ПРОГРАММИРОВАНИЕ, 2019, № 4, с. 54–63

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.921

УРОВЕНЬ ДЕТАЛИЗАЦИИ ДЛЯ ПРЕДРАСЧИТАННЫХ ПРОЦЕДУРНЫХ ТЕКСТУР

© 2019 г. В. В. Санжаров[a,*], В. А. Фролов[b,c,]**

[a] Российский государственный университет нефти и газа (Национальный исследовательский университет) имени И.М. Губкина 119296 Москва, Ленинский пр., д. 65, Россия

[b] Институт прикладной математики имени М.В. Келдыша РАН 125047 Москва, Миусская пл., д. 4, Россия

[c] Московский государственный университет имени М.В. Ломоносова 119899 Москва, Ленинские горы, д. 1, стр. 8, Россия

**e-mail: vs@asugubkin.ru*

***e-mail: vfrolov@graphics.cs.msu.ru*

Поступила в редакцию 16.02.2019 г.
После доработки 16.02.2019 г.
Принята к публикации 18.03.2019 г.

В данной работе рассматривается проблема определения уровня детализации для предрасчитанных процедурных текстур в фотореалистичном рендеринге. Предлагается решение в виде быстрого предварительного прохода визуализации сцены с вычислением уровня детализации алгоритмом, основанным на mip-текстурировании. Предложенный подход протестирован для вычисления разрешения процедурных текстур разных видов и обыкновенных текстур-изображений.

DOI: 10.1134/S0132347419040071

1. ВВЕДЕНИЕ

Процедурная генерация – одна из основных технологий в компьютерной графике применительно к области визуальных эффектов в кинопроизводстве, поскольку она позволяет снизить стоимость ручного создания трехмерных сцен и их компонентов. Также высока значимость такого рода решений и при использовании фотореалистичного синтеза изображений и видеопоследовательностей с целью обучения алгоритмов искусственного интеллекта в компьютерном зрении [1].

Процедурный подход в моделировании материалов в первую очередь основан на синтезе текстурных карт, которые в дальнейшем используются для задания распределенных в пространстве параметров модели материалов, геометрической модели (например, карты смещения). Методы генерации процедурных текстур традиционно делят на явные и неявные [2]. Явные методы заранее (т.е. до непосредственно рендеринга) вычисляют всю текстуру целиком. Эти текстуры затем могут быть использованы при расчете освещения как изображения. Неявные методы определяют текстуру как функцию от некоторых аргументов, например, координаты в текстурном или мировом пространстве, и затем могут быть вызваны рендер-системой в процессе расчета освещения по необходимости. Другими словами, алгоритм отвечает некоторым значением на запрос выборки из текстуры. Неявный подход тем самым подразумевает создание некоторого алгоритмического описания желаемого визуального результата.

2. ОБЗОР ТЕХНОЛОГИЙ СИНТЕЗА ПРОЦЕДУРНЫХ ТЕКСТУР

Неявный подход в синтезе процедурных текстур используется уже довольно давно, – одним из первых подобных алгоритмов является шум Перлина [3], предложенный в 1985 г., и продолжает развиваться и сегодня (например, [4–6]). Неявные методы обладают рядом ощутимых преимуществ. Во-первых текстурные карты не хранятся в памяти, что актуально в свете большого размера современных 3d-сцен – например, сырые данные для расчета одного кадра в сцене Moana Island, выложенной компанией Disney в открытый доступ [7], занимают 93 Гб. Во-вторых, у неявных процедурных текстур нет фиксированного разрешения (оно потенциально бесконечное) и поэтому отсутствуют швы. Кроме того, алгоритмы синтеза могут быть параметризованы произвольными переменными (например, позицией в мировом пространстве) и применены к геометрическим моделям, которые не имеют тек-

стурной развертки. А изменяя входные параметры алгоритма, можно получить большое количество различных вариантов результирующей текстуры. Главным недостатком неявного подхода является сложность процесса алгоритмической формализации и необходимость тщательной настройки параметров для достижения желаемого визуального результата. Также в некоторой степени к недостаткам относится независимое вычисление выборок из текстуры, что ограничивает возможные визуальные результаты и усложняет процедуру анти-алиасинга. Несмотря на эти недостатки, неявные подходы широко используются и реализованы в большинстве распространенных программных продуктов для создания 3d-сцен (Houdini, Blender, Maya, 3ds Max и др.).

В случае явных алгоритмов синтеза процедурных текстур, значения текселей влияют друг на друга и тем самым не могут быть рассчитаны независимо. Алгоритмы явного текстурного синтеза преимущественно ориентированы на создание текстуры на основе изображения-примера желаемого результата и могут быть разделены на три группы – статистические, на основе перераспределения частей (патчей) текстуры и гибридные. Подробный обзор ранних явных методов представлен в [8]. Современное развитие данного подхода ориентировано как на улучшение имеющихся алгоритмов [9, 10], применение нейронных сетей [11, 12], а также комбинирование этих двух направлений [13, 14]. Одно из основных достоинств явного подхода по сравнению с неявным заключается в том, что нет необходимости в формальном описании желаемого визуального результата, который задается в виде изображения-примера. Общей проблемой явных методов является значительный объем вычислений, который напрямую зависит от разрешения выходной текстуры. Алгоритм [11], по словам авторов, вычисляет текстуру в разрешении 256 на 256 пикселей за 10 минут на профессиональной видеокарте NVIDIA Tesla K40. В работе [15] сообщается, что время вычислений для различных алгоритмов текстурного синтеза на основе изображения-примера может достигать нескольких часов для больших разрешений. Кроме того, каждая из групп методов синтеза текстур на основе примера также обладает своими особенностями и проблемами, относящимися к качеству полученной текстуры и ее близости к поданному на вход примера. Эта тема подробно рассмотрена в [16], где проводится сравнение классических и современных нейросетевых подходов.

Существующие проблемы. Разрешение текстуры оказывает влияние на занимаемый размер в памяти, время, необходимое для выполнения алгоритма синтеза, и на качество изображения при рендеринге. Таким образом, для фотореалистичного синтеза изображений значительную роль играет задача определения такого разрешения текстур, которое позволит получить максимальное визуальное качество при минимальной занимаемой памяти и минимальном объеме вычислений.

Другая задача связана с часто возникающей на практике необходимостью поддерживать существующие реализации методов неявного текстурного синтеза в проприетарных программных продуктах, в первую очередь в 3d-редакторах, как, например, Maya или 3ds Max. В типичном случае это означает, что рендер-системе необходимо вызывать виртуальную функцию 3d-редактора для получения значения выборки из текстуры. Однако, если адресное пространство, в котором работает рендер-система, отличается от адресного пространства 3d-редактора, то подобный вызов функции в процессе рендеринга становится невозможным. Подобная ситуация возникает, например, если рендер-система работает на графическом процессоре или на другом компьютере в сети. Таким образом, для поддержки функциональности таких процедурных текстур, разработчикам рендер-системы нужно использовать другой подход. Один из вариантов – реализация аналогичных алгоритмов процедурного синтеза текстур в рендер-системе и сопоставление их параметров с таковыми в программном обеспечении, в которое производится интеграция рендер-системы. Однако, поскольку исходные коды алгоритмов текстурного синтеза в 3d-редакторе неизвестны (а в большинстве случаев отсутствуют и ссылки на описание алгоритмов), подобное решение нельзя назвать “поддержкой существующей функциональности редактора” в полном смысле, а только ее *эмуляцией*. Кроме того, для каждого из программных продуктов, в который интегрируется рендер-система, необходимо будет заново повторять реализацию его специфических процедурных текстур. Второй вариант – рассматривать неявные методы текстурного синтеза, которые необходимо поддержать, как явные – то есть производить предварительный расчет текстур, используя имеющуюся реализацию в 3d-редакторе. В этом случае мы приходим к первой обозначенной в данном разделе задаче – определению необходимого разрешения текстур.

В данной работе предлагается подход для оценки разрешения для предварительного расчета процедурных текстур в фотореалистичном синтезе изображений. Предлагаемый подход также может быть использован для выбора разрешения текстур, заданных в виде изображений с целью уменьшения затрат памяти при рендеринге.

3. ПРЕДЫДУЩИЕ РАБОТЫ

Существующие подходы в большинстве случаев ориентированы на вычисление уровня детализации (и тем самым разрешения) для текстур за-

данных изображениями для рендер-систем, работающих в реальном времени. Типичные решения задачи выбора разрешения текстур основаны на mip-текстурировании [17] и подразумевают хранение нескольких копий изображения с разными разрешениями, mip-пирамиды, из которой подходящий mip-уровень выбирается в процессе рендеринга, например, на основе размера спроецированного на экран текстурируемого объекта. Есть методы, позволяющие проводить параллельный синтез разных mip-уровней для некоторых явных процедурных текстур [18–20]. Некоторые из существующих подходов нацелены на выполнение сравнительно небольшого числа предварительных вычислений для ускорения дальнейшего синтеза всей mip-пирамиды [21]. Усовершенствование mip-текстурирования, называемое clipmapping [22] основано на том, что для большой текстуры не все данные в mip-пирамиде используются при рендеринге конкретного кадра, поэтому достаточно хранить только те части, которые являются видимыми. Более сложный подход – виртуальное текстурирование [23–25] основано на равномерном разбиении текстур на тайлы и загрузке в память только тех тайлов, которые нужны для расчета конкретного кадра, также на основе информации о видимости. Виртуальное текстурирование получило аппаратную поддержку на GPU [26] и доступно в Vulkan API и как расширение ARB_SPARSE_TEXTURE OpenGL API [27].

Ограничения существующих подходов. Существующие решения для оценки разрешения текстур в основном ориентированы на рендеринг в реальном времени. Также, поскольку поставленная задача заключается в оценке разрешения для предрасчитанных процедурных текстур *до начала рендеринга*, существующие подходы сами по себе не позволяют ее решить. Кроме того, реализация любого из существующих подходов предполагает тесную интеграцию с расчетным ядром рендер-системы и может потребовать значительных изменений в нем. Например, если меши могут иметь больше одной назначенной текстуры, то использование виртуального текстурирования потребует реализации текстурных атласов [28]. Поскольку переписывание кода уже существующих и стабильно работающих частей сложной системы обычно является нежелательным, мы считаем это недостатком для прямой реализации рассмотренных решений в некоторой существующей рендер-системе. Среди других возможных проблем имеющихся решений можно выделить возможное возникновение визуальных артефактов при фильтрации текстур [24, 25]. Для некоторых видов текстурных карт, например, карт шероховатости поверхности и карт нормалей, фильтрация может привести к значительным потерям в детальности [29], что является недопустимым при фотореалистичном рендеринге, так как детальность поверхностей объектов — одна из основ реалистичного изображения.

4. ПРЕДЛАГАЕМЫЙ ПОДХОД

В основе предлагаемого решения лежит выделение всех необходимых предварительных вычислений из рендер-системы в отдельный этап быстрой отрисовки сцены, который может быть выполнен с помощью растеризации или трассировки лучей, за которым следует синтез текстур. На этом предварительном этапе определяется разрешение каждой текстуры в сцене, исходя из mip-уровней, которые рассчитываются с использованием модифицированной реализации подхода, предложенного в [25], и подхода, представленного в спецификации OpenGL API [27]. Далее рассмотрим предложенное решение подробнее.

4.1. Описание алгоритма

В первую очередь необходимо сохранить указатели на функции, реализующие расчет процедурных текстур, использующихся в текущей сцене, а также значения их входных параметров. Эта информация будет использована в дальнейшем для непосредственного синтеза текстур после определения требуемого разрешения.

Следующий этап заключается в отрисовке геометрии сцены с информацией о назначенных материалах и текстурных координатах. Из полученных таким образом изображений можно определить градиент текстурных координат для всех текстурированных объектов, а на основе градиента и mip-уровень. Выражения для вычисления градиента (4.1), (4.2) схожи с таковыми для вычисления mip-уровня в [25] и спецификации OpenGL API [27].

$$G_x(uv) = \frac{R\_p_x}{R\_r_x} \times tex\_res \times \frac{\partial u}{\partial x} \tag{4.1}$$

$$G_y(uv) = \frac{R\_p_y}{R\_r_y} \times tex\_res \times \frac{\partial v}{\partial y}, \tag{4.2}$$

где R_p_x, R_p_y – разрешение изображения для предварительной отрисовки, R_r_x, R_r_y – разрешение изображения для фотореалистичного рендеринга, tex_res – разрешение текстуры, u, v – текстурные координаты, x, y – координаты в пространстве изображения, полученного при предварительной отрисовке.

Однако, для процедурных текстур величина tex_res является искомой. Предположим, что она представляет собой некоторое произвольное большое разрешение, которое требуется уменьшить.

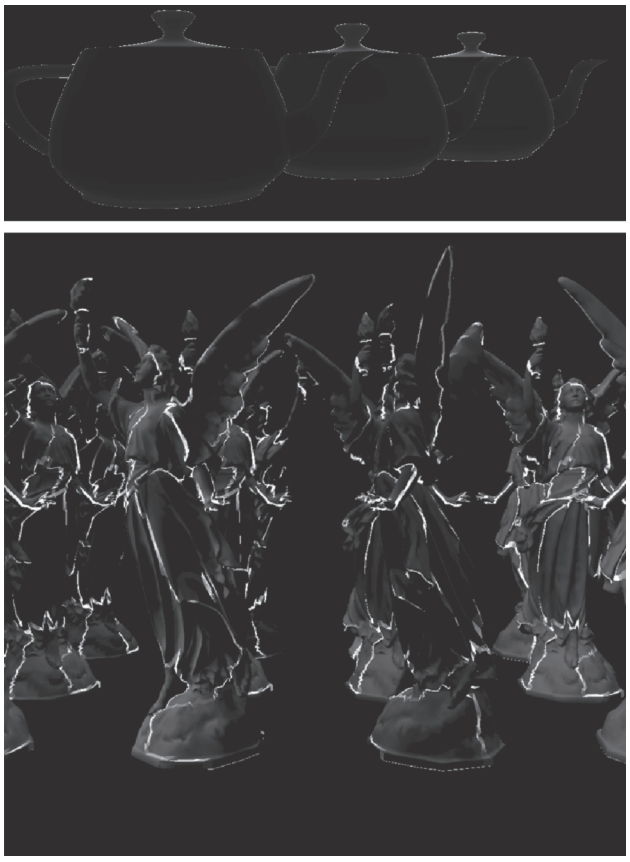

Рис. 1. Визуализация модуля градиента текстурных координат.

Также следует учесть соотношение между разрешением предварительной отрисовки (R_p_x, R_p_y) и разрешения, в котором будет производиться основной, фотореалистичный рендеринг (R_r_x, R_r_y). Значения этих разрешений могут отличаться, т.к. предварительная отрисовка не требует высокого разрешения, в то время как фотореалистичный рендеринг, очевидно, может быть осуществлен в произвольном разрешении.

После расчета градиентов, они используются для вычисления mip-уровня *mip_lvl* (4.3), опять же аналогично [27] и [25]:

$$mip\_lvl = \log_2[\max(G_x, G_y)] \tag{4.3}$$

Необходимое разрешение текстуры R тогда может быть рассчитано как (4.4):

$$R = \frac{tex\_res}{2^{mip\_lvl}} \tag{4.4}$$

Подставив выражения для вычисления градиентов из (4.1), (4.2) в (4.3), а затем результирующее выражение для *mip_lvl* в (4.4), получим следующее (4.5):

$$R = \frac{tex\_res}{\max\left(\frac{R\_p_x}{R\_r_x} \times \frac{tex\_res}{\frac{\partial u}{\partial x}}, \frac{R\_p_y}{R\_r_y} \times \frac{tex\_res}{\frac{\partial v}{\partial y}}\right)} \tag{4.5}$$

Легко заметить, что *tex_res* можно сократить, после чего получим финальное уравнение для разрешения (4.6):

$$R = \max\left(\frac{R\_r_x}{R\_p_x} \times \frac{1}{\frac{\partial u}{\partial x}}, \frac{R\_r_y}{R\_p_y} \times \frac{1}{\frac{\partial v}{\partial y}}\right) \tag{4.6}$$

При реализации алгоритма, нужно определить минимальный mip-уровень (т.е. соответствующий наибольшему разрешению) для каждой текстуры в сцене – одни и те же текстуры могут иметь разные mip-уровни при использовании в разных материалах и на разных мешах. Тем самым мы оставляем только один уровень из mip-пирамиды для каждой текстуры. Далее остается только передать полученное разрешение в сохраненные функции синтеза текстур. Такая же процедура может быть применена к текстурам заданным как изображения для изменения их размера до достижения соотношения 1 к 1 между пикселем экрана и текселем с целью экономии памяти. Для разных видов текстур рекомендуется задавать разные максимально допустимые mip-уровни, например, чтобы генерировать карты отражений и смещения в более высоком разрешении, чем текстуры для диффузного цвета.

Следует отметить, что рендер-система может выполнять масштабирование текстур в соответствии с рассчитанными предложенным подходом разрешениями только в том случае, когда есть необходимость в экономии памяти.

4.2. Способы вычисления производной текстурных координат

В финальном выражении для разрешения текстуры (4.6) присутствуют частные производные текстурных координат по координатам полученного на этапе предварительной отрисовки сцены изображения – $\frac{\partial u}{\partial x}, \frac{\partial v}{\partial y}$. Существует несколько способов вычисления данных величины.

В случае реализации предварительного этапа как растеризации с помощью одного из графических API (например, OpenGL или Vulkan), можно воспользоваться доступными во фрагментном шейдере функциями *dFdx* и *dFdy*, позволяющими использовать аппаратные средства для вычисления значений частных производных переданного им параметра (в нашем случае – текстурных координат) по координатам в пространстве изображения. В этом случае частные производные вычисляются как

Рис. 2. Тестовые сцены – Arch, Bathroom, Alley. Arch и Alley – преимущественно процедурные текстуры, Bathroom – текстуры-изображения высокого разрешения.

разности значений дифференцируемого параметра в текущем фрагменте и его ближайших соседях [27]. Если предварительный этап реализован через растеризацию без использования графических API или с помощью трассировки лучей, то аналогичные вычисления могут быть реализованы разработчиком, при условии сохранения в каждом пикселе рассчитанного изображения информации о значении дифференцируемого параметра.

Следует отметить, что как при использовании аппаратных функций *dFdx* и *dFdy*, так и в случае собственной реализации описанного способа, предполагается, что дифференцируемая величи-

Таблица 1. Память, занимаемая текстурами

Сцена, разрешение	Эталон, Мбайт	Предложенный метод, Мбайт	Отношение
Arch, 1920 × 1080	276	156	1.77
Arch, 3840 × 2160	276	173	1.60
Bathroom, 1024 × 1024	521	142	3.67
Bathroom, 2048 × 2048	521	178	2.93
Alley, 1024 × 1024	527	129	4.09
Alley, 2048 × 2048	527	146	3.61

на является непрерывной. На практике же будут появляться разрывы, например, на границах объектов (рис. 1, вверху). Или же, например, высокие значения производных на текстурных швах (рис. 1, снизу). Однако, поскольку в предложенном решении происходит поиск минимального значения mip-уровней, высокие значения производной на различного рода границах будут отброшены и не повлияют на вычисление финального результата.

Второй подход вычисления частных производных текстурных координат основан на трассировке лучей с вычислением так называемых дифференциалов луча [30, 31]. Идея метода заключается в том, что для каждого луча, трассируемого из виртуальной камеры, создается два вспомогательных луча со сдвигом в один пиксель по горизонтали и по вертикали. При этом все пересечения с геометрическими объектами вычисляются только для основного луча. Дополнительные лучи используются только для оценки частных производных, например, мировых или текстурных координат геометрического объекта, с которым было найдено пересечение основного луча. При этом принимается допущение, что поверхность геометрического объекта локально плоская. Таким образом, точки пересечения вспомогательных лучей могут быть вычислены как лежащие на плоскости, заданной точкой пересечения основного луча и вектором нормали в этой точке. Для сильно искривленных поверхностей и на границах объектов подобное допущение может иметь большую погрешность, но при этом гарантирует отсутствие разрывов.

Метод дифференциалов луча может быть использован при реализации предварительного расчета изображения как с помощью трассировки лучей, так и с помощью растеризации. В случае растеризации направление луча из виртуальной камеры в конкретном пикселе изображения может быть получено по следующему алгоритму 4.7:

$$
\begin{gathered}
P1.x := \frac{2\times(x+0.5)}{w} - 1 \\
P1.y := \frac{-2\times(y+0.5)}{h} + 1 \\
P1.z := 0 \\
P1.w := 1 \\
P2 := (mView \times mProj)^{-1} \times P1 \\
P2 := \frac{P2}{P2.w} \\
D := \frac{D}{\|D\|},
\end{gathered}
\tag{4.7}
$$

где x, y – координаты пикселей, w, h – ширина и высота изображения, $mView$ – модельно-видовая матрица, $mProj$ – матрица проекции.

4.3. Ограничения и частные случаи

Отдельно следует рассмотреть несколько частных случаев, которые требуют внесения изменений в предложенный подход – наличие прозрачных объектов и текстурированных объектов, видимых в отражениях. Необходимые изменения будут отличаться в зависимости от используемого способа расчета частных производных текстурных координат.

Рассмотрим сначала случай первого варианта вычисления производных. При присутствии в сцене прозрачных объектов для корректной оценки разрешения текстур при вычислении первым из рассмотренных выше методов необходимо отдельно обрабатывать прозрачные и непро-

Рис. 3. Эталон и изображения с масштабированными по предложенному алгоритму текстурами, сцена Bathroom.

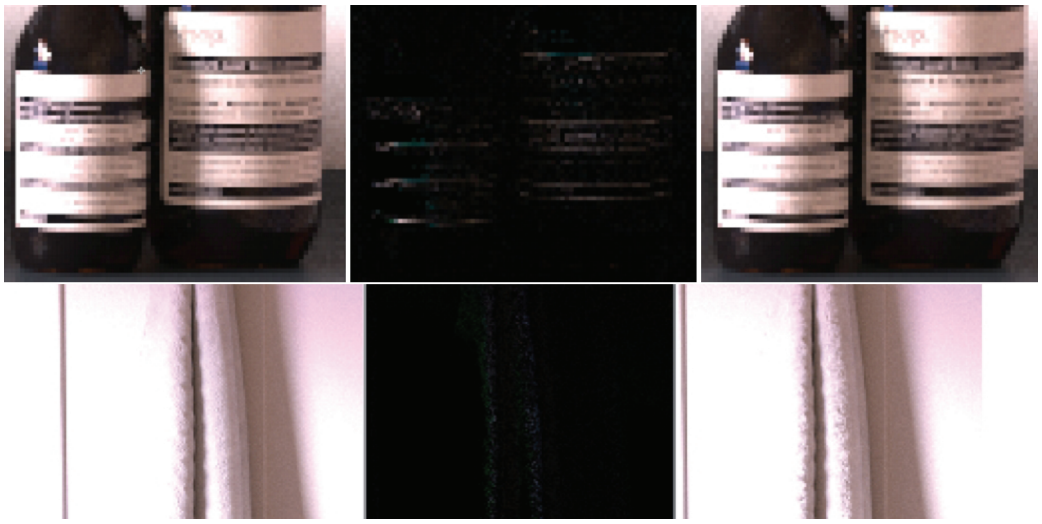

Рис. 4. Слева – эталон, справа – изображение с масштабированными по предложенному алгоритму текстурами, по центру – разница этих изображений. Сцена Bathroom, в верхнем ряду – текстура-изображения для диффузного цвета, в нижнем ряду – карта нормалей расчитанная из текстуры-изображения.

Рис. 5. Слева – эталон, справа – изображение с масштабированными по предложенному алгоритму текстурами, по центру – разница этих изображений. Сцена Alley, процедурная текстура на основе шума на объекте с неравномерным масштабированием текстурных координат.

зрачные объекты. Например, сначала отрисовать все непрозрачные объекты и вычислить для них уровень детализации текстур, затем последовательно отрисовывать прозрачные объекты и выполнять вычисления для них. При этом возможные преломления в прозрачных объектах не будут учтены. В ситуации с текстурированными объектами, которые видимы в отражениях (возможно после многократных переотражений и преломлений) возможно только эвристическое решение – например, предположить, что отраженный объект не будет больше чем разрешение экрана и задать разрешение текстур на данном объекте соответствующим образом. Другая возможная эвристика – назначать отраженным объектам уровень детализации, полученный для наибольшего отражающего объекта (зеркала) в сцене.

Если же частные производные рассчитываются вторым из рассмотренных выше способов, то оба частных случая могут быть решены более эффективно, т.к. при трассировке дифференциалов луча возможно также рассчитать отражения и преломления для вспомогательных лучей [30, 31]. Это можно считать значительным преимуществом подхода на основе дифференциалов луча. Однако, в этом случае необходимо сам этап предварительного расчета реализовывать на основе трассировки лучей.

Рис. 6. Сцены с явными процедурными текстурой, слева – сгенерированной алгоритмом [34] на ковре, справа – сгенерированной алгоритмом [35] на стульях.

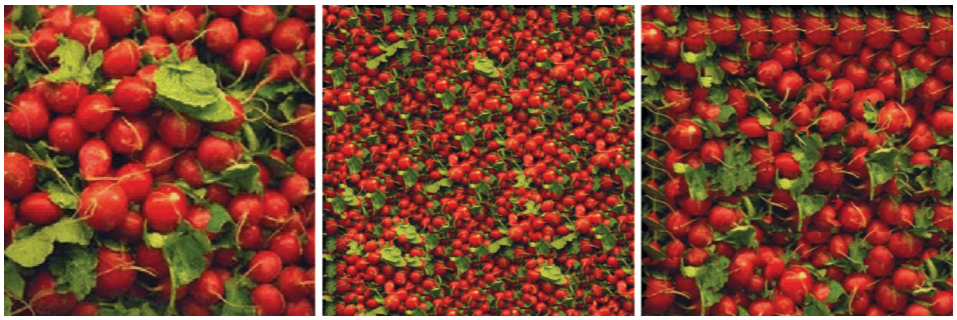

Рис. 7. Слева – базовая текстура, по центру – синтезированная [34] в разрешении 600 на 600 (исходное разрешение в сцене), справа – синтезированная [34] в разрешении 256 на 256 (разрешение полученное предложенным методом).

5. ВЫВОДЫ И РЕЗУЛЬТАТЫ

Тестирование предложенного решения было проведено на нескольких сценах с процедурными текстурами и текстурами-изображениями. Фотореалистичный рендеринг производился с помощью рендер-системы Hydra ([32]) на GPU.

В первом эксперименте были взяты 3 сцены (рис. 2), в которых текстуры-изображения были заданы в их исходном разрешении и процедурные текстуры в разрешении, выбранном вручную 3d-художником. Результат расчета данных сцен был принят за эталонный. На следующем этапе первого эксперимента к тем же самым сценам был применен предложенный в данной работе подход, реализованный в виде предварительного расчета на основе растеризации, для определения разрешения текстур. После чего процедурные текстуры синтезировались в рассчитанном разрешении, а текстуры-изображения масштабировались. Далее был произведен фотореалистичный рендеринг сцен и проводилось сравнение с эталонными результатами.

Таблица 2. Среднеквадратическая ошибка (mean squared error, MSE) между изображением-эталоном и изображением, полученным после изменения разрешения текстур

Сцена	MSE
Alley	1.67
Bathroom	3.59
Arch	3.02

Сравнение показало уменьшение затрат памяти на текстуры в несколько раз при использовании предложенного подхода (табл. 1). При этом визуальные отличия с эталонными изображениями практически отсутствуют (рис. 3, 4, 5), что подтверждается расчетом среднеквадратической ошибки между изображениями эталонов и полученных с использованием предложенного подхода (табл. 2).

Рис. 8. Слева – базовая текстура, по центру – синтезированная [35] в разрешении 256 на 256 (исходное разрешение в сцене), справа – синтезированная [35] в разрешении 128 на 128 (разрешение полученное предложенным методом).

Во втором эксперименте были взяты 2 сцены (рис. 6), где отдельные текстуры были назначены, как явные процедурные, использующие два алгоритма расчета:

• алгоритм Image Quilting [33], основанный на перераспределении частей (патчей) текстуры, который показывает хорошие результаты и в современных сравнениях [15];

• алгоритм DeepTextures на основе сверточных нейронных сетей [11].

Для расчетов использовались реализации алгоритмов синтеза текстур от их авторов, находящиеся в открытом доступе [34, 35]. Сначала производился синтез текстур в разрешении, заданном в соответствии с разрешением выбранных 3d-художником исходных текстур. Затем разрешение текстур вычислялось с помощью предложенного алгоритма. После чего было произведено сравнение времени синтеза текстур, показавшее выигрыш от 2 до 3.5 раза (табл. 3). Сравнение самих рассчитанных изображений в данном случае не имеет смысла, т.к. алгоритмы синтеза текстур генерируют разные изображения при изменении входных параметров (в том числе и желаемого разрешения) (рис. 7, 8).

5.1. Заключение

Таким образом, предложенный подход позволяет обеспечить экономию памяти, требуемую для текстур, от 1.5 до 4 раз и времени, необходимого для синтеза явных текстур, от 2 до 3.5 раза по сравнению с наивным подходом, при котором текстуры синтезируются в некотором фиксированном разрешении (заданном, например, исходя из разрешения рендеринга), а текстуры-изображения используются в своем исходном разрешении.

Таблица 3. Время синтеза текстур в разном разрешении

Алгоритм	Разрешение	Время синтеза, сек.
Image Quilting	600 × 600	537
Image Quilting	256 × 256	270
Deep Textures	128 × 128	1031
Deep Textures	256 × 256	3684

СПИСОК ЛИТЕРАТУРЫ

1. *Tsirikoglou A., Kronander J., Wrenninge M., Unger J.* Procedural Modeling and Physically Based Rendering for Synthetic Data Generation in Automotive Applications // arXiv:1710.06270v2, 2017
2. *Ebert D.S., Musgrave F.K., Peachey D., Perlin K., and Worley S.* Texturing and Modeling: A Procedural Approach // Morgan Kaufmann, 1998.
3. *Perlin K.* An Image Synthesizer // Association of Computing Machinery's Special Interest Group on Computer Graphics and Interactive Techniques. 1985. P. 287–296.
4. *Lefebvre L., Poulin P.* Analysis and synthesis of Structural Textures // Proceedings of Graphics Interface 2000. P. 77–86.
5. *Maung D., Shi Y., Crawfish R.* Procedural textures using tilings with Perlin Noise // 17th International Conference on Computer Games (CGAMES). 2012. P. 60–65.
6. *Gilet G., Sauvage B., Vanhoey K., Dischler J.-M., Ghazanfarpour D.* Local random-phase noise for procedural texturing // Association of Computing Machinery's Transactions on Graphics. 2014. V. 33. № 6. Article 195.
7. Сцена Moana Island. https://www.technology.disneyanimation.com/ islandscene
8. *Wei L.-Y., Lefebvre S., Kwatra V., Turk G.* State of the Art in Example-based Texture Synthesis // Eurographics 2009, State of the Art Report, EG Association (2009). P. 93–117.
9. *Kaspar A., Neubert B., Lischinski D., Kopf J.* Self Tuning Texture Optimization // Computer Graphics Forum. 2015. V. 34. № 2. P. 349–359.
10. *Jamriska O., Fiser J., Asente P., Lu J., Shechtman E., Sykora D.* LazyFluids: appearance transfer for fluid animations // Association of Computing Machinery's Transactions on Graphics. 2015. V. 34. № 4. Article 92.

11. *Gatys L.A., Ecker A.S., Bethge M.* Texture Synthesis Using Convolutional Neural Networks // arXiv:1505.07376, 2015
12. *Li Y., Fang C., Yang J., Wang Z., Lu X., Yang M.-H.* Diversified Texture Synthesis with Feed-Forward Networks // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. P. 266–274.
13. *Li C., Wand M.* Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis // arXiv:1601.04589, 2016.
14. *Dong J., Wang L., Liu J., Sun X.* A Procedural Texture Generation Framework Based on Semantic Descriptions // arXiv:1704.04141. 2017.
15. *Kolar M., Debattista K., Chalmers A.* A Subjective Evaluation of Texture Synthesis Methods // Computer Graphics Forum. 2017. V. 36. P. 189–198.
16. *Raad L. et al.* A survey of exemplar-based texture synthesis // arXiv preprint arXiv:1707.07184. 2017.
17. *Williams L.* Pyramidal parametrics // Association of Computing Machinery's Special Interest Group on Computer Graphics. 1983. V. 17. № 3. P. 1–11.
18. *Wei L., Levoy M.* Order-independent texture synthesis // Technical Report TR-2002-01. Computer Science Department, Stanford University. 2013.
19. *Lefebvre S., Hoppe H.* Parallel controllable texture synthesis // ACM Transactions on Graphics. 2005. V. 24. № 3. P. 777–786.
20. *Han C. et al.* Multiscale texture synthesis // ACM Transactions on Graphics. 2008. V. 27. № 3. P. 51.
21. *Dong Y., Lefebvre S., Tong X., Drettakis G.* Lazy Solid Texture Synthesis // Computer Graphics Forum. 2008. V. 27. P. 1165–1174.
22. *Tanner C.C., Migdal C.J., Jones M.T.* The clipmap: a virtual mipmap // Association of Computing Machinery's Special Interest Group on Computer Graphics and Interactive Techniques. 1998. P. 151–158.
23. *Lefebvre S., Darbon J., Neyret F.* Unified texture management for arbitrary meshes // Technical Report RR5210, INRIA, May 2004.
24. *Mittring M. et al.* Advanced virtual texture topics // Association of Computing Machinery's Special Interest Group on Computer Graphics and Interactive Techniques. 2008. P. 23–51.
25. *Barrett S.* Sparse virtual textures. http://silverspaceship.com/src/svt/
26. *Bilodeau B., Sellers G., Illesland K.* Partially Resident Textures on Next-Generation GPUs // Game Developers Conference, 2012.
27. *Peecmp Khronos Group*. https://www.khronos.org/registry
28. *Mayer A.J.* Virtual texturing // Master's thesis, Institute of Computer Graphics and Algorithms. Vienna University of Technology, October, 2010.
29. *Yan L.Q., Hasan M., Jakob W., Lawrence J., Marschner S., Ramamoorthi R.* Rendering glints on high-resolution normal-mapped specular surfaces // ACM Transactions on Graphics. 2014. V. 33. № 4. P. 116.
30. *Igehy H.* Tracing Ray Differentials // Proceedings of ACM SIGGRAPH. 1999. P. 179–186.
31. *Pharr M., Jakob W., Humphreys G.* Physically based rendering: From theory to implementation. Morgan Kaufmann, 2016.
32. Рендер-система Hydra. http://www.raytracing.ru/
33. *Efros A. A., Freeman W. T.* Image quilting for texture synthesis and transfer // Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM. 2001. С. 341–346.
34. Реализация алгоритма ImageQuilting. https://github.com/PJunhyuk/ImageQuilting
35. Реализация алгоритма DeepTextures. https:// github.com/leongatys/DeepTextures