ПРОГРАММИРОВАНИЕ, 2023, № 3, с. 3–12

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.921

ПОДДЕРЖКА ВЕКТОРНЫХ ТЕКСТУР В СИСТЕМЕ ФОТОРЕАЛИСТИЧНОГО РЕНДЕРИНГА НА GPU

© 2023 г. В. В. Санжаров[a,*] (ORCID: 0000-0001-6455-6444),
В. А. Фролов[a,b,] (ORCID: 0000-0001-8829-9884), В. А. Галактионов[b,***] (ORCID: 0000-0001-6460-7539)**

[a] *Московский государственный университет им. М.В. Ломоносова*
Россия, 119991, Москва, Ленинский горы, д. 1

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

**e-mail: vadim.sanzharov@graphics.cs.msu.ru*

***e-mail: vladimir.frolov@graphics.cs.msu.ru*

****e-mail: vlgal@gin.keldysh.ru*

Поступила в редакцию 09.01.2023 г.
После доработки 16.01.2023 г.
Принята к публикации 20.01.2023 г.

Изображения в векторном формате представлены в виде последовательности аналитических описаний геометрических объектов. Такой подход позволяет воспроизвести изображение в любом разрешении без потери качества. На текущий момент не существует готовых решений для использования векторных изображений в системах фотореалистичного рендеринга на GPU. В данной работе представлен подход к реализации такой поддержки, основанный на базовых методах – полей расстояний со знаком и растеризации. Анализ результатов показывает эффективность подхода на основе полей расстояний для различных векторных изображений. Однако, в отдельных случаях возможно появление артефактов, в этом случае предлагается использовать подход на основе растеризации.

DOI: 10.31857/S0132347423030044, **EDN:** DELQSS

1. ВВЕДЕНИЕ

Векторные изображения представляют изображения на основе математического аппарата аналитической геометрии. Вводятся описания геометрических объектов в декартовой системе координат. Среди поддерживаемых объектов – точки, прямые, окружности, полигоны, кривые Безье и пр. Один из наиболее распространенных форматов векторных изображений – формат SVG, являющийся стандартом Консорциума Всемирной паутины (World Wide Web Consortium, W3C) [1]. В данном формате векторные изображения представлены текстовым описанием в виде XML-файла. Геометрические объекты в таком описании объединяются в группы, которым может назначаться набор свойств. Для получения растрового изображения, необходимо последовательно “нарисовать” геометрические объекты в соответствии с их описанием и свойствами.

Такой подход к представлению графической информации обладает рядом преимуществ. Во-первых, он позволяет получить изображение в нужном разрешении без потерь информации. Во-вторых, размер векторного изображения зависит от его сложности, а не от разрешения или глубины цвета, что позволяет получить компактное представление, по сравнению с растровыми изображениями (особенно в случае высокого разрешения последних). Наконец, описание объектов на основе координат и уравнений позволяет получить большую точность описания изображения, чем использование растровой сетки с конечным разрешением. Также имеется ряд преимуществ с точки зрения создания векторных изображений. В частности, возможность манипулировать отдельными объектами независимо, а также – выполнять алгоритмическую генерацию текстового описания изображения, включая генерацию сложных паттернов [2].

Эти преимущества привели к распространению векторной графики в различных приложениях, где информация может быть представлена с помощью набора геометрических примитивов – типография, пользовательские интерфейсы, веб-дизайн, автоматизированное проектирование (чертежи) и др.

В 3D-графике использование векторных изображений ограничивается интерактивными графическими приложениями, где таким способом представляют преимущественно пользователь-

Рис. 1. Пример изображений, синтезированных системой фотореалистичного рендеринга с использованием предлагаемого решения. **Слева** – рендер сцены производился в разрешении FullHD, векторные изображения на картине и на полу. **По центру** – на этой же сцене камера была приближена к текстурированной картине и рендер был произведен в разрешении 8192 на 8192 без изменения векторной текстуры. **Справа** – векторные текстуры использовались как маски для смешения разных моделей материалов.

ские интерфейсы [3] и текст [4]. Это во многом связано с тем, что с помощью простых геометрических форм, используемых в векторной графике, сложно описывать объекты реального мира. Это также касается (даже в большей мере) описания цвета. Векторные изображения обычно используют закраску сплошным цветом или градиенты. Подобными средствами чрезвычайно сложно представить такие эффекты, как микрорельеф поверхности, мягкие тени, каустики, глубину резкости и др. [5]. Проблема сложности создания векторных изображений объектов реального мира начинает находить решение в использовании подходов, позволяющих автоматически синтезировать векторные изображения из растровых изображений (в том числе фотографий), на основе дифференцируемого рендеринга [6], нейронных сетей [7], включая диффузионные модели [8].

Другая проблема использования векторных изображений связана с тем, что для вычисления цвета пикселя необходимо обойти все геометрические примитивы в изображении. Тогда как для растрового изображения возможен эффективный случайный доступ к любому пикселю, имеющий также аппаратную поддержку в графических процессорах.

Несмотря на описанные недостатки, использование векторных текстур в фотореалистичном рендеринге на текущий момент может быть оправдано при необходимости синтезировать в высоком разрешении изображения, включающие текст, изображения, которые могут быть представлены геометрическими примитивами (рис. 1), а с учетом развития методов автоматической генерации векторных изображений, возможно, и произвольные текстуры. Так как при рендеринге в высоких разрешениях для обеспечения качественного результата необходимо использовать текстуры также высокого разрешения, в то время как векторные изображения не зависят от разрешения.

В данной работе представлен подход к поддержке векторных текстур в формате SVG в фотореалистичной системе рендеринга на GPU.

2. СУЩЕСТВУЮЩИЕ РЕШЕНИЯ

В большинстве систем фотореалистичного рендеринга для использования изображения в векторном формате в качестве текстуры пользователь сначала должен самостоятельно выполнить его конвертацию в растровое изображение в некотором стороннем инструменте. При этом ему понадобится некоторым образом определить необходимое разрешение, например, исходя из желаемого разрешения рендеринга. А при изменении разрешения рендеринга в большую сторону, скорее всего потребуется выполнить конвертацию заново, также увеличив разрешение растрового представления векторного изображения. Такой подход создает неудобства для конечного пользователя, вынуждая его найти и использовать некоторый сторонний инструмент для конвертации, а также задавать параметры этой процедуры.

Также следует отметить, что многие области применения фотореалистичного рендеринга подразумевают синтез изображения в высоком разрешении. В частности, полиграфия, визуальные эффекты и анимационные фильмы, моделирование оптических систем. Таким образом, векторные изображения требуется конвертировать в растровые изображения высокого разрешения, занимающие значительные объемы памяти. А с учетом перехода многих систем фотореалистичного рендеринга на использование GPU, связанное с появлением аппаратного ускорения трассировки лучей [9], этот фактор приобретает еще большую значимость.

2.1. Обзор существующих решений

Существуют различные программные решения для работы с векторными изображениями, такие как Skia [10], cairo [11], Blend2d [12], resvg [13]. Данные решения обеспечивают полноценную поддержку стандарта SVG, высокую производительность, различные целевые системы. В случае реализации рендер-системы на CPU, было бы возможно подключить одно из этих решений в качестве библиотеки и вызывать ее при необходимости обратиться к векторной текстуре. Однако, для GPU рендер-системы такая возможность отсутствует, так как вычислительное ядро такого рендера работает в другом адресном пространстве.

Консорциумом Khronos Group был разработан стандарт программного интерфейса (API) для аппаратно-ускоренного рендеринга векторной графики под названием OpenVG [14], нацеленный на приложения 2D-графики – навигаторы, чтение электронных книг и др. Данный стандарт предлагает новый вид графического конвейера для векторной графики. Поддержка OpenVG была реализована главным образом в мобильных GPU [15]. Однако, широкого распространения данный стандарт не получил.

Расширение NV_path_rendering [16] для OpenGL позволяет использовать примитивы типа “путь” (path, по сути параметрическая кривая, сплайн) векторной графики в 3D графических приложениях на GPU от NVidia. Отсутствие поддержки аппаратной трассировки лучей в OpenGL, а также отсутствие кроссплатформенности делают это решение неактуальным для современных систем фотореалистичного рендеринга.

В работе [17] предлагается метод рендеринга параметрических кривых с использованием аппаратного ускорения на GPU. Предлагаемый метод основан на представлении сегментов кривой с помощью треугольников, которые затем отрисовываются с помощью графического конвейера. Для систем фотореалистичного рендеринга, подавляющее большинство которых основано на трассировки лучей, таким образом потребуется выполнять построение ускоряющих структур данных для сгенерированной геометрии. А при изменении и/или добавлении текстур в сцену, потребуется обновлять ускоряющие структуры. При этом количество сгенерированных треугольников может быть достаточно велико, а для мелких элементов векторного изображения метод дает артефакты.

В работе [18] авторы предлагают специальную ускоряющую структуру данных для реализации эффективного рендеринга векторных изображений на GPU. Данный подход ориентирован на приложения 2D-графики.

Работа [19] предлагает новое представление для векторных изображений, ориентированное на эффективный рендеринг с помощью графического конвейера. Основным недостатком подхода является отсутствие автоматического решения по преобразованию произвольных векторных изображений в новое представление, предложенное авторами. Также, использование функций, специфичных для графического конвейера, ограничивает применение подхода рендер-системами, уже использующими графический конвейер.

В работе [20] впервые предложен подход на основе полей расстояний со знаком (signed distance field, SDF). Суть подхода состоит в том, что для параметрических кривых вычисляются поля расстояний, которые затем используются в качестве текстур. Рендер-система обращается к текстуре поля расстояний и на основе полученного значения, определяет, виден ли в данном пикселе геометрический объект, которому соответствует данная текстура. Авторы [4] предлагают комбинировать поля расстояний со знаком с механизмом альфа-смешивания для вычисления цвета при использовании векторных изображений как текстур. Работа [21] совершенствует подход на основе полей расстояний за счет расчета нескольких полей расстояний для разных граней геометрической формы, которые хранятся в разных цветовых каналах SDF-текстуры. Это позволяет повысить точность представления исходной геометрической формы. Недостатком подходов на основе SDF являются визуальные артефакты для некоторых видов геометрических объектов, в частности, обладающих острыми углами. Кроме того, для точного представления сложных форм требуется генерация SDF-текстур в достаточно высоком разрешении. Наконец, эти методы не могут представить несколько кривых, пересекающихся в одном пикселе и сами по себе рассматривают случай только отдельных непересекающихся геометрических объектов. На практике же векторное изображение может состоять из большого числа наложенных друг на друга и пересекающихся объектов.

В работе [22] также используется представление на основе расстояний, однако уже рассматривается целое векторное изображение, представленное в виде слоев, содержащих геометрические объекты. Векторное изображение представляется в виде сетки, в каждой ячейке которой хранится информация о всех геометрических объектах, попадающих в данную ячейку. Для корректного рендеринга наложения объектов используется композирование. Недостатком подхода является использование специфичных для графического конвейера функций вычисления частных производных по изображению (*dFdx* и *dFdy* в GLSL) для отображения из текстурного пространства в пространство экрана. Реализация данных операций в фотореалистичной рендер-системе на основе трассировки лучей возможна с использованием т.н. дифференциалов луча (ray differentials) [23].

Однако метод дифференциалов луча дает строго корректные результаты только для прямых лучей и может давать значительную ошибку для многократных отражений. Насколько данная ошибка будет влиять на подход к реализации векторных текстур в [22] является открытым вопросом. Отчасти похожий подход, также основанный на расчете расстояний, предлагается в [24]. Информация о векторном изображении здесь сохраняется в специальную ускоряющую структуру данных, представленную в виде трех текстур. Для данного подхода, а также для [22] остаются недостатки, присущие другим подходам на основе расстояний, связанные с возможными артефактами для некоторых видов геометрических объектов. Также в [22] и [24] вычисление расстояний происходит в процессе рендеринга, что ограничивает использование возможных алгоритмов для этой задачи и может ощутимо влиять на производительность.

2.2. Резюме по существующим решениям

Все существующие решения имеют недостатки, так или иначе ограничивающие их применение. Готовые решения для 2D-рендеринга [10–13] могут быть использованы либо в CPU рендер-системе как библиотеки для вычисления векторной текстуры на лету, или же могут быть интегрированы в качестве средства предварительной растеризации векторной текстуры. Часть решений [16] накладывают ограничения по использованию определенного аппаратного и программного обеспечения. В [19, 22] используется функциональность, специфичная для графического конвейера. Подходы, основанные на вычислении расстояний [4, 20–22, 24] могут давать артефакты для некоторых видов геометрических объектов. При этом часть подходов обрабатывают отдельные геометрические объекты [4, 20, 21], а часть [22, 24] выполняют расчет расстояний в процессе рендеринга, оказывая влияние на производительность.

Таким образом, для использования векторных текстур в системе фотореалистичного рендеринга на GPU требуется объединить несколько существующих подходов, чтобы покрыть основные сценарии использования.

3. ПРЕДЛАГАЕМОЕ РЕШЕНИЕ

В качестве формата векторных изображений предлагаемое решение поддерживает формат SVG [1], как стандартизированный и один из наиболее распространенных. Формат SVG является достаточно сложным и помимо различных геометрических примитивов может включать встроенные растровые изображения, встроенный код на языке JavaScript, анимацию и многие другие компоненты. При этом эти компоненты могут обладать различными свойствами, включая разные виды цветовой заливки, контурную обводку разными типами линий, фильтры и др. Предлагаемое решение реализует поддержку ограниченного подмножества функциональности допустимой в формате SVG. В частности, в качестве геометрических объектов поддерживаются объекты типа path (параметрические кривые), так как все остальные геометрические объекты, включая текст, могут быть представлены с помощью объектов типа path.

В первую очередь предлагаемое решение загружает файл в формате SVG и вычленяет все объекты типа path в порядке их расположения по слоям, а также свойства этих объектов – цвет, геометрические преобразования, наличие контуров и их цвет.

Далее каждый объект path подается на вход компоненту вычисления поля расстояний. После расчета поля расстояний для объекта опционально выполняется обрезание поля расстояний по ограничивающему прямоугольнику соответствующего геометрического объекта. Это позволяет избавиться от артефактов, вносимых неточностями расчета поля расстояний при удалении от объекта (рис. 5). На выходе имеем массив отдельных полей расстояний для каждого из объектов path в исходном векторном изображении.

В качестве следующего этапа может опционально производиться объединение отдельных полей расстояний в единое путем наложения, если геометрические объекты в исходном изображении не пересекаются. Такой случай характерен в первую очередь для текста, т.к. обычно в компьютерных шрифтах буквы не пересекаются и не накладываются одна на другую.

Наконец массив полей расстояний (или комбинированное поле расстояний) вместе с массивами, содержащими свойства геометрических объектов, передается на GPU. Эти данные используются в процессе рендеринга при обращении к текстуре, соответствующей векторному изображению. При таком обращении вызывается текстурный шейдер, сгенерированный на основе настроек, заданных пользователем при загрузке векторного изображения. В текстурном шейдере (по сути в процедурной текстуре), являющейся программируемой частью рендер системы (см. рис. 2), выполняется расчет цвета пикселя на основе полей расстояний и композирования объектов path. В простом случае одного поля расстояний происходит альфа-смешивание цвета объекта path и некоторого фонового цвета (например, цвета диффузного материала, для которого используется векторная текстура), где в качестве веса для смешивания используется расстояние из поля расстояний:

$$dist = median(msdf)$$

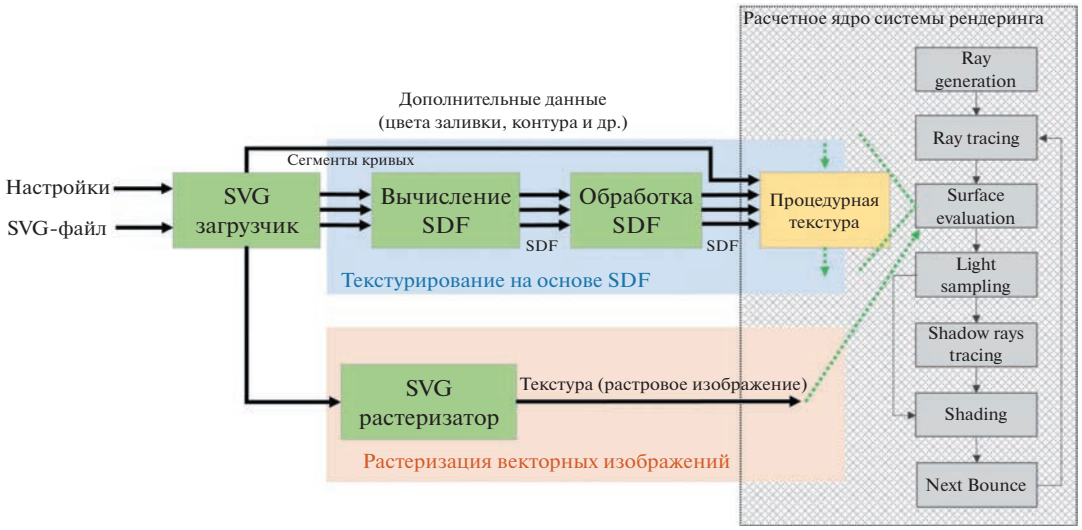

Рис. 2. Схема предлагаемого решения. Блок “Обработка SDF” – опциональный, включает в себя вырезание SDF по ограничивающему прямоугольнику геометрической формы, для которой рассчитывалось поле расстояний.

$$\alpha = smoothstep(T - S, T + S, dist) \quad (1)$$

$$color = bgColor * (1 - \alpha) + baseColor * \alpha$$

где:

• *median(a)* – функция, принимающая на вход вектор и возвращающая медиану его компонентов;

• *msdf* – вектор из трех компонент, содержащий значения полей расстояний;

• *smoothstep(a, b, x)* – функция выполняющая эрмитову интерполяцию между 0 и 1, при $a < x < b$;

• T – константа, которая задает значение расстояния, определяющее границы геометрического объекта;

• S – константа сглаживания, при больших значениях цветовой переход на границе объекта будет более плавным;

• *bgColor* – цвет фона;

• *baseColor* – цвет геометрического объекта.

Функция *smoothstep* служит в качестве механизма антиалиасинга для “смягчения” перехода на границе геометрического объекта, вместо попиксельного вычисления частных производных по координатам экрана с помощью встроенной функциональности графического конвейера. При необходимости она может быть заменена на другую схожую по характеристикам функцию или же на использование производных, при их доступности.

В более общем случае, при наличии нескольких полей расстояний, необходимо следовать формату SVG, в котором геометрические объекты задаются в порядке отрисовки – второй объект рисуется поверх первого, третий поверх второго и т.д. Таким образом, необходимо по порядку обойти массив полей расстояний, на каждом этапе выполняя альфа-смешивание очередного геометрического объекта с результатом предыдущей итерации аналогично (1):

$$\begin{aligned} &color = bgColor \\ &for\ each\ object: \\ &\quad baseColor = object.color \\ &\quad msdf = object.msdf \\ &\quad dist = median(msdf) \\ &\quad \alpha = smoothstep(T - S, T + S, dist) \\ &\quad color = color * (1 - \alpha) + baseColor * \alpha \\ &return\ color \end{aligned} \quad (2)$$

где:

• *object.color* – цвет текущего геометрического объекта;

• *object.msdf* – поле расстояний текущего геометрического объекта.

Для того чтобы реализовать поддержку контуров, диапазон между минимальным и максимальным значениям расстояния задается равным ширине контура. Тем самым минимальное расстояние будет соответствовать объекту, максимальное – фону, а между ними будет располагаться контур. Тогда, на этапе смешивания, если значения расстояния попадают в этот диапазон, то значение *baseColor* в (1) и (2) рассчитывается как результат смешивания цветов геометрического объекта и контура:

$$\omega = \begin{cases} smoothstep(0, S, dist), dist < S \\ smoothstep(1, 1 - S, dist), dist \geq S \end{cases} \quad (3)$$

$$baseColor = shapeColor * (1 - \omega) + outlineColor * \omega$$

где:

• *shapeColor* – цвет заливки геометрического объекта;

• *outlineColor* – цвет контура.

В предлагаемой реализации поддерживается только сплошная заливка объекта и контура. Однако, поддержка, например, градиентной заливки представляет собой довольно простую модификацию. Для этого необходимо сохранять не просто значения цвета заливки *shapeColor*, а значения нескольких цветов и коэффициенты смешивания. Например, вычислить медиану между полями расстояний заранее, а в другие цветовые каналы записать коэффициенты градиентов.

3.2. Детали реализации

Предлагаемая реализация была интегрирована в рендер-систему, которая взаимодействует с клиентскими приложениями через промежуточный программный слой (API). Для загрузки векторного изображения пользователю предоставляется возможность задания настроек реализованного подхода, которые включают:

• способ вычисления полей расстояний или же использование механизма растеризации;

• множитель разрешения поля расстояний или растеризованной текстуры;

• параметры алгоритма расчета расстояний;

• флаг генерации одноканальной маски из векторного изображения;

• указание комбинировать все поля расстояния вместе и указание выполнять обрезание поля расстояний по ограничивающему прямоугольнику геометрического объекта;

• текстурную матрицу и способ обработки текстурных координат, превышающих 1;

• включение/выключение отрисовки контуров;

• цвет фона и переназначение цвета для всех геометрических объектов и всех контуров.

Большинство из этих параметров нет необходимости менять при стандартных сценариях использования. Однако, при необходимости возможна тонкая настройка алгоритма. Например, в специальных случаях, таких как текст или паттерн из непересекающихся объектов, можно получить прирост в производительности и экономию памяти путем комбинирования полей расстояния. Если векторное изображение используется только как маска, то достаточно вычислить одноканальное изображение. А при появлении артефактов, изменение параметров расчета полей расстояния может помочь от них избавиться или уменьшить их проявление. На основе заданных параметров производится генерация кода процедурной текстуры. Пример фрагмента сгенерированного кода представлен ниже:

```
float4 prtex14_main(const SurfaceInfo * sHit, unsigned mode, ...){
    const float S = 0.015625; const float T = 0.5;
    const float2 texCoord = readAttr_TexCoord0(sHit);
    float2 texCoord_adj = prtex14_applyTexMatrix(texCoord, texMatrix);
    float4 resColor = bgColor;
    for(int i = 0; i < numTextures; i += 1) {
    const float4 objColor = prtex14_colorFromUint(objColors[i]);
    const float4 texColor = texture2D(sdfTexture[i], texCoord_adj, texFlags);
    const float distance = (mode & MSDF) ?
    prtex14_median(texColor.x, texColor.y, texColor.z) : texColor.x;
    const float alpha = prtex14_smoothstep(T - S, T + S, distance);
    const float4 outlineColor = prtex14_colorFromUint(outlineColors[i]);
    float4 baseColor = (hasOutline[i] == 0) > objColor:
    prtex14_addOutline(distance, S, outlineColor, objColor);
    resColor = prtex14_mix4(resColor, baseColor, alpha);
    }
    return resColor;
}
```

В частности, если отключена отрисовка контуров, часть алгоритма, ответственная за нее выбрасывается из результирующего кода. Если выполняется композирование полей расстояний или в исходном изображении только один объект, то нет необходимости передавать дополнительные

данные для смешивания. Таким образом, код процедурной текстуры генерируется в максимально упрощенном виде.

3.3. Экспериментальная оценка

Как было отмечено ранее, предлагаемый подход в отличие от [22, 24] выносит вычисление полей расстояний в некоторый предварительный этап, внешний по отношению к процессу рендеринга. Но при этом возникает необходимость загрузки полей расстояний как текстур на GPU и обращению к ним в процессе рендеринга. Если для каждого из объектов в исходном векторном изображении сгенерировано отдельное поле расстояний, то это так же может оказывать ощутимое влияние на производительность. Для оценки этого был проведен эксперимент:

1. Алгоритмически были сгенерированы векторные изображения, содержащие заданное в диапазоне от 16 до 2048 количество геометрических объектов (случайных секторов окружности), расположенных случайным образом.

2. Для каждого из сгенерированных изображений рассчитано поле расстояний с одинаковыми настройками алгоритма.

3. Произведен рендеринг простейшей 3D-сцены с равномерным фоновым освещением, содержащей плоскость со сгенерированным векторным изображением в качестве текстуры.

4. Проведены замеры производительности.

Результаты эксперимента приведены на рис. 3. Видно, что зависимость от количества объектов в векторном изображении (т.е. от числа текстур полей расстояний) имеет линейный характер. На тестовой сцене, где текстурированная плоскость занимает все видимое изображение, при количестве объектов меньше 256, затраты на исполнение предложенного алгоритма составляют меньше 1 секунды для 256 выборок (путей) на пиксель при разрешении рендеринга 1024 на 1024. А для 2048 объектов в векторном изображении время исполнения процедурной текстуры равно примерно 8.2 секунды. В контексте фотореалистичного рендеринга это приемлемые затраты. При этом видно, что процент времени рендеринга, необходимый для исполнения процедурной текстуры “насыщается” и замедляет свой рост с увеличением числа объектов.

Для следующего эксперимента были выбраны векторные изображения из открытых источников [25, 26], представляющие некоторые типичные сценарии использования (узоры, стилизованные изображения, текст) с разной сложностью – разным количеством объектов. Выбранные изображения использовались в качестве текстуры в такой же сцене с плоскостью, как и в предыдущем эксперименте. Были произведены аналогичные замеры производительности (табл. 1), а также оценивалось качество результатов (рис. 4, 5).

Для некоторых изображений были зафиксированы артефакты, возникающие для определенных элементов (рис. 5).

Часть этих артефактов обусловлена проблемами, возникающими у метода расчета полей расстояний для острых углов. Это становится особенно заметным при отключении вырезания поля расстояний по ограничивающему прямоугольнику объекта (рис. 5D, E). Обрезание поля расстояний в значительной мере устраняет артефакты (рис. 5A). Если при вырезании поля расстояний использовать выпуклую оболочку, а не ограничивающий прямоугольник, то потенциально может быть устранено еще больше артефактов данного вида.

Другая группа артефактов возникает из-за некорректной интерпретации определенных форм. Например, у незамкнутых сегментов параметрической кривой, которые не имеют внутренней заливки, а отображаются только как контуры (рис. 5B, C). Решение данной проблемы требует внесения модификаций в алгоритм расчета расстояний для специальной обработки таких случаев. В предлагаемой реализации на текущий момент при возникновении таких артефактов предлагается использовать вспомогательный подход на основе растеризации.

Однако при использовании подхода на основе растеризации потребуется использовать растровое изображение с потенциально значительно большим разрешением, чем разрешение текстур полей расстояний. Для адекватного представления исходного векторного изображения необходимо обеспечить достаточно разрешения и текстуры поля расстояний. Однако, как только разрешения будет достаточно, чтобы передать все детали векторного изображения, дальнейшее увеличение разрешения не будет давать никаких результатов. Таким образом, можно увеличивать разрешение рендеринга и приближать текстурированный объект как угодно без потерь качества. А для растеризованной текстуры, при увеличении разрешения рендеринга потребуется соответствующим образом увеличить и разрешение текстуры, чтобы избежать появления артефактов. Это становится особенно важным для случая явных, резких границ между объектами, характерных для векторных изображений. Поэтому при больших разрешениях рендеринга, выигрыш от использования подхода на основе полей расстояний будет расти.

Наконец, следует отметить, что в предлагаемой архитектуре решения (рис. 2), включающей модуль растеризации, можно рассматривать векторные текстуры как случай “предварительно вычисленных процедурных текстур” и использовать

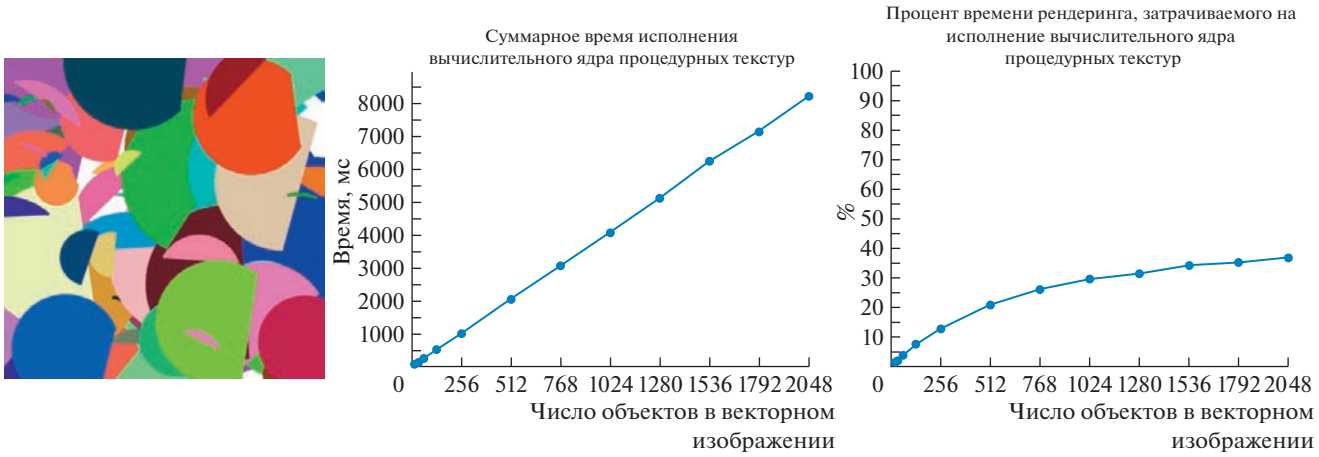

Рис. 3. Результаты тестирования производительности на синтезированных текстурах. Слева – пример синтезированной текстуры. По центру – зависимость суммарного времени исполнения вычислительного ядра процедурных текстур от числа объектов в векторном изображении. Справа – зависимость процента времени рендеринга, затрачиваемого на исполнение вычислительного ядра процедурных текстур. Тестирование производилось на GPU RTX2070 Super, рендеринг в разрешении 1024 на 1024, 256 выборок на пиксель, разрешение SDF текстур 256 на 256.

описанный в [27] механизм для автоматического определения разрешения растеризации, необходимого для достижения соотношения текселей к пикселям 1 : 1.

4. ЗАКЛЮЧЕНИЕ

Предложено решение, обеспечивающее возможность использования векторных изображений, содержащих объекты, представленные параметрическими кривыми, в качестве текстур в фотореалистичном рендеринге 3D-сцен. Решение представляет собой программную архитектуру (рис. 2), интегрирующую метод на основе расчета полей расстояний (в рассмотренной реализации использовался метод [21]) и вспомогательный механизм растеризации векторных изображений. Для подхода на основе полей расстояний предложен подход обработки случая пересекающихся объектов векторных изображений и объектов, имеющих контуры. Предложенная программная архитектура является гибкой и позволяет заменять модули расчета полей расстояний и растеризации, независимо от рендер-системы. Также предложенная архитектура снимает с пользователя рендер-системы задачу по ручной конвертации векторных изображений в стороннем инструменте и предоставляет возможность использовать их напрямую в рендер-системе. Реализованный подход на основе полей расстояний не оказывает значи-

Таблица 1. Замеры производительности предлагаемого решения для векторных изображений из открытых источников (см. рис. 4). Замеры производилось на GPU RTX2070 Super, рендеринг в разрешении 1024 на 1024, 256 выборок на пиксель, разрешение SDF-текстур 256 на 256

Изображение	Число объектов в изображении	Время рендеринга, мс	Суммарное время исполнения выч. ядра проц. текстур, мс	Процент от времени рендеринга, затрачиваемый на исполнение выч. ядра проц. текстур, %	Разрешение текстур полей расстояний
tiger	239	7751	1056	13.6	512 × 512
autumn leaves	21	5750	123	2.1	512 × 512
trophy	15	5750	83	1.4	512 × 512
Horus and Thoth	11	6251	127	2.0	1024 × 1024
prismatic mandala	223	8251	1041	12.6	1024 × 1024
text	105	6751	421	6.2	512 × 512
text combined	1	5751	58	1.0	512 × 512

Рис. 4. Тестирование предложенного подхода на векторных изображениях из открытых источников. В верхнем ряду – результат рендеринга с использованием предложенного подхода, внизу – эталонные изображения. На изображении листьев демонстрируется применение текстурных матриц с бесшовной векторной текстурой. Присутствуют артефакты – например, на изображении тигра (см. рис. 5).

Рис. 5. Примеры артефактов на изображении тигра. A – результат рендеринга с использованием предлагаемого метода с вырезанием поля расстояний по ограничивающему прямоугольнику объекта; B – наверху фрагмент эталонного изображения, по середине – фрагмент синтезированного изображения A, внизу – элементы векторного изображения, вызывающие проблему; C – текстуры поля расстояний для проблемных элементов; D – результат рендеринга без вырезания поля расстояний; E – текстуры поля расстояний с артефактными значениями для острых углов.

тельного влияния на время рендеринга. Для случая векторных изображений, содержащих текст и простые паттерны, их использование равноценно использованию одной обычной растровой текстуры (табл. 1).

СПИСОК ЛИТЕРАТУРЫ

1. Scalable Vector Graphics (SVG) Full 1.2 Specification, 2023. URL: https://www.w3.org/TR/SVG12
2. *Tu P., Wei L. Y., Zwicker M.* Clustered vector textures // ACM Transactions on Graphics (TOG). 2022. Т. 41. № 4. С. 1–23. https://doi.org/10.1145/3528223.3530062
3. Noesis Gui. User Interface middleware for videogames and real-time applications, 2023. URL: https://www.noesisengine.com
4. *Green C.* Improved alpha-tested magnification for vector textures and special effects // ACM SIGGRAPH 2007 Courses. 2007. P. 9–18.
5. *Orzan A. et al.* Diffusion curves: a vector representation for smooth-shaded images // ACM Transactions on Graphics (TOG). 2008. Т. 27. № 3. С. 1–8. https://doi.org/10.1145/1360612.1360691
6. *Li T. M. et al.* Differentiable vector graphics rasterization for editing and learning // ACM Transactions on Graphics (TOG). 2020. Т. 39. № 6. С. 1–15. https://doi.org/10.1145/3414685.3417871
7. *Reddy P. et al.* Im2vec: Synthesizing vector graphics without vector supervision // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021. С. 7342–7351.
8. *Jain A., Xie A., Abbeel P.* VectorFusion: Text-to-SVG by Abstracting Pixel-Based Diffusion Models //arXiv preprint arXiv:2211.11319. 2022.
9. *Sanzharov V.V., Frolov V.A., Galaktionov V.A.* Survey of Nvidia RTX technology // Programming and Computer Software. 2020. Т. 46. № 4. С. 297–304. https://doi.org/10.1134/S0361768820030068
10. Skia: The 2D graphics library, 2023. URL: https://skia.org/

11. Cairo, a 2D graphics library with support for multiple output devices, 2023. URL: https://www.cairographics.org/
12. Blend2d. 2D Vector graphics engine, 2023. URL: https://blend2d.com/
13. resvg, SVG rendering library, 2023. URL: https://github.com/RazrFalcon/resvg
14. OpenVG, the standard for vector graphics acceleration, 2023. URL: https://www.khronos.org/openvg/
15. OpenVG, conformant Products, 2023. URL: https://www.khronos.org/conformance/adopters/conformant-products/openvg
16. *Kilgard M.J., Bolz J.* GPU-accelerated path rendering // ACM Transactions on Graphics (TOG). 2012. Т. 31. № 6. С. 1–10. https://doi.org/10.1145/2366145.2366191
17. *Loop C., Blinn J.* Resolution independent curve rendering using programmable graphics hardware // ACM SIGGRAPH 2005 Papers. 2005. С. 1000–1009. https://doi.org/10.1145/1186822.1073303
18. *Ganacim F. et al.* Massively-parallel vector graphics // ACM Transactions on Graphics (TOG). 2014. Т. 33. № 6. С. 1–14. https://doi.org/10.1145/2661229.2661274
19. *Ray N., Cavin X., Lévy B.* Vector texture maps on the GPU //Inst. ALICE (Algorithms, Comput., Geometry Image Dept. INRIA Nancy Grand-Est/Loria), Tech. Rep. ALICE-TR-05-003. 2005.
20. *Qin Z., McCool M.D., Kaplan C.S.* Real-time texture-mapped vector glyphs // Proceedings of the 2006 symposium on Interactive 3D graphics and games. 2006. С. 125–132. https://doi.org/10.1145/1111411.1111433
21. *Chlumsky V.* Shape decomposition for multi-channel distance fields: Master's thesis, Czech Technical University. URL: https://dspace.cvut.cz/bitstream/handle/10467/62770/F8-DP-2015-Chlumsky-Viktor-thesis. pdf. 32, 2015.
22. *Nehab D., Hoppe H.* Random-access rendering of general vector graphics // ACM Transactions on Graphics (TOG). 2008. Т. 27. № 5. С. 1–10. https://doi.org/10.1145/1409060.1409088
23. *Akenine-Möller T. et al.* Texture level of detail strategies for real-time ray tracing // Ray Tracing Gems. Apress, Berkeley, CA, 2019. С. 321–345. https://doi.org/10.1007/978-1-4842-4427-2_20
24. *Qin Z., McCool M D., Kaplan C.* Precise vector textures for real-time 3D rendering // Proceedings of the 2008 symposium on Interactive 3D graphics and games. 2008. С. 199–206. https://doi.org/10.1145/1342250.1342281
25. Vector images in public domain, 2023. URL: https://www.publicdomainvectors.org
26. Open Clipart, online media collection, 2023. URL: https://openclipart.org/
27. *Sanzharov V.V., Frolov V.A.* Level of detail for precomputed procedural textures // Programming and Computer Software. 2019. Т. 45. № 4. С. 187–195. https://doi.org/10.1134/S0361768819040078