Субпиксельные изображения на основе BSP дерева

В. А. Фролов[1,2], А. С. Щербаков[2]

[1]*МГУ им. М.В.Ломонсова, ГСП-1, Ленинские горы, д. 1, стр. 52, факультет ВМК, Москва, 119234, Россия*
[2]*ИПМ им. М.В.Келдыша РАН, Миусская пл., д.4, Москва, 125047, Россия*

Аннотация

Мы предлагаем представление изображений, запоминающиее суб-пиксельные границы между объектами. Это даёт ряд преимуществ в 4 сценариях: (1) устранение ступенчастоти, (2) повышение разрешения, (3) шумоподавление, (4) дифференцируемый рендеринг. Для пикселей, содержащих границы мы сохраняем информацию об этих границах при помощи BSP дерева с небольшой глубиной. Это позволяет нам восстанавливать геометрическую форму объекта внутри пиксела с высокой точностью если есть необходимость, например, в существенном увеличении разрешения изображения (8-16 раз). Мы демонстриурем, что предложенный метод позволяет сохранить информацию о границах с высокой точность и сделать качественное увеличение разрешения с незначительным увеличением потребления памяти относительно исходного изображения в среднем от (+1%) на типичных сценариях. Область применения предложенного метода – это синтез изображений в компьютерной графике.

Ключевые слова

Устранение ступенчатости, повышение разрешения, дифференцируемый рендеринг

Sub-pixel BSP images

Vladimir Frolov[1,2], Alexander Scherbakov[2]

[1]*Moscow State University, GSP-1, Leninskie gory, 1, 52, CMC, Moscow, 119234, Russia*
[2]*Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, Moscow, 125047, Russia*

Abstract

We propose new image representations that saves sub-pixel borders between objects. This provides a number of benefits in 4 scenarios: (1) anti-aliasing, (2) up-sampling, (3) de-noising, (4) differentiable rendering. For pixels containing borders, we store information about these borders using a small BSP tree. This allows us to restore the geometric shape of an object inside a pixel with high accuracy if there is a need, for example, to significantly increase the image resolution (8-16 times). We demonstrate that the proposed method allows to save edge information with high accuracy and to make a qualitative increase in resolution with a slight increase in memory consumption relative to the original image by an average of (+1%) in typical scenarios. The scope of the proposed method is the synthesis of images in computer graphics.

Keywords

anti-aliasing, up-sampling, differentiable rendering

GraphiCon 2022: 32nd International Conference on Computer Graphics and Vision, September 19–22, 2022, Ryazan State Radio Engineering University named after V.F. Utkin, Ryazan, Russia
vfrolov@graphics.cs.msu.ru (В. А. Фролов); alex.shcherbakov@graphics.cs.msu.ru (А. С. Щербаков)
0000-0001-8829-9884 (В. А. Фролов); 0000-0002-5360-4479 (А. С. Щербаков)

1. Введение

Идея сохранения агреггированной субпиксельной информации по элементам поверхности сама по себе далеко не нова. Она поднималась во множестве предыдущих работ: [1, 2, 3, 4, 5, 6]. Эти работы в основном решают задачу компактного хранения изображения и устранения супенчатости (анти-алиасинг). Мы рассмотрим их более подробно в обзоре. Однако, если принять во внимание широкий спектр современных приложений, существующие методы будут иметь ряд ограничений, поскольку мы хотели бы решать не одну, а сразу несколько проблем в комплексе:

1) Проблема алиасинга остаётся актуальной на сегодняшний день в приложениях компьютерной графики по ряду причин. Здесь есть по крайней мере один проблемный случай – это т.н. спекулярный (от англ. “specular”) алиасинг, который проявляется когда яркие но маленькие объекты попадают внутрь пикселя. Чтобы корректно показать такие изображения на LDR мониторах, необходимо сначала сохранить изображение в увеличенном разрешении (или с какой-то субпиксельной информацией) в HDR, затем в увеличенном разрешении применить алгоритм тон-маппинга (т.н. тонирующий оператор), и лишь затем даунсэмплить полученное изображени до исходного разрешения. В противном случае яркий объект засвечивает весь пиксель, и появляется алиасинг.

2) Проблема увеличения разрешения изображений. Зачастую мы хотели бы иметь изображения с динамическим разрешением. То есть, рендерить кадр в низком разрешении на слабом по производительности мобильном телефоне, а отображать его на экране телевизора с большим разрешением (8K). Или поближе рассмотреть какие-то детали в уже посчитанном фотореалистичном изображении без необходимости повторого расчёта (т.к. это может быть очень долго).

3) Шумоподавление в применение к Монте-Карло рендерингу так-же нуждается в субпиксельной информации, потому что в большинстве существующих методов граничные пиксели обрабатывается хуже чем неграничные. С учётом того что трассировка лучей с последующим применением денойзинга всё глубже поникает в приложения реального времени, проблема аккуратного но быстрого денойзинга является актуальной.

4) Наконец, новое направление дифференцируемого рендеринга требует информацию о границах объектов внутри писелей для реализации т.н. сэмплирования вдоль границ объектов (“edge-sampling”). Этот алгоритм является неотъемлемой частью рассчёта производных от интеграла по пикселу на основе транспортной теоремы Рейнольца, которая позволяет корректно обрабатывать разрывы [7].

2. Обзор существующих методов

2.1. Основные понятия анти-алиасинга

Наиболее затратным и по ресурсам и времени вычисления методом анти-алиасинга является супер-смпэлинг, (Super-Sampling Anti-Aliasing, SSAA), который можно считать эталоном. Изображение отрисовывается в большем разрешении, а затем разрешение уменьшается до требуемого путём усреднения пикселей.

Менее требовательным к вычислительным ресурсам является мульти-сэмплинг (Multi-Sample Anti-Aliasing , MSAA) [8]. Для каждого треугольника проверяется не только пересечение с центром пикселя, но и с несколькими другими точками внутри пикселя. Это аналогично нескольким сэмплам внутри пикселя. Если все сэмплы лежат внутри треугольника, пиксель обрабатывается как и при обычной отрисовке сцены, иначе данные о пересеченном тругольнике созраняются только для сэмплов, которые лежат внутри треугольника. В худшем случае метод требует столько же памяти, как и SSAA при аналогичном количестве сэмплов на пиксель, но на практике количество используемое памяти гораздо меньше, чем в худшем случае. MSAA требует выполнения отдельного прохода, который усредняет значения хранящиеся в одном пикселе. Данный метод решает проблему антиалиасинга на границах объектов, но не решает обозначенную во введении проблему спекулярного алиасинга.

Таким образом, если говорить о компьютерной графике реального времени, основное отличие MSAA от супер-сэмплинга (SSAA) заключатеся в том, что в MSAA для каждого треугольника шейдер выполняется только 1 раз. Проведём аналогию с рендерингом на основе Монте-Карло трассировки лучей и сделаем сноску на то, что именно мы понимаем под примененем MSAA для Монте-Карло трассировки лучей:

- При применении супер-сэмплинга (SSAA) в трассировке лучей увеличивается как число пикселей, так и число Монте-Карло сэмплов. Это означает, что если мы хотим изображение в 2 раза большего разрешения, нам придётся как минимум в 4 раза дольше его считать.
- При применении мульти-сэмплинга (MSAA) в трассировке лучей мы увеличиваем только число субпикселей изображения везде или в некоторых местах, не изменяя итоговое число сэмплов. Таким образом, время рассчёта не изменяется, но растёт потребление памяти.

Необходимо отметить, что в таком понимании **MSAA всё-же будет решать проблему спекулярного алиасинга**, если тон-маппинг применяется к изображению в увеличенном разрешении. Например, используется глобальный тонирующий оператор вроде Рейнхарда [9], работающий отдельно над каждым суб-пикселем.

2.2. Суб-пиксельные представления для анти-алиасинга

Одним из наиболее близких аналогом нашему методу является метод DCAA [4]. Он использует идею о том, что различные сэмплы внутри пикселя в действительности могут попасть на одну поверхность. Поэтому их можно сгруппировать в небольшое число “консолидированных поверхностей” или, что то же самое, агреггированных сэмплов. Это позволяет получить улучшенный анти-алиасинг по сравнению с MSAA за счёт “угадывания” или “восстановления” поверхности, что не было достигнуто в предыдущих работах [1, 2, 3], также использующих идею аггрегирования субпикселей. Например, AGAA [3] использует специфику растеризации чтобы уменьшить количество вызовов фрагментного шейдера; работы [1, 2] также направлены в основном на компактное представление и производительность.

В некотором смысле метод DCAA можно считать предшественником предложенного метода. Например, так же как и предложенный метод, он использует спроецированные в пространство экрана рёбра треугольника для того чтобы детектировать возможную границу поверхности. Однако далее он генерирует битовую маску (64 бит) для каждого треугольника, попадающего в пиксел чтобы сохранить для субпиксела т.н. "карту покрытия" (coverage map) пиксела треугольником. То есть представляет субпиксел при помощи значения и битовой маски — карты покрытия. Это подходит для решения задачи антиалиасинга, но хуже подходит для решения других обозначенных во введении проблем чем предложенный нами метод, поскольку квантует субпиксельное представление.

Следующим близким аналогом нашего метода является метод CASAA [5]. Он является в некоторой степени развитием метода DCAA [4] и работ [1, 2, 3]. CASAA использует аналитическое представление для суб-пиксела (называемых в работе [5] агрегатами) в виде прямых, определяющих пересечение треугольника с пикселом и непрерывных интервалов по глубине. Такое представление позволяет лучше решать проблему "проникающих" друг в друга треугольников, с которыми предыдущие подходы плохо справляются. Данный метод, как и DCAA, предназначен для работы с треугольниками, но при этом ещё более "заточен" под растеризацию: авторы [5] даже проводят параллель с порядко-независимой прозрачностью. На наш взгляд это затрудняет широкое применение данного метода для задач, описанных во введении.

Метод [6] тоже использует идею сохранения суб-пиксельных границ. Как и в нашем методе, рёбра восстанавливаются на основании информации из MSAA-сэмплов внутри пиксела. В отличие от нашего метода рёбра восстанавливаются на основании некоторой статистики, известных характеристик пирамиды видимости и специально-разработанного фильтра, рассматривающего дискретное число возможных расположений ребра внутри пиксела. Как и в методе DCAA, дискретность на наш взгляд ограничивает применимость метода для более широкого класса задач.

2.3. Другие методы анти-алиасинга

Методы основанные на *морфологии* (такие подходы называют иногда "MLAA" или MorphoLogical Anti Aliasing) [10, 11] могут применяться как отдельный этап пост-обработки без использования дополнительной памяти или дополнительного расчёта. Границы объектов опредяются по яркости соседних пикселей и сглаживание применяется вдоль найденных границ. Из-за ограниченности информации, используемой данными подходами, на изображении могут проявляться нежелательные артефакты. Кроме того они не решают проблему алиасинга между кадрами.

В последнее время наибольшее распространение получили методы антиалиасинга основанные на *темпоральной репроекции* (Temporal Anti-Aliasing, TAA) [12, 13]. Для их использования необходимо хранить предыдущий кадр и карту скоростей. Данные предыдущего кадра репроецируются на новый кадр с учетом карты скоростей построенной относительно камеры. Данные между кадрами адаптивно смешиваются. Помимо необходимости хранения двух дополнительных полноэкранных текстур, эти методы имеют недостатки, связанные с их темпоральной природой. Объекты могут оставлять следы при движении или размываться. При работе с TAA требуется настройка алгоритма

для достижения приемлемого качества сглаживания при низком количестве артефактов.

Аналитический анти-алиасинг (NSAA) [14] использует симметричность т.н. фильтр-функции. Он предполагает, что вклад от каждого сэмпла осуществляется не в 1 пиксел (т.н. бокс-фильтр), а в 9 соседних пикселей с использованием гауссианы (то есть гаусс-фильтр). Это позволяет аналитически вычислять вклад от каждого треугольника в пиксел, основываясь на том, какой участок пиксела перекрыт треугольником. Метод применим только к растеризации треугольников и только с условием использования гауссовой фильтр-функции.

2.4. Методы повышения разрешения

В работе [15] предложен метод увеличения разрешения при растеризации геометрии. Каждый пиксель хранит ссылку на список треугольников, которые пересекаются в данном пикселе. Освещение вычисляется для каждого элемента в списке. На заключительном этапе отрисовки, генерируется итоговая текстура. Если пиксель изображения в исходном разрешении содержит список элементов размера 1, то этот элемент занимает все соответствующие пиксели итогового изображения. Иначе, используя данные о геометрии треугольников, разным пикселям назначаются разные элементы списка. Не смотря на схожесть генерации сэмплов в этом методе с методом MSAA, он не решает задачу алиасинга и требует особого подхода к генерации сэмплов. Метод работает только с геометрией заданной треугольниками.

В последние годы производители аппаратного обеспечения активно развивают методы увеличения разрешения на основе нейронных сетей. Так Nvidia разработала метод Deep Learning Super Sampling [16], позволяющий увеличить разрешение до двух раз. Аналогичная технология аннонсирована компанией Intel, но на данный момент она ещё не представлена пользователям. Принципы работы у этих методов аналогичны методу темпорального увеличения разрешения основанного на темпоральном анти-алиасинге [12] и методу темпорального супер-разрешения [13] от компании Epic Games, за исключением того, что логика алгоритма реализована средствами моделей машинного обучения, а не традиционными языками программирования. Аналогичную идею использует работа [17].

Все эти методы темпоральные и применимы только для растеризации. Они требуют наличия карты скоростей для пикселей и используют информацию с предыдущих кадров. Как следствие это приводит к артефактам для полупрозраных объектов и не применимо для приложений, в которых нет возможности накапливать информацию между кадрами.

Компания AMD реализовала метод увеличения разрешения FidelityFX Super Resolution [18], который не требует наличия истории кадров для применения, но требует отдельной реализации метода анти-алиасинга, артефакты которого проявляются заметно сильнее после увеличения разрешения. Также была анонсирована новая версия FidelityFX Super Resolution 2.0 [19], которая реализует метод темпорального анти-алиасига, но на данный момент данная технология недоступна для использования. Кроме того, алгоритмы разработанные компаниями Nvidia и AMD могут быть использованы только с аппаратным обеспечением соответствующих производителей видеокарт.

Традиционные нейросетевые методы повышения разрешения [20, 21, 22, 23] обычно ориентированы на работу с одним изображением (без какой-либо дополнительной информации от рендера) и являются ресурсо-ёмкими из-за большого размера обученной сети. При адаптации этих подходов к компьютерной графике реального времени их качество падает т.к. приходится уменьшать размер нейросети. Например, первая версия DLSS давала очень размытые изображения.

Описанные выше нейросетевые методы можно применять для повышения разрешения изображений, полученных Монте-Карло трассировкой лучей, как и на любых других изображениях – в этом их достоинство. Однако, не обладая реальной субпиксельной информацией, при увеличении разрешения нейросетевые методы дают сильно размытые изображения.

Необходимо отдельно отметить работу [24], которая подходит к вопросу повышения разрешения с точки зрения изменения способа представления изображения. Нейросеть учится находить непрерывные участки на изображении в виде специальных функций, называемых “Local Implicit Image Function (LIIF)”, и, таким образом учится восстанавливать субпикельную структуру объектов внутри изображений. Мы считаем эту работу чрезвычайно перспективной, однако полагаем, что для ряда приложений компьютерной графики её применение в данный момент ограничено. Например, мы не можем использовать LIIF предствление для накопления отдельных Монте-Карло сэмплов (и прогрессивного отображения накапливаемого результата на экране) так как это можно делать с MSAA или предложенным методом, и мы также не можем растеризовать в него треугольники. Мы полагаем что такое применение возможно в будущем, но это требует отдельного исследования.

3. Предложенный метод

Предлагаемый метод подходит как для изображений, построенных при помощи растеризации, так и для изображений, полученных трассировкой лучей (детерминированной или Монте-Карло трассиовкой). Не ограничивая общности рассмотрим второй вариант т.к. случай растеризации на практике более прост — информация о потенциальных границах (рёбра треугольника) непосредственно доступна во время растеризации и, кроме того, мы, как правило, ограничены треугольниками. В трассировке лучей геометрические примитивы могут быть гораздо более разнообразными.

3.1. Построение BSP-изображений

Предлагаемый метод состоит из следующиш шагов:

1. На первом шаге мы производим трассировку лучей через каждый пиксел с некоторым небольшим числом сэмплов на пиксел (от 1 до 4) (рис 1, шаг 1).
    - Это могут быть лучи, проходящие через центр пиксела (1 сэмпл на пиксел), центр и границы пикселей (2 сэмпла на пиксел) или любые другие варианты.
2. Мы определяем т.н. *функцию индекса поверхности*, которая кодирует информацию о поверхности в некотором небольшом числе бит (32–64). Задача этой функции –

возвращать одинаковый индекс для одинаковых участков поверхности или каких-то других желаемых областей на изображении, которые мы не хотим дробить. На рисунке 1 разные значения индекса поверхности показаны кружочками и звёздочками.

- В нашей работе для упрощения интерпретации результата эта функция возвращала индеск объекта (меша). Поэтому, например, на сцене hairballs внутри шарика разбиение не производилось – мы видим результат подразбиения только на границах.
- На практике логично, как минимум, дополнительно использовать нормаль и глубину, чтобы детектировать разрыв поверхности аналгично с [1].
- Если бы мы захотели аккуратно обрабатывать границы бликов от ярких источников, в эту функцию стоило бы добавить несколько бит от значения оценки функции яркости изображения.
- Для дифференцируемого рендеринга с поддержкой, например, зеркальных отражений может потребоваться больше бит т.к. при наличии отражений или преломллений в эту функцию придётся сохранять информацию о следующих поверхностях.

3. Для каждого пиксела мы проверяем, является ли одинаковым значение функции индекса поверхности для всех сэмплов внутри этого пикса и во всего его соседях (8 штук).
4. Для всех пикселей, у которых хотя бы для 1 сэмпла значение функции индекса поверхности отличается от остальных проверенных сэмплов мы трассируем дополнительное и достаточно большое число лучей (16–256); риссунок 1, шаги 2,3.
5. На последнем этапе мы строим внутри пиксела небольшое BSP дерево, используя дополнительную информацию из сэмплов (рис 1, шаг 4).

    - Сначала мы кластеризуем сэмплы по значению функции индекса поверхности, и для каждого сэмпла вытаскиваем вершины всех треугольников (если там были треугольники), с которыми было найдено пересечение.
    - Мы проверяем, являются ли спроецированные в пространство экрана рёбра этих треугольников хорошими кандидатами для разбиения пиксела вычисляя расстояние от каждого сэмпла до прямой. Поскольку геометрия не обязательно может быть представлена треугольниками и по некоторым другим причинам рёбра могут не давать качественного разбиения. Тогда мы переходим к следующему шагу в этом списке.
    - Для всех оставшихся писелей мы используем машину опорных вектов [25]. в двумерном пространстве чтобы найти наилучшие разбивающие прямые и продолжаем строить BSP дерево до фиксированной глубины. Запомним эту деталь. Поскольку машина опорных векторов может работать в многомерном пространстве, в будущем это свойство будет полезно для дифференцируемого рендеринга чтобы сэмплировать вдоль границ в многомерном пространстве.

Таким образом, для всего изображения мы дополнительно сохраняем хэш-таблицу граничных пикселей, а для каждого граниченого пикселя – неглубокое BSP дерево. После

построения изображения с пикселями хранящими BSP-деревья, мы можем использовать его как некоторое непрерывное хранилище, индексируемое двумя координатами с плавающей точкой в интервалле от 0 до 1: $access(x, y)$. Для любых двух координат функция access будет возвращать ссылку либо на одиночный пиксел, либо на одиночный суб-пиксел, в который попадает точка с координатами (x, y).

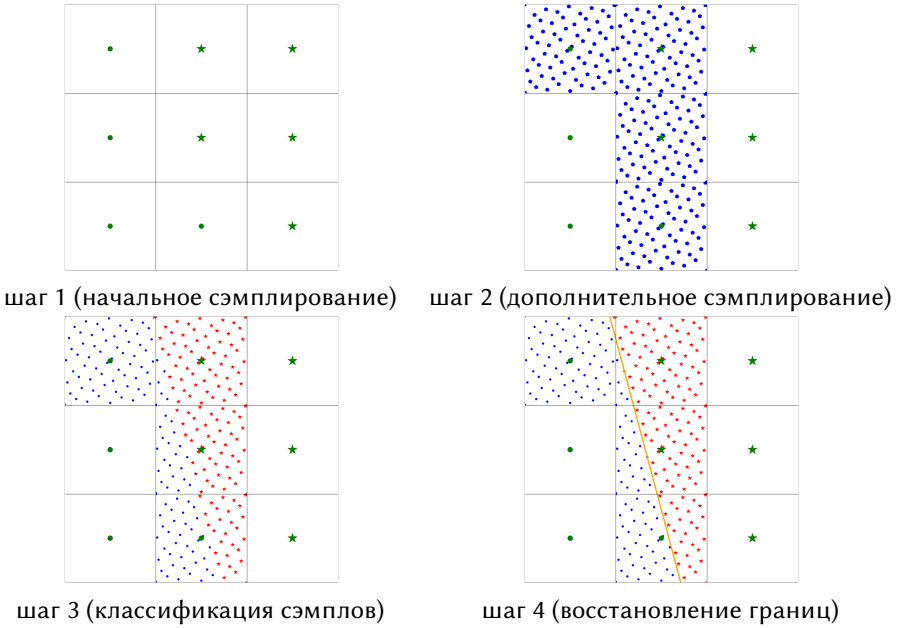

Рисунок 1 – Иллюстрация работы предложенного алгоритма

3.2. Использование для буфера кадра

Мы можем использовать функцию access, например, для накопления результатов Монте-Карло трассировки лучей в буфере кадра и последуюшей визуализации любой области изображения с повышенным разрешением. Однако на практике только лишь 2 координат x и y недостаточно для этой цели, поскольку прямые, полученные нашим подходом являются апроксимациями, а не точными границами объектов. Для корректного учёта вклада мы должны дополнительно для каждого Монте-Карло сэмпла вычислять не только его цвет (color), но и функцию индекса поверхности (surfId), и сравнивать её значение со значением функции индекса поверхности в суб-пикселе. Это необходимо делать для того, чтобы аккмулировать внури субпиксела только те сэмплы, которые принадлежать одной и той же поверхности.

```
auto& subPixel = frameBuffer.access(sam.x, sam.y);
if(subPixel.surfId == sam.surfId)
{
  subPixel.color += sam.color;
```

```
    subPixel.hits++;
}
```

Листинг 1. Отбраковка сэмплов, содержащих неподходящий идентификатор пов-ти.

Таким образом, Монте-Карло сэмплы, которые попали в неправильный суб-пиксель в силу апроксимации нашего решения не будут учитываться при формировании изображения. К счастью это чрезвычайно редкие случаи, поэтому на точность рассчёта они не влияют. Тем не менее, при желании всё-таки можно учитывать вклад от такого сэмпла в какой-то другой ближайший субпиксел (из текущего или соседнего пикселя), у которого индекс поверхности совпадает с индексом поверхности сэмпла.

3.3. Использование для дифференцируемого рендеринга

Одина из ключевых проблем дифференцируемого рендеринга – эффективное сэмплирование вдоль границ объектов для обработки разрывов. В традиционном edge samplig-е [7] предлагается запоминать все рёбра треугольного меша или строить многомерные ускоряющие структуры, чтобы затем производить сэмплирование вдоль всех потенциальных границ. При этом, если говорить о сэмплировании в 2D пространстве, количество сэмплов на 1 проход должно быть как минимум сравниммо с числом пикселей всего изображения, поскольку зарнее неизвестно, какие же именно рёбра будут граничными, а какие нет (для тех ребёр которые не будут граничными, разница будет нулевой и они не внесут вклад в расчёт градиента). Эта проблема особенно остро стоит, если для представления геометрии используются какие-либо другие примитивы кроме треугольников (например, SDF функции или CSG геометрия), т.к. для таких представлений не очень просто найти их границы.

В предложенном подходе границы восстановлены с высокой точностью, а в субпикселах уже содержится проинтегрированое значение функции от нескольких сэмплов. Для того чтобы получить разницу вдоль границ, неободимо перечислить все граничные пиксели изображения и, спускаясь рекурсивно по BSP дереву, вычислять разницу между всеми соприкасающимися листьями. За счёт этого, в предложенном методе за счёт использования машины опорных векторов мы можем работать с любым типом геометрических примитивов.

Кроме того, мы можем относительно-легко расширить метод на многомерный случай (когда нужно учитывать вклад в градиент от глобального освещения). Для этого мы будем рассматривать многомерное уравнение гипер-плосокости вместо двумерного, но принциц и алгоритм построения изображения не изменятся. Апробация предложенного подхода для дифференцируемого рендеринга является темой наших будущих исследований.

4. Сравнения

Для задачи устранения ступенчатости мы сравнили предложенный метод с Multi-Sampled Anti Aliasing или MSAA (16xMSAA, рисунок 3). Как можно видеть из этого рисунка, предложенный метод решает проблему алиасинга на ярких объектах, так же

Таблица 1
Сравнение предложенного метода с MSAA и DCAA [4] по дополнительным затратам памяти в процентах от исходого объёма памяти, занимаемого изображением без субпикселей. Стрелка вниз у значения указывает, что лучшее значение – меньшее в строке. Для метода DCAA мы использовали оценку, исходящую из того что он сохраняет фиксированно 4 суб-пикселя для каждого граничного пикселя без учёта 64-битных масок. Сравнение с учётом дополнительной информации представлено в таблице 2.

сцена	граничных пикселей	узлов BSP дерева	рост памяти в % предлож. метод	рост памяти в % 16xMSAA	рост памяти в % DCAA [4]
bunny	6847	6937	**+0.66%**↓	+10.5%↓	+2.62%↓
self-illum	9785	10035	**+0.96%**↓	+14.9%↓	+3.75%↓
hairballs	179975	2663025	+254.0%↓	+273.0%↓	**+68.25%**↓

как MSAA (*При растеризации в графическом конвейере MSAA не решает эту проблему, как мы обсуждали во введении). При этом он не вносит дополнительных искажений даже для довольно сложной геометрии (последняя строка). Однако, по сравнению с методом MSAA предложенный метод потребляет на прядок меньше дополнительной памяти на сценах, где границы внутри пиксела проходят между небольшим числом объектов (Таблица 1). Интересно отдельно отметить вторую строку (рисунок 3), где в исходном разрешении алиасинг в HDR изображении после применения тонирующего оператора приводит к цветовым искажениям в LDR изображении (красные пиксели по границе жёлтого объекта). Кроме того, мы сравнили разичные методы по затратам памяти на 1 пиксел (таблица 2).

сцена bunny сцена self-illum сцена hairballs

Рисунок 2 – Тестовые сцены. Небольшие красные квадраты отмечают участки, которые демон-стрируются далее с увеличением на рисунки 3 и 4

Для задачи повышения разрешения наряду с MSAA мы дополнительно сравнили предложенный подход с нейросетовым методом LapSRN [23], доступный в библиотеке OpenCV (рисунок 4, таб. 3). Этот метод был выбран т.к. он имеет обученную модель для сильного повышения разрешения (до 8 раз). Для того чтобы получить итоговое

Таблица 2
Сравнение предложенного метода с аналогами (DCAA [4] и CASAA[5]) с учётом дополнительной информации, которую необходимо хранить на 1 пиксель. Будем считать, что в одном субпикселе изображения хранится 4 байта или 32 бита (R8G8B8A8), то есть это LDR изображение, и будем записывать информацию в ячейке в формате "A+B" где A – память под хранилище субпикселей, а B – дополнительная информация. Для каждого из методов мы демонстрируем результаты для двух, приблизительно эквивалентных уровней качества (то есть MSAA (16x) эквивалентно DCAA (2x), а MSAA (64x) эквивалентно DCAA (4x)). Предложенный метод выделен жирным шрифтом (BSPAA). Мы предполагаем что прямые хранятся без сжатия в нормализованном представлении в виде 2 чисел с плавающей точкой в формате IEEE 754 (8 байт).

Метод	что сохраняет	потребляет памяти на пиксел (в байтах)	итого байт на пиксел
MSAA (16x)	16 субпикселей	16*4 + 0	64
DCAA (2x)	2 субпиксела + 1 маска	2*4 + 8	16
CASAA (2x)	2 субпиксела + прямая и глубина	2*4 + 12	20
BSPAA (2x)	2 субпиксела + прямая	2*4+8	16
MSAA (64x)	64 субпикселя	64*4 + 0	256
DCAA (4x)	4 субпиксела + 2 маски	4*4 + 8*2	32
CASAA (4x)	4 субпиксела + 3 прямых и 3 глубины	4*4 + 3*12	52
BSPAA (4x)	4 субпиксела + 3 прямых	4*4+3*8	40

увеличение разрешения в 16 раз мы применяли две обученные модели метода LapSRN последовательно – сначала в 8 раз, а затем ещё в 2 раза. Из этого сравнения можно сделать несколько выводов:

1) Применение MSAA для Монте-Карло рендеринга порождает визуальные артефакты в виде шума вдоль границ. Это является прямым следствием идеи работы MSAA: увеличение хранилища (в виде добавления субпикслей) без увеличения числа фактических сэмплов. Для его устранения необходимо не только увеличивать число субпикселей (что делает MSAA), но и увеличивать количество сэмплов на пиксел вдоль границ (что делает классический супер-сэмплинг), чтобы все субпиксели получили одинаковое количество сэмплов. Однако **это не всегда можно сделать**, т.к. двунаправленные методы рассчёта освещения (BPT) и даже, например, однонаправленный Metropolis Light Transport в принципе не позволяют локально контролировать количество сэмплов на пиксел – можно увеличить только общее число сэмплов на всё изображение.

2) Нейросетевой метод повышения разрешения, не имея информации о точных границах объекта делает переход размытым. Кроме того, он добавляет искажения в виде волнистой границы, которых быть не должно. При большом количестве субпиксельных деталей (последняя строка на рисунке 4), нейросетевой метод в чистом виде можно считать неприменимым. При этом, если предложенный метод и MSAA позволяют сохранять более детальную информацию вдоль границ объектов, то нейросетевой метод в действительности решает другую задачу и дополнительно сглаживает изображение. Мы хотели бы отметить, что нейросетевое повышение разрешения и предложенный метод **не является конкурентами**, а скорее могут взаимно дополнять друг друга. Для

Таблица 3
Сравнение предложенного метода по метрике PSNR/SSIM для задачи повышения разрешения

Номер строки	16xMSAA	Предложенный метод	LapSRN
1	17.73↑/**0.546**↑	17.77↑/0.544↑	**18.23**↑/0.543↑
2	13.83↑/0.261↑	13.82↑/0.261↑	**14.47**↑/**0.270**↑
3	13.38↑/0.264↑	**13.42**↑/**0.267**↑	13.27↑/0.262↑
4	26.99↑/0.383↑	**27.02**↑/0.384↑	27.01↑/**0.386**↑
5	**18.46**↑/**0.253**↑	18.27↑/0.234↑	16.92 ↑/0.107↑

полученя гладкого изображения (без шума) можно было бы применить нейросетевой метод повышения разрешения или обыкновенный билатеральный фильтр шумоподавления поверх нашего подхода или MSAA чтобы сгладить изображение при увеличении разрешения. Это тема для будущих исследований.

3) Анализируя изображения из последней строчки можно заменить, что не все пиксели тёмно-красного шарика (слева) подразбились методами MSAA и предложенным методом. Это связано не с ограничениями этих методов, а с нашей текущей реализации функции вычисления индекса поверхности, которая зависит только от индекса объекта. Поэтому все отдельные волосинки красного шарика считались за единый объект. Этой проблемы можно избежать если, например, закодировать нормаль в индекс поверхности.

5. Благодарности

Работа выполнена по проекту “Кросс-платформенная, аппаратно-ускоренная среда дифференцируемого моделирования”, финансируемого некоммерческим фондом ИНТЕЛЛЕКТ в Москвоском Государственном университете имени М.В. Ломоносова в 2022 году.

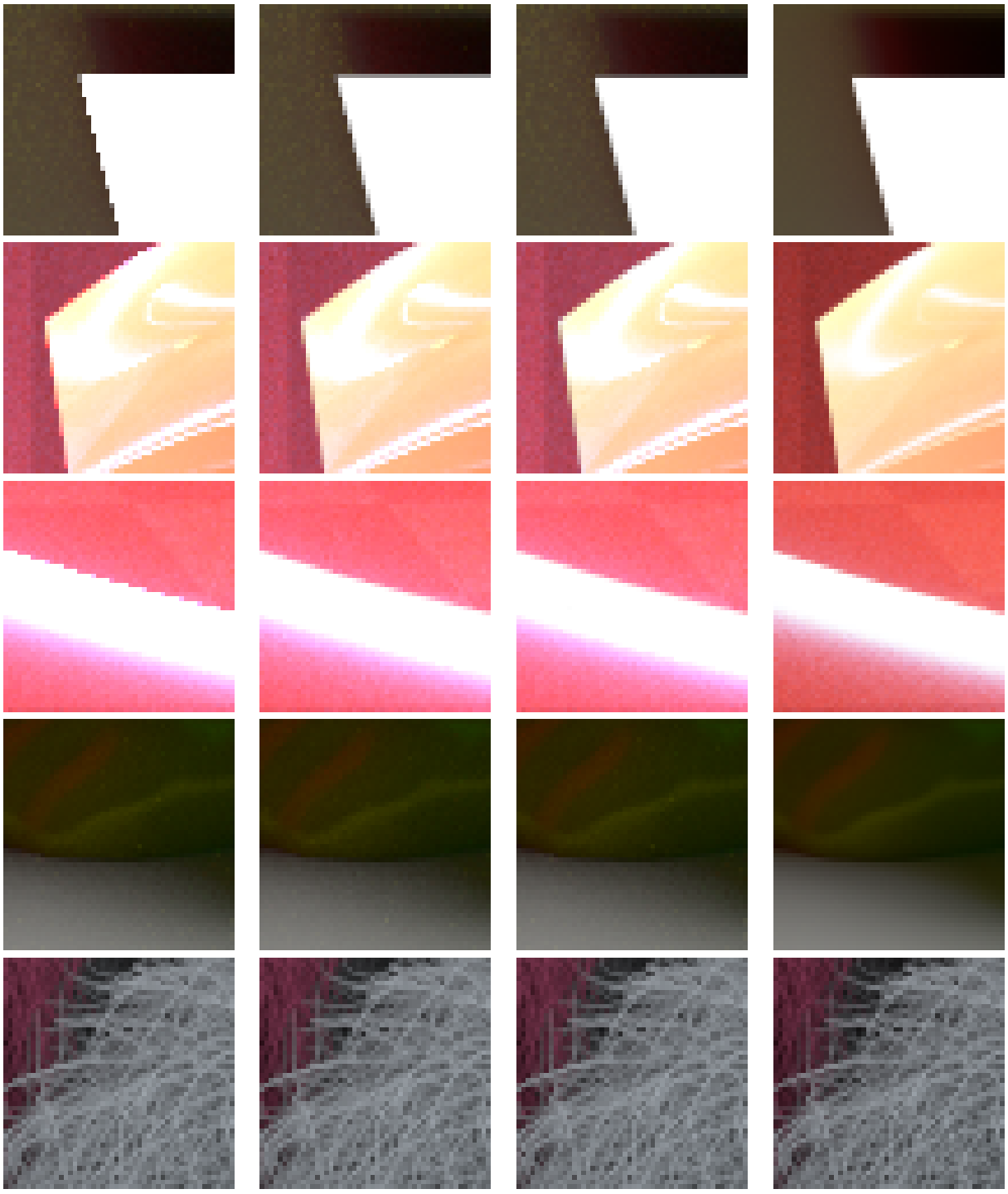

Рисунок 3 – Визуальное сравнение предложенного метода с MSAA и эталоном в задаче устранения ступенчатости.

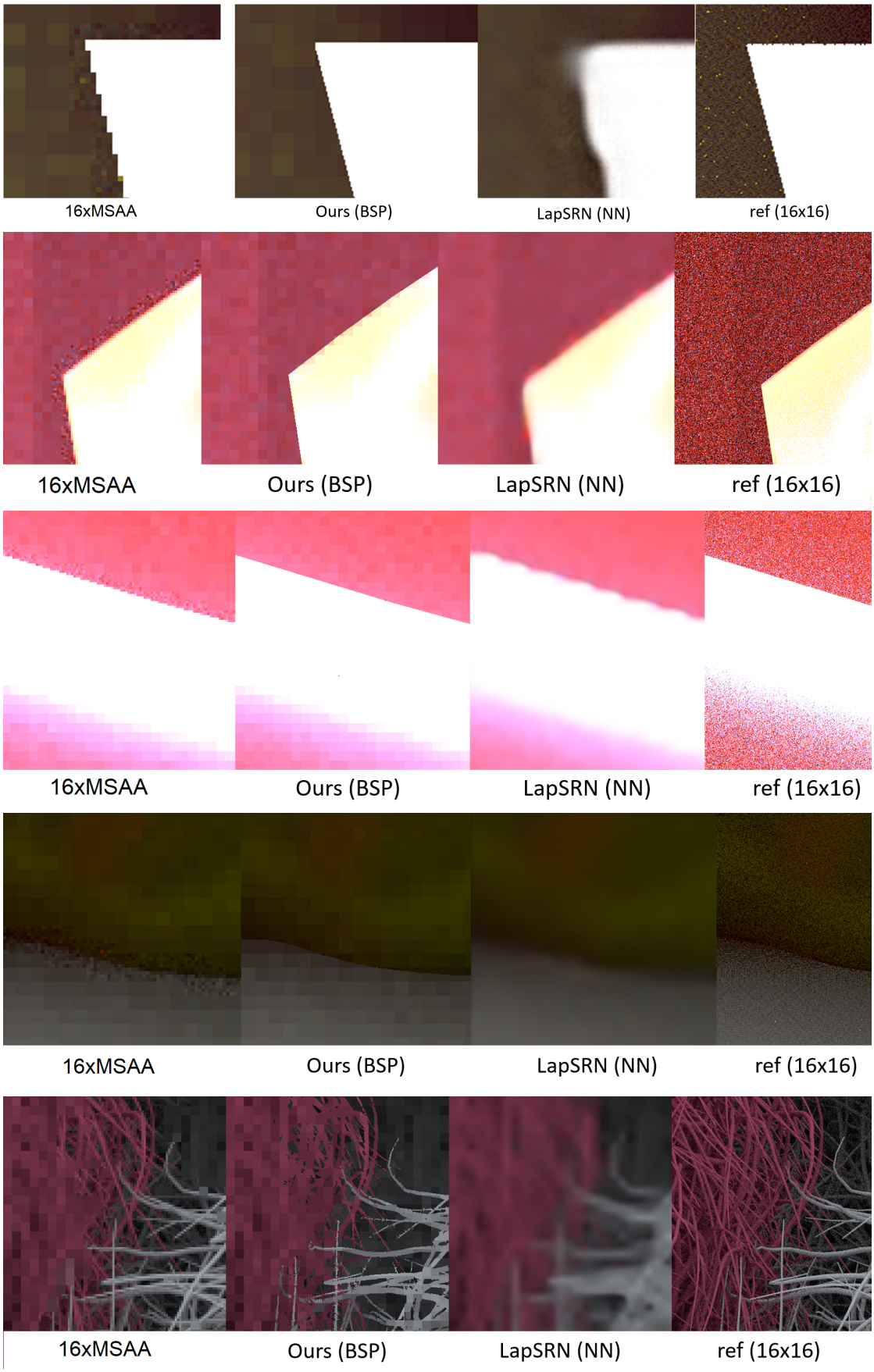

Рисунок 4 – Визуальное сравнение предложенного метода с аналогами и эталоном в задаче повышения разрешения изображений.

Список литературы

[1] M. Salvi, K. Vidimče, Surface based anti-aliasing, in: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '12, Association for Computing Machinery, New York, NY, USA, 2012, p. 159–164. URL: https://doi.org/10.1145/2159616.2159643. doi:10.1145/2159616.2159643.

[2] E. Kerzner, M. Salvi, Streaming g-buffer compression for multi-sample anti-aliasing, in: Proceedings of High Performance Graphics, HPG '14, Eurographics Association, Goslar, DEU, 2014, p. 1–7.

[3] C. Crassin, M. McGuire, K. Fatahalian, A. Lefohn, Aggregate g-buffer anti-aliasing - extended version-, IEEE Transactions on Visualization and Computer Graphics 22 (2016) 2215–2228. doi:10.1109/TVCG.2016.2586073.

[4] Y. Wang, C. Wyman, Y. He, P. Sen, Decoupled Coverage Anti-Aliasing, in: P. Clarberg, E. Eisemann (Eds.), High-Performance Graphics, ACM Siggraph, 2015. doi:10.1145/2790060.2790068.

[5] C. Crassin, C. Wyman, M. McGuire, A. Lefohn, Correlation-aware semi-analytic visibility for antialiased rendering, in: Proceedings of the Conference on High-Performance Graphics, HPG '18, Association for Computing Machinery, New York, NY, USA, 2018. URL: https://doi.org/10.1145/3231578.3231584. doi:10.1145/3231578.3231584.

[6] D. Luo, J. Zhang, The sharp-filter anti-aliasing based on sub-pixel continuous edges, Mathematics 8 (2020). URL: https://www.mdpi.com/2227-7390/8/10/1655. doi:10.3390/math8101655.

[7] T.-M. Li, M. Aittala, F. Durand, J. Lehtinen, Differentiable monte carlo ray tracing through edge sampling, ACM Trans. Graph. 37 (2018). URL: https://doi.org/10.1145/3272127.3275109. doi:10.1145/3272127.3275109.

[8] Multisampling, 2021. URL: https://www.khronos.org/opengl/wiki/Multisampling.

[9] E. Reinhard, M. Stark, P. Shirley, J. Ferwerda, Photographic tone reproduction for digital images, ACM Trans. Graph. 21 (2002) 267–276. URL: https://doi.org/10.1145/566654.566575. doi:10.1145/566654.566575.

[10] T. Lottes, Fxaa, 2010. URL: https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf.

[11] J.-H. Nah, S. Ki, Y. Lim, J. Park, C. Shin, Axaa: Adaptive approximate anti-aliasing, in: ACM SIGGRAPH 2016 Posters, SIGGRAPH '16, Association for Computing Machinery, New York, NY, USA, 2016. URL: https://doi.org/10.1145/2945078.2945129. doi:10.1145/2945078.2945129.

[12] L. Yang, S. Liu, M. Salvi, A survey of temporal antialiasing techniques, Computer Graphics Forum 39 (2020) 607–621. doi:10.1111/cgf.14018.

[13] Temporal super resolution, 2021. URL: https://www.unrealengine.com/en-US/features/temporal-super-resolution.

[14] T. Auzinger, P. Musialski, R. Preiner, M. Wimmer, Non-Sampled Anti-Aliasing, in: M. Bronstein, J. Favre, K. Hormann (Eds.), Vision, Modeling and Visualization, The Eurographics Association, 2013. doi:10.2312/PE.VMV.VMV13.169-176.

[15] J. Kang, Scalable Dynamic Rasterization for Postprocessing, Master's thesis, Department of Computer Science and Engineering The Graduate School, Sungkyunkwan University,

2021.

[16] E. Liu, Dlss 2.0 – image reconstruction for real-time rendering with deep learning, 2020. URL: http://behindthepixels.io/assets/files/DLSS2.0.pdf.

[17] L. Xiao, S. Nouri, M. Chapman, A. Fix, D. Lanman, A. Kaplanyan, Neural supersampling for real-time rendering, ACM Trans. Graph. 39 (2020). URL: https://doi.org/10.1145/3386569.3392376. doi:10.1145/3386569.3392376.

[18] Fidelityfx super resolution, 2021. URL: https://www.amd.com/ru/technologies/fidelityfx-super-resolution.

[19] Fidelityfx super resolution 2.0, 2022. URL: https://gpuopen.com/fidelityfx-superresolution-2/.

[20] B. Lim, S. Son, H. Kim, S. Nah, K. M. Lee, Enhanced deep residual networks for single image super-resolution, 2017, pp. 1132–1140. doi:10.1109/CVPRW.2017.151.

[21] W. Shi, J. Caballero, F. Huszár, J. Totz, A. Aitken, R. Bishop, D. Rueckert, Z. Wang, Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016. doi:10.1109/CVPR.2016.207.

[22] C. Dong, C. C. Loy, X. Tang, Accelerating the super-resolution convolutional neural network, in: B. Leibe, J. Matas, N. Sebe, M. Welling (Eds.), Computer Vision – ECCV 2016, Springer International Publishing, Cham, 2016, pp. 391–407.

[23] W.-S. Lai, J.-B. Huang, N. Ahuja, M.-H. Yang, Fast and accurate image super-resolution with deep laplacian pyramid networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 41 (2019) 2599–2613.

[24] Y. Chen, S. Liu, X. Wang, Learning continuous image representation with local implicit image function, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 8628–8638.

[25] C. Cortes, V. Vapnik, Support-vector networks, Chem. Biol. Drug Des. 297 (2009) 273–297.