происходят сначала на стороне GPU/CPU, а уже после отображаются на экране), о котором также скажем пару слов. Для придания реалистичности (или просто единого стиля) в графике используют так называемые модели освещения и материалов: от простых bloom, lambert (названных по фамилиям инженеров предложивших формулу их расчёта) до сложных, учитывающих степень реалистичности распространения света в предложенной среде (которая может сильно варьироваться) и использующих формулы Чандрасекхара или Макгири [23] (McGuire), модели отражения, теней и сглаживания. До последнего времени решение поставленной задачи требовало высокой производительности GPU, за счёт которой и проводились столь ресурсоёмкие вычисления на стороне шейдера, и всё же качественное отображение поверхности текстуры, состоящей из комбинации разных материалов, было очень низким, а постоянно увеличивающиеся разрешения (4K) и возросшая частота кадров требовали больших ресурсов и сильно загружали графический процессор.

Поставлена задача: получить в процессе рендеринга несколько текстур (а именно Albedo, Normal, Roughness, Metallic), которые будут отображать изменения в окружающем игровом мире. То есть, учитывать интенсивность, позицию и источники света, а также тени. Таким образом модель будет стилизоваться под окружающий её мир. В статье будут рассмотрены несколько существующих подходов для решения аналогичных или смежных задач.

Для придания трёхмерным моделям реалистичности каждый пиксель текстуры (на самом деле microfacet – небольшую область текстуры) считают по формуле, учитывающей свет, падающий на эту поверхность, способность её материала отражать/рассеивать и угол обзора (откуда можно наблюдать этот эффект и некоторую аппроксимацию). Называется эта методика bidirectional reflectance distribution function (сокращённо BRDF). Каждое такое вычисление необычайно громоздкое (к тому же с повышением точности и реалистичности требуется всё больше переменных для вычисления). Законченная трёхмерная модель имеет несколько текстур, в которых хранится информации о цвете (Albedo map), шероховатости и неровности (Roughness map), нормалей (Normal map) и металл (Metallic map), бывают дополнительные и другие текстуры, но они всё равно используют такие же данные для отображения. Рассмотрим несколько методов: первый – SVBRDF – где по одной входящей фотографии NN генерирует на выходе четыре карты, пригодные для использования в игре или другой задаче по визуализации (рис. 3).

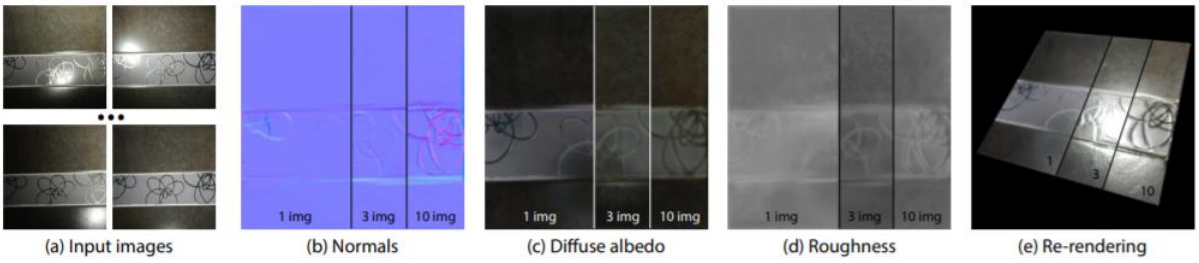

Рисунок 3 – Четыре карты для использования в задачах по визуализации [10].

Теоретические основы работы

Методы DeepBRDF, SVBRDF.

Существует два этапа работы метода: генерация карты методом [13, с. 2], основанном на distant pixels, а тем генерация карты функции, сохраняющей информацию для обработки.

Интересный момент: в метод не поступает информация о свете и позиции камеры, так как поставленная задача – максимально упростить входные данные.

Вторая часть объединяет несколько карт функций производимыми сетями одного изображения, чтобы сформировать единый объект карт фиксированного размера.

В частности, дорожка кодера-декодера каждой сети с одиночным изображением создает карту промежуточных характеристик размером 256 × 256 × 64, соответствующую обработанному входному изображению. Все они объединены в единую карту объектов одинакового размера путём выбора максимального значения, сообщаемым любой сетью с одним изображением для каждого пикселя и канала функции. Эта процедура максимального объединения даёт каждому отдельному изображению равные

возможности в сети и средства для внесения вклада в содержание совместной функции финальных (генерируемых) карт совершенно независимо от порядка.

Наконец объединённая промежуточная карта признаков декодируется тремя слоями свёрток и нелинейностей, которые обеспечивают сети достаточную выразительность для преобразования извлеченной информации в четыре карты (рис. 4). Глобальные функции полностью подключенной дорожки объединяются в максимальный пул и декодируются аналогичным образом. Через сквозное обучение сети с одним изображением учатся производить функции SVBRDF, которые имеют значение для операции объединения и полезны для восстановления окончательной оценки.

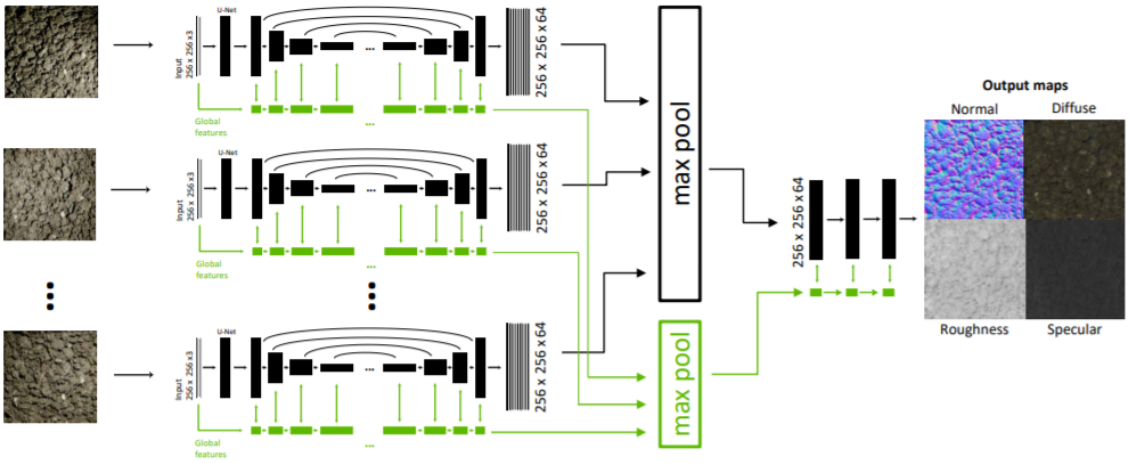

Рисунок 4 – Преобразования извлеченной информации в четыре карты.

Оценивается качество сетевого предсказания с дифференцируемой потерей рендеринга рассчитанной по следующей формуле:

$$L = LRender + 0.1\,LNormal + LDiffuse + LSpecular + LRoughness, \quad (1)$$

где все названия соответствуют генерируемым картам.

Ещё один метод, который пока не применяется в играх, но является интересным результатом переосмысления и практического применения стилизации – получение из одной фотографии нескольких карт (света, глубины, самого изображения и т. д.), которые уже могут быть применимы в игре – DeepBRDF. С использованием доработанных моделей Convolutional Inverse Graphics Network (DC-IGN) [24] получается не просто текстура, а скорее завершённый аппарат для финального преобразования всей композиции, включающую не только материалы, но и расчёт освещения (рис. 5).

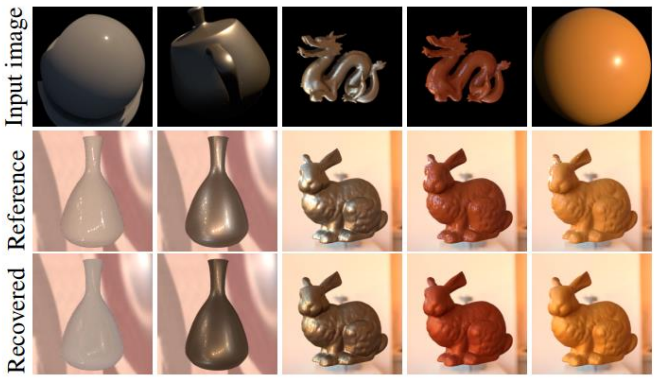

Рисунок 5 – Завершённый аппарат для финального преобразования всей композиции.

Эта сеть натренирована на датасете (dataset) MERL BRDF [19, с. 2] (более 100 изотропных материалов с разной степенью коэффициента отражения) с весьма интересными особенностями: так как поступающая картинка часто имеет довольно широкий HDR диапазон, здесь применена формула, сжимающая данные – такая обработка позволяет значительно расширить диапазон входного изображения. После предварительной обработки данных BRDF в создании решения мы проектируем и обучаем глубокий автоэнкодер [25, с. 75] для исследования низкоразмерного представления входных BRDF. Автоэнкодер использует кодировщик для уменьшения размерности входных данных X и для извлечения низкоразмерного скрытого признака Y для каждого BRDF. Математически есть:

$$Y = fq(X), \quad (2)$$

где fq обозначает кодировщик, параметризованный q.

Затем последующая сеть декодера преобразует скрытое представление Y обратно в многомерный выход Xˆ, который, как ожидается, будет похож на вход, то есть:

$$\hat{X} = gq(Y), \quad (3)$$

Набор параметров q кодера и декодера изучен одновременно с задачей реконструкции BRDF. В (табл. 1) архитектура нашего автокодировщика.

Таблица 1. Архитектура автокодировщика

Layer	Kernel	Stride	Resolution
Input			540 × 90 × 90
Conv2D	3 × 3	2	256 × 45 × 45
Conv2D	3 × 3	2	128 × 23 × 23
Conv2D	3 × 3	2	64 × 12 × 12
ResidualBlock	3 × 3	1	64 × 12 × 12
ResidualBlock	3 × 3	1	64 × 12 × 12
ResidualBlock	3 × 3	1	64 × 12 × 12
FC-10			
FC-64 × 12 × 12			
DeConv2D	3 × 3	2	128 × 24 × 24
DeConv2D	3 × 3	2	256 × 48 × 48
DeConv2D	4 × 4	2	540 × 90 × 90
Output			540 × 90 × 90

В предлагаемой сети три 2D свёрточных слоя (Conv2D) с размером ядра 3 × 3 и шагом 2 используются для субдискретизации входных данных. Разрешение уменьшается вдвое после каждого свёрточного слоя. Три остаточных блока (ResidualBlock) вставляются для повышения эффективности обучения. Каждый остаточный блок содержит два свёрточных слоя, блок активации ReLU с утечкой и остаточное соединение. Эти свёрточные шаги расширяют рецептивные поля предлагаемой сети. Для извлечения скрытого вектора малой размерности используется полностью связанный слой. Чтобы расшифровать скрытый вектор, его расширяют другим полностью связанным слоем (FC-64 × 12 × 12), за которым следуют три деконволюционных слоя (Deconv2D). На вход автоэнкодера поступают полные данные BRDF, представленные переупорядоченными угловыми координатами половинной разности Русинкевича: (fd; qh; qd), которые получаются из dataset, и преобразованные в 540 × 90 × 90 понижающей формулой, описанной выше. Таким образом, каждая измеренная BRDF фактически рассматривается, как последовательность срезов изображения. После обучения сложенные фрагменты изображения могут быть полностью восстановлены сетью (рис. 6).

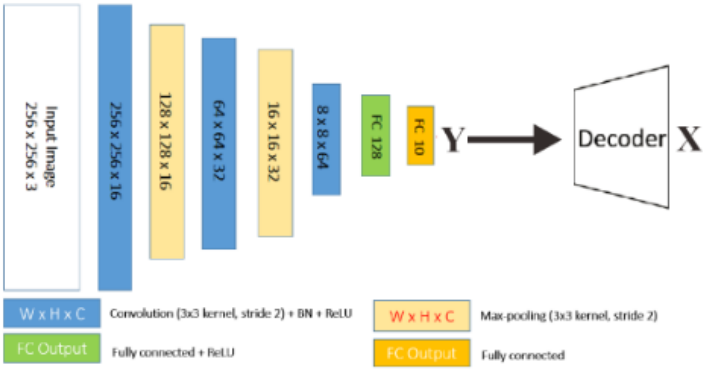

Рисунок 6 – Сложенные фрагменты изображения.

В итоге можно видеть, как это решение, при получении небольшой картинки с произвольно большим HDR диапазоном (что критически важно для современных игр), возвращает полноценный объект, сочетающий в себе всё необходимое для добавления в рендер (в описанные выше шаги отрисовки). На (рис. 7) приведены входные изображения и их получившиеся стилизации.

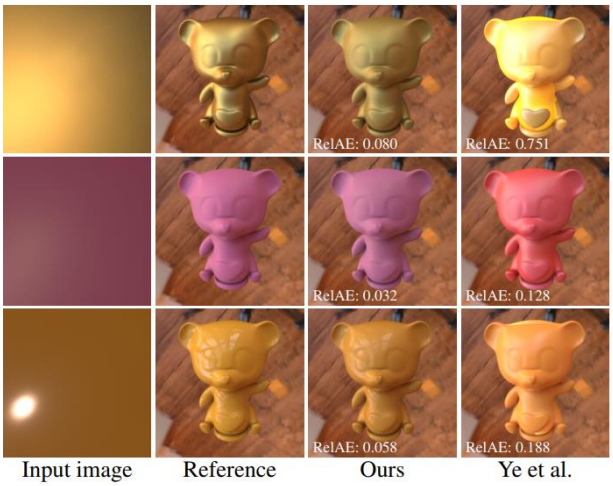

Рисунок 7 – Входные изображения и их стилизация.

Видно, как использование алгоритмов NN уже сегодня позволяет создавать не только материалы, но и готовые текстурные объекты, сохраняющие в себе все необходимые свойства для полноценного и качественного текстурирования.

Алгоритмы постобработки и глубокого обучения

Важной частью стилизации, придания единообразия и «выравненности» общей картине (завершённому кадру) является процесс Upscaling. В примере обработки кадра на (рис. 8) очень явно прослеживается переход от обычных методов (MSAA [multi-sampling anti-aliasing]), TSAA [temporal anti-aliasing], FXAA [fast approximate anti-aliasing]) к алгоритмам, использующим ML DLSS (Deep learning super sampling). Если предыдущие алгоритмы использовали попиксельную обработку на стороне шейдера в один из последних проходов кадра (так называемая постобработка), то DLSS от Nvidia использует методы ML (разработанный самой компанией convolutional autoencoder обученный на сети NGX, где выходное изображение сравнивалось с автономно визуализированным эталонным изображением сверхвысокого качества 16K, и разница от такого сравнения передавалась обратно в сеть. Такой процесс на суперкомпьютере повторяется десятки тысяч раз, пока сеть не выдаст изображения высокого качества в высоком разрешении. После завершения обучения такая сеть передаётся на компьютер вместе с драйвером, где может работать в режиме реального времени), выстраивая «очередь» из предыдущих кадров (в игре или ином видео процессе) и обрабатывая

получившийся массив кадров, по завершению возвращает изображения повышенного качества.

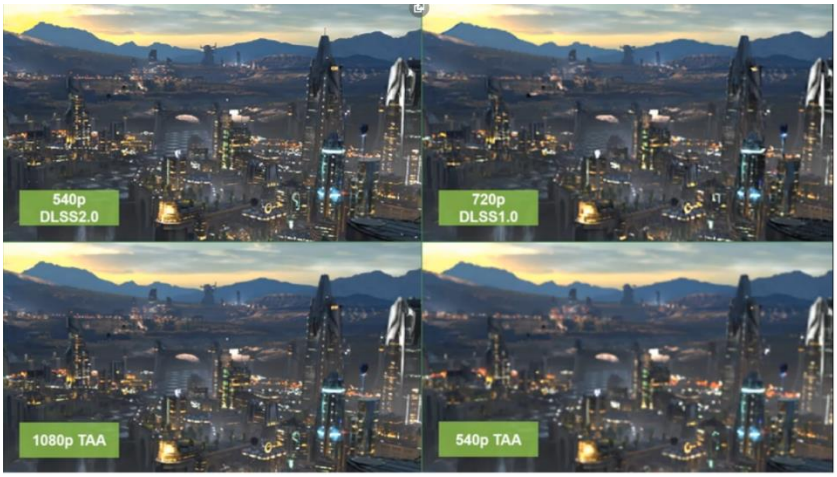

Рисунок 8 – Переход от MSAA, TSAA и FXAA к DLSS.

На (рис. 9) хорошо прослеживается замена устаревших методов на алгоритмы ML, позволяющие усовершенствовать входное изображение.

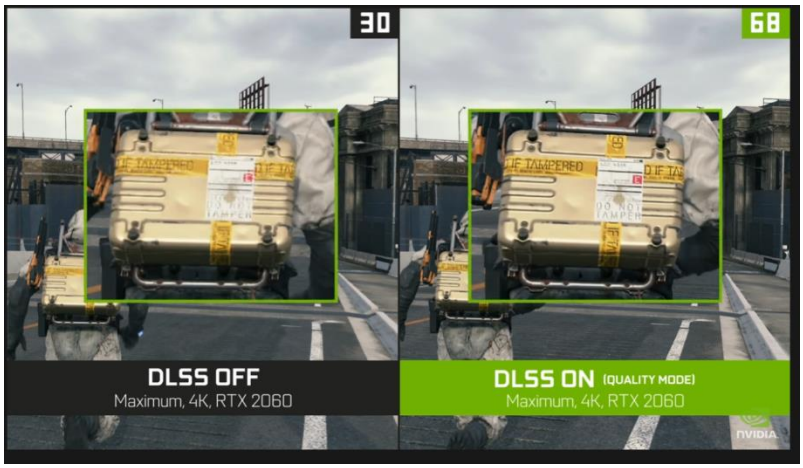

Рисунок 9 – Разница между устаревшими методами и алгоритмами ML.

Часто стилизация модели происходит не только с помощью материалов и текстур, но и с помощью эффектов, накладываемых на финальный кадр, когда все остальные «слои» рендеринга завершены. Они могут быть направлены не только на повышение чёткости или upscaling без потери качества, но и на иные эффекты (Bloor, Depth of Field, Ambient Occlusion), таким образом завершая некоторые эффекты материалов текстур (такая практика довольно распространена).

Стилизация игрового персонажа по входящей фотографии человека

Следующая возможность стилизации – применение алгоритмов NN для создания персонажей (модели и текстуры) по фотографии пользователя, на примере RPG-игр от компании NetEase Games. Здесь используется две фазы: в первой Generative Neural Network обучается как рендер (Imitation learning), во второй с помощью метода Gradient Descent Method определяется сходство лица, и по лекалам (если так можно сказать) создаётся фигура игрового персонажа, то есть происходит не просто создание текстуры, но и деформация заготовленной модели под входные контуры (рис. 10).

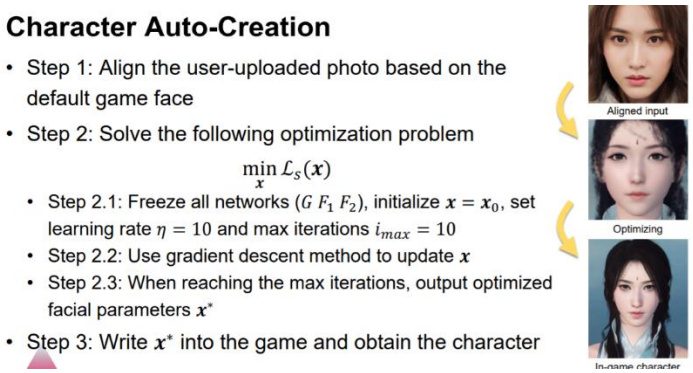

Рисунок 10 – Создание текстуры и деформации модели под входные контуры.

В первом шаге сначала определяются параметры лица (Facial Parameter) и прогоняем их через 8 Transposed Convolution Layers с прикреплёнными к ним Rectified Linear Unit (ReLU) и Batch Normalization (BN). После каждого прохода элемент увеличивается в 2 раза.

Во втором шаге процесс разбивается на две части: на Face Recognition Model F1 [15] и Face Segmentation Model F2 [14] для получения параметров Identity Loss (лямбда 1) и Content Loss (лямбда 2) соответственно. Для метода Face Recognition Model F1 используется натренированная нейронная сеть Light CNN-29v2 [15] принимающая исходное изображение размера 128x128 (сгенерированное на предыдущем шаге) и отдающая на выходе необходимую функцию (лямбда 1):

Definition:

$$\mathcal{L}1\,(\mathrm{x}) \;=\; 1 \;-\; \cos \;<\; \mathrm{F1}(\,\mathrm{yr})\,,\mathrm{F1}\;\mathrm{G}(\mathrm{x}))\;>, \tag{4}$$

Cosine distance:

$$cos\;(e1,e2) \;=<\; e1,e2\;>/\;||e1||\;2\;||e2\;||2\,, \tag{5}$$

Optimal object:

$$\min \mathrm{x}\;\mathcal{L}1(\mathrm{x}) \;-\; \text{лямбда}\;1\,, \tag{6}$$

Во втором методе (Face Semantic Segmentation Model F2) применяется Convolution Network с основой Resnet-50 [26], последний слой заменяется на Convolation Layer 1x1. В конце разрешение выходного изображения увеличивается с 1/32 до 1/8.

Базовое определение:

$$\mathcal{L}2\;(\mathrm{x}) \;=\; ||\;\mathrm{F2}\;(\mathrm{yr}) \;-\; \mathrm{F2}(\mathrm{G}(\mathrm{x1}))1, \tag{7}$$

Используются семантические карты вероятности для «уточнения» чёрт лица.

Оптимальное определение:

$$\min \mathrm{x}\;\mathcal{L}2(\mathrm{x}) \;-\; \text{лямбда}\;2, \tag{8}$$

На (рис. 11) виден результат работы нейросети.

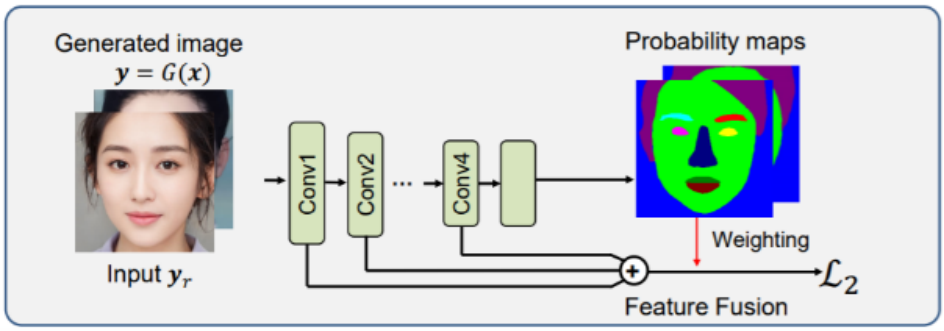

Рисунок 11 – Результат работы нейросети.

Функция общих потерь:

$$\mathcal{L}s = \alpha\mathcal{L}1 + \mathcal{L}2, \quad (9)$$

Оптимальная функция:

$$\min x\, \mathcal{L}s(x), \quad (10)$$

Gradient descent method

Обновление параметров:

$$xi = xi - 1 - \eta\,(\partial\mathcal{L}s\, xi - 1 / \partial xi - 1\,), i = 1,2,\dots, imax, \quad (11)$$

Вычисление градиента:

$$\partial\mathcal{L}s(x) / \partial x = \alpha\,(\partial\mathcal{L}1\,(x) / \partial x + \partial\mathcal{L}2\,(x) / \partial x,$$
$$\partial\mathcal{L}1(\mathrm{x}) / \partial \mathrm{x} = \partial\mathcal{L}1(\mathrm{x}) / \partial \mathrm{F}1(\mathrm{G}(\mathrm{x}))\ \partial \mathrm{F}1(\mathrm{G}(\mathrm{x})) / \partial \mathrm{G}(\mathrm{x})\ \partial \mathrm{G}(\mathrm{x}) / \partial(\mathrm{x}),$$
$$\partial\mathcal{L}2(\mathrm{x}) / \partial \mathrm{x} = \partial\mathcal{L}2(\mathrm{x}) / \partial \mathrm{F}2(\mathrm{G}(\mathrm{x}))\ \partial \mathrm{F}2(\mathrm{G}(\mathrm{x})) / \partial \mathrm{G}(\mathrm{x})\ \partial \mathrm{G}(\mathrm{x}) / \partial(\mathrm{x}), \quad (12)$$

На (рис. 12) первое изображение – фото пользователя, второе и третье – получившееся с помощью вышеописанного метода изображение.

Рисунок 12 – Получение персонажа с фотографии человека.

Это только основные шаги этого метода, чтобы наглядно проиллюстрировать, каким именно образом алгоритмы NN позволяют модифицировать текстуры и объекты на основе входных данных и получать приемлемый результат.

Разработка алгоритма стилизации текстуры трёхмерного объекта

Создаваемая программа будет использовать TensorFlow Graphics от компании Google. В этой статье будет рассмотрена его часть, отвечающая за работу света, материала, а также определения (detect) частей тела в трёхмерном объекте – и оценена возможность применения этого решения для стилизации уже сегодня.

Программа будет получать на вход простые данные – карту основного цвета (albedo) (рис. 13), которую будет стилизована с применением алгоритмов NN, доступных в пакете.

Рисунок 13 – Карта основного цвета.

За основу стилизации взята не совсем обычная идея: отображение максимизированной «потери» (loss) в виде градиента – на выходе получается нанесённая на исходное изображение карта градиентов. В данной статье используется предтренированная модель InceptionV3 [22], в которой есть объединенные слои (convolutions layers) – идущие по возрастанию от mixed0 до mixed10, и усиливающие «раздражители» от меньшего к большему – слои высокого порядка реагируют на раздражители более высокого порядка (явные элементы: глаза, впадины, отчётливо контрастные переливы). В обычных вариантах тренировки сетей NN преследуется цель уменьшения потерь с помощью метода градиентного спуска (gradient descent), но в этой задаче потери, наоборот, усиливаются с помощью градиентного подъёма (gradient ascent) и эти градиенты наносятся на первоначальную текстуру (рис. 14).

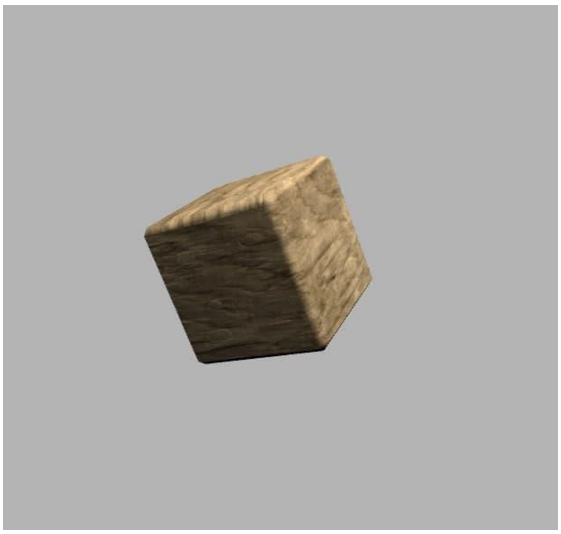

Рис.14 – Нанесение gradient ascent на первоначальную текстуру.

Именно таким образом выглядит изначальная текстура на трёхмерной модели (рис. 14).

С каждым последующим проходом карта будет получаться всё более стилизованной и узорчатой (рис. 15).

```
class DeepDream(tf.Module):
 def __init__(self, model):
 self.model = model

 @tf.function(
 input_signature=(
 tf.TensorSpec(shape=[None,None,3],
dtype=tf.float32),
 tf.TensorSpec(shape=[], dtype=tf.int32),
 tf.TensorSpec(shape=[], dtype=tf.float32),)
 )
def __call__(self, img, steps, step_size):
print("Calculated out images")
loss = tf.constant(0.0)
for n in tf.range(steps):
with tf.GradientTape() as tape:

tape.watch(img)
loss = calc_loss(img, self.model)

gradients = tape.gradient(loss, img)

gradients /= tf.math.reduce_std(gradients) +
1e-8

img = img + gradients*step_size
img = tf.clip_by_value(img, -1, 1)

return loss, img
```

Рисунок 15 – Карта после множества проходов алгоритма.

На (рис. 16) эта же модель, но уже в визуализированном виде. Вносимые изменения можно усиливать или, наоборот, ослаблять так, чтобы текстуры получившихся объектов не выглядели рваными. Можно использовать для этого алгоритмы этой же сети InceptionV3. Можно увеличить размер первоначальной картинки (так как размер потери напрямую зависит от размера исходного изображения).

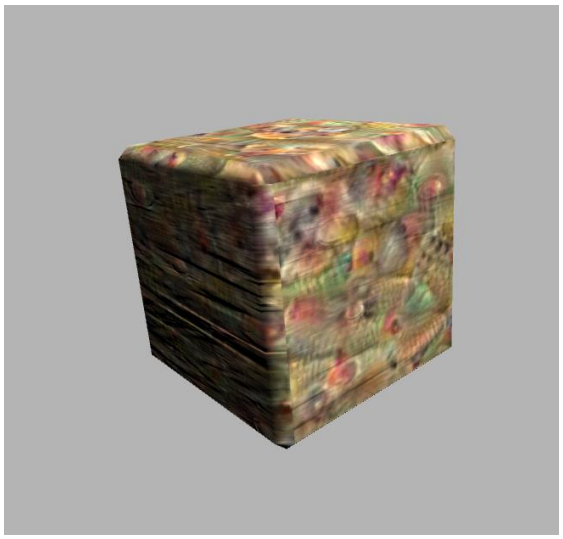

Рисунок 16 – Визуализация модели.

Сравним метод с предыдущими: как правило, на текстуру трёхмерных объектов, наносят такие эффекты стилизации за счёт заранее высчитанных сил GPGPU. (вычислительных сил шейдеров), чаще всего являющимися комбинацией наборов алгоритмов (Voronoi Diagram, Fast Fourier transform и прочих). Сейчас сравнивать результаты довольно проблематично, так как момент внедрения таких методов и алгоритмов в сферу практического использования только наступает.

Заключение

Дальнейшее направление исследований в данной области в первую очередь касается повышения качества генерируемой модели и уменьшения выборки датасета (набора данных) для создания предобученной нейронной сети. На сегодняшний день большая часть исследований включает поиск возможности включить больше переменных в генерируемые текстуры: не только свет, тени и некоторые их параметры, но и особенности и нюансы окружения (например

Height map для отображения глубины трещин и повреждений), которые также должны найти отражения в стилистике представленных трёхмерных моделей. Часть исследований, показанных выше, работают над иными аспектами применения алгоритмов NN в визуализации, в частности, в стилизации трёхмерных моделей (DLSS, Deep BRDF, Charactered Auto).

Можно сказать, что дальнейшее развитие исследуемых проблем приведёт к однозначному замещению привычных решений с участием человека (например подготовка первоначальных карт), и будет происходить исключительно с использованием алгоритмов нейронных сетей.

References:

1. (n.d.). Retrieved from https://blog.selfshadow.com/2020/08/18/siggraph-2020-links/
2. (n.d.). Retrieved from https://semantic-pyramid.github.io
3. (n.d.). Retrieved from http://reality.cs.ucl.ac.uk/projects/btf/rainer2020unified.html
4. (n.d.). Retrieved from https://hal.archives-ouvertes.fr/hal-01678122v2/document
5. (n.d.). Retrieved from https://www.gdcvault.com/play/1026713/Machine-Learning-Physics-Simulation-Kolmogorov
6. (n.d.). Retrieved from https://drive.google.com/drive/folders/1TM5oz4Rr2ji8mVzmMzrHMJlPg9k1pdud
7. (n.d.). Retrieved from https://developer.download.nvidia.com/video/siggraph/2020/presentations/sig01-compositional-neural-scene-representations-for-shading-inference.pdf
8. (n.d.). NVIDIA DLSS 2.0: A Big Leap In AI Rendering
9. (n.d.). Retrieved from http://reality.cs.ucl.ac.uk/projects/btf/rainer19neural.pdf
10. (n.d.). Retrieved from https://team.inria.fr/graphdeco/projects/multi-materials/
11. (n.d.). Retrieved from https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s21764-3d-deep-learning-in-function-space.pdf
12. Tsirikoglou, A., Eilertsen, G., & Unger, J. (2020). *A Survey of Image Synthesis Methods for Visual Machine Learning*, p.26.
13. (2018). Single-Image SVBRDF Capture with a Rendering-Aware Deep Network (arxiv.org), p.15.
14. (n.d.). Retrieved from https://www.gdcvault.com/play/1026581/Face-to-Parameter-Translation-via
15. (n.d.). Retrieved from https://github.com/AlfredXiangWu/LightCNN
16. Yair, R., Yibo, Z., Harun, G., Da, T., & Aydogan, O. (2017). *Phase recovery and holographic image reconstruction using deep learning in neural networks*. (p.9).
17. (n.d.). *TensorFlow Graphics.* Retrieved from https://www.tensorflow.org/graphics
18. (n.d.). *GPU Pro.* Retrieved from http://gpupro.blogspot.com/
19. Matusik, W., Pfister, H., Brand, M., & Mcmillan, L. (2003). *A data-driven reflectance model.* ACM Transactions on Graphics, (p.12).
20. Deschaintre, V., Aittala, M., Durand, F., Drettakis, G., & Bousseau, A. (2019). *Flexible SVBRDF Capture with a Multi-Image Deep Network*, (p.14).
21. (n.d.). *Image-to-Image Translation with Conditional Adversarial Networks.* Retrieved from https://arxiv.org/pdf/1611.07004v1.pdf
22. (n.d.). Retrieved from https://keras.io/api/applications/#inceptionv3
23. (n.d.). Retrieved from http://rtintro.realtimerendering.com
24. (n.d.). Retrieved from https://arxiv.org/abs/1503.03167#:~:text=This%20paper%20presents%20the%20Deep,an%20interpretable%20representation%20of%20images.&text=Given%20a%20single%20input%20image,variations%20in%20pose%20and%20lighting.
25. (n.d.). *BENGIO* Y.: Learning deep architectures for AI. Retrieved from https://www.researchgate.net/publication/215991023_Learning_Deep_Architectures_for_AI/link/546cd25f0cf26e95bc3ca67a/download
26. (n.d.). Retrieved from https://docs.exponenta.ru/deeplearning/ref/resnet50.html