

# Имплементация виртуализации геометрии в рендеринге реального времени

Санду Р.А.\*

7 июня 2021 г.

## Аннотация

Данная работа посвящена решению задачи адаптивного рендеринга в приложениях реального времени. В качестве цели работы была выбрана имплементация иерархического атласа с виртуальным текстурированием, одного из актуальных подходов к решению этой задачи. В результате работы был выявлен и исправлен ряд недостатков и неточностей этой техники, и предложены улучшения оригинального метода. Была написана эффективная многопоточная реализация приложения конвертации моделей в формат иерархического атласа, а также приложение для рендеринга конвертированных моделей с использованием современного графического API “Vulkan”.

---

\*Факультет прикладной математики и информатики, кафедра корпоративных информационных систем 1С, Московский физико-технический институт, Долгопрудный 141700, Россия

# Содержание

<b>1 Введение</b>	<b>3</b>
1.1 Формальная постановка задачи . . . . .	4
1.2 Обзор литературы . . . . .	5
1.2.1 Subdivision surfaces with displacement maps . . . . .	5
1.2.2 Воксельный рейкастинг . . . . .	6
1.2.3 Маскируемые полоски . . . . .	6
1.2.4 Иерархический атлас . . . . .	7
<b>2 Разработанное решение</b>	<b>8</b>
2.1 Предобработка . . . . .	9
2.1.1 Раскладывание треугольников по бакетам . . . . .	9
2.1.2 Кластеризация . . . . .	9
2.1.3 Глобальная и локальная кластеризация . . . . .	11
2.1.4 Распрямление границ . . . . .	13
2.1.5 Квадрангulation . . . . .	13
2.1.6 Репараметризация . . . . .	15
2.1.7 Ресемплинг . . . . .	15
2.2 Рендеринг . . . . .	16
2.2.1 Виртуализация . . . . .	17
2.2.2 Устранение разрывов . . . . .	17
2.2.3 Графический конвейер . . . . .	19
<b>3 Результаты</b>	<b>19</b>
<b>4 Заключение</b>	<b>23</b>

# 1. Введение

С самого зарождения области интерактивной компьютерной 3D графики люди стремились повысить количество примитивов одновременно отображаемых на экране. Это один из самых естественных способов повысить общее качество изображения. Однако, не смотря на стремительное развитие графических ускорителей, наивный подход не даёт удовлетворительных результатов: в современных приложениях реального времени бюджет количества треугольников на экране остаётся достаточно ограниченным. С другой стороны современные программы для скульптурирования, а также различные технологии 3D сканирования, позволяют получить модели количества полигонов которых почти не ограничено. Бюджет приложения может быть насыщен всего лишь одним сканом в высоком качестве.

Исторически для решения этой проблемы было придумано множество техник экономии количества треугольников без потери визуального качества. Самым частым подходом является использование так называемых уровней детализации совместно с различными картами. Уровень детализации – результат геометрического упрощения исходной модели содержащий меньше полигонов. Как правило генерируется несколько различных уровней детализации для модели с числом треугольников отличающимся на порядок, а затем при рендеринге динамически выбирается одна из них в зависимости от расстояния до наблюдателя. Различные карты используются чтобы восстановить визуальное качество модели потерянное за счёт упрощения. Среди часто используемых карт – normal mapping [1], различные формы displacement mapping [2], [3]. Это семейство подходов основано на одном простом наблюдении: чем меньше видимый размер объекта на экране, тем меньше экранных пикселей он занимает, а следовательно тем менее заметны высокочастотные детали модели. Однако наивное использование уровней детализации сталкивается с множеством проблем. В случае если целевая модель много больше размера наблюдателя (например ландшафт), не редка ситуация когда наблюдатель всегда находится вблизи какой-то из частей модели, а следовательно необходимо всегда отображать всю модель в максимальном качестве. В литературе задачу решения этой проблемы называют *задачей аддитивного рендеринга*. Ещё одной проблемой является потребление видеопамяти. Если просто загружать все уровни детализации всех моделей сцены в видеопамять, она достаточно быстро израсходуется. Существует множество техник решающих эту проблему. Стандартным подходом является комбинирование раннего отбрасывания моделей с графического конвейера ([4], [5]) в сочетании с уровнями детализации и асинхронной загрузкой данных в видеопамять. Заметим, что подобные подходы не применимы к крупным объектам по аналогичным причинам: объекты рассматриваются как неделимое целое.

Виртуализация геометрии – один из подходов к решению задачи аддитивного рендеринга. Предлагается хранить в видеопамяти только те части модели, которые необходимо отображать в данный момент, при этом только в том качестве, в котором необходимо. При этом в случае если необходимого качества в данный момент нет, вполне допустимо использовать геометрию в более низком качестве. Таким образом с точки зрения итогового изображения создаётся видимость что вся модель всегда доступна в максимальном разрешении, откуда

название по аналогии с технологией виртуальной памяти используемой в операционных системах. Вариаций в реализации этой идеи достаточно много. В данной работе за основу взята статья [6]. Причиной этому послужил анонс “Unreal Engine 5” [7] летом 2020 года. Было выяснено, что реализованная в нём технология микрополигонального рендеринга “Nanite” тоже берёт своё начало в упомянутой выше статье (согласно [8], блогу одного из разработчиков Nanite). Целью данной работы положим современную эффективную имплементацию идей этой статьи, а также исследование деталей реализации опущенных в оригинальной статье.

## 1.1. Формальная постановка задачи

Введём несколько определений используемых в данной работе.

- Мешем из  $n$  вершин называют пару  $M = (I, A)$ , числа  $1, \dots, n$  называют вершинами,  $I \subset \{1, \dots, n\}^3$  – индексное множество задающее треугольники (тройки вершин), а  $A : \{1, \dots, n\} \rightarrow \mathbb{R}^3 \times \mathbb{R}^n$  – атрибуты вершин, первые 3 компоненты которых считаются координатами в трёхмерном евклидовом пространстве. Меш индуцирует подмножество евклидового пространства как объединение выпуклых оболочек координат треугольников. Ради простоты отождествим треугольник как тройку индексов и выпуклую оболочку его координат, а также меш и индуцированное им множество. Остальные атрибуты меша продолжаются с вершин треугольника на все его точки посредством барицентрической интерполяции. Систему отсчёта в которой заданы координаты вершин меша называют системой отсчёта модели. В данной работе в качестве вершинных атрибутов взяты помимо координат направления нормалей в системе отсчёта модели, сдвинутой в соответствующую точку поверхности, а также координаты параметризации меша, иначе говоря UV-развёртки.
- Дискретным многообразием (с краем) назовём меш, задающий подмножество евклидова пространства, являющееся многообразием (с краем).
- Картой набора треугольников дискретного многообразия образующих замкнутое связное множество назовём гомеоморфизм его с  $[0, 1]^2$  (далее – параметрическое пространство), однозначно задаваемый его значениями в вершинах соответствующих треугольников. Дополнительно потребуем чтобы углы параметрического пространства переходили в вершины меша, а рёбра в наборы рёбер. Таким образом область определения карты задаёт четырёхугольник на поверхности меша.
- Набор карт, области определения которых полностью покрывают всё дискретное многообразие, назовём непрерывным атласом, если любые две области определения пересекаются либо по общему ребру этих четырёхугольников, либо по общей вершине, а также значения гомеоморфизмов разных карт согласованы на точках пересечения их областей определения. Это позволяет определить понятие соседей для карты атласа – карт, граничащих с ней с четырёх сторон её области определения.

- Геометрическим изображением назовём матрицу, элементами которой являются некоторые элементы  $\mathbb{R}^3 \times \mathbb{R}^n$ . Геометрическое изображение задаёт дискретное многообразие (или его часть) посредством отождествления атрибутов вершин и элементов матрицы и взятием множества треугольников как триангуляции прямоугольной решётки.
- Иерархическим атласом [6] дискретного многообразия называют лес квадродеревьев, с каждой вершиной которого связана карта. При этом карты корней деревьев должны образовывать непрерывный атлас, а дети каждой вершины обязаны разбивать родительскую карту на 4 равных квадрата в параметрическом пространстве.
- Разрезом иерархического атласа называют минимальное *по включению* подмножество вершин леса такое, что любой путь из корня дерева в лист проходит по вершине этого множества. Заметим, что подмножество вершин удовлетворяет этому определению тогда и только тогда, когда объединение карт вершин этого подмножества покрывает всё многообразие без наложений.

Задача виртуализации состоит из следующих подзадач: построение непрерывного атласа по имеющемуся дискретному многообразию, построение геометрических изображений для каждой карты атласа, построение mip-уровней геометрических изображений, построение иерархического атласа, адаптация разреза, рендеринг. При этом в результате рендеринга разреза иерархического атласа изображение должно как можно меньше отличаться от изображения исходного многообразия.

## 1.2. Обзор литературы

Рассмотрим несколько перспективных подходов к адаптивному рендерингу и сравним их с выбранным в этой работе.

### 1.2.1. Subdivision surfaces with displacement maps

Subdivision surfaces with displacement maps (далее SSDM, [9]) – достаточно популярная в областях анимации и визуальных эффектов технология. Основная идея subdivision surfaces заключается в задании модели неявным образом как интерполяции набора хранимых явно узлов и рёбер, определяющих топологию поверхности (далее – каркас). При повторной дискретизации интерполированной поверхности появляется возможность выбрать частоту соответствующую плотности экранных пикселей на поверхности для текущего положения камеры. Сама интерполяция происходит на GPU при помощи тесселяционных шейдеров, что позволяет ограничиться передачей в графическую память лишь каркаса. Название displacement mapping же говорит само за себя, каждой точке поверхности сопоставляется вектор-смещение, которые дискретизируются при помощи традиционных текстур. Использование этих идей вместе позволяет добиться динамической смены уровня детализации на близких расстояниях без потери качества картинки, при этом качество ограничено исключительно разрешением используемых текстур. Более того, SSDM полностью анимируемы.

Основные проблемы же этой техники начинаются на средних и дальних дистанциях отрисовки, ведь наиболее разреженной интерполяцией будет служить сам каркас. При рендеринге объектов со сложной топологией необходимо задать эту топологию на уровне каркаса, что может заметно его измельчить. Также к измельчению приводит поддержка скелетных анимаций, так как привязывать к скелету можно только узлы каркаса, а не индивидуальные точки поверхности. В своей статье [10] Брайан Карис, один разработчиков Nanite, приводит как пример модели персонажей в одном из его проектов для которых каркас пришлось сделать настолько измельчённым, что он фактически совпадал с традиционной моделью персонажа на близком уровне детализации. Ещё одним значительным недостатком является специфичность формата используемого этой техникой. Универсального алгоритма конвертации в этот формат нет, и скорее всего никогда не появится, ведь форму и плотность каркаса нужно выбирать учитывая большое количество факторов, в том числе визуальное качество результата, поэтому модели приходится делать изначально в ПО поддерживающем этот формат. Также это ограничивает возможность применения SSDM к 3D-сканам различных объектов и ландшафтов. Дальнейшее ознакомление с SSDM можно начать с [11].

### 1.2.2. Воксельный рейкастинг

Воксельный рейкастинг [12] – техника, заключается в редискретизации статической геометрии по трёхмерной сетке, хранении результата в разреженном октодереве и последующем рендеринге при помощи рейкастинга из каждого экранного пикселя. Также в процессе рендеринга используется трёхмерное виртуальное текстурирование. В отличии от SSDM воксельный рейкастинг не накладывает никаких ограничений на используемые модели и форматы. Производительность не зависит от свойств модели. С помощью вокселей можно хранить любые свойства поверхности и форму геометрии в однородном виде, что упрощает весь пайплайн. Также к трёхмерным текстурам можно применять традиционные алгоритмы компрессии из обработки изображений. Ну и наконец как и SSDM эта техника позволяет выбирать частоту дискретизации основываясь на плотности пикселей, что приводит к хорошему качеству картинки на всех уровнях детализации с приемлемой производительностью. Из явных недостатков же следует упомянуть дороговизну рейкастинга с точки зрения производительности. Также заметим требовательность техники к видеопамяти и необходимость дополнительных надстроек для её экономии. Наконец отметим главный недостаток этой техники – она абсолютно не совместима с традиционными динамическими моделями и анимациями. Как следствие появляется необходимость использовать совершенно иной подход для рендеринга динамических объектов на сцене, что приводит к сложной логике взаимодействия двух фундаментально разных систем.

### 1.2.3. Маскируемые полоски

Ещё один подход к динамическому уровню детализации представлен в [13]. Модель разбивается на набор полосок из треугольников и на этапе препроцессинга генерируется стратег-

гия схлопывания рёбер внутри этих полосок для достижения нужного уровня детализации. Главным преимуществом этой техники является поддержка сабмешей внутри модели с разным набором вершинных атрибутов. Но в отличии от остальных упомянутых алгоритмов маскируемые полоски очень плохо справляются с низкими уровнями детализации. Алгоритм не подразумевает упрощений между несколькими полосками, из-за чего на дальних дистанциях приходится рендерить сильно больше геометрии чем необходимо. Также расширяемость размера полосок лишь в одном направлении не позволяет сильно увеличить их размер, ведь это привело бы к уменьшению качества картинки. Наконец алгоритму необходимо полностью хранить модель в памяти GPU, что ограничивает его применимость к сильно детализированным моделям.

#### 1.2.4. Иерархический атлас

Сравнивая метод представленный в [6] с упомянутыми выше, использование иерархического атласа фактически позволяет избавится от всех упомянутых проблем. Путём небольшой модификации алгоритма описанного в [14] достигается поддержка скелетных анимаций. Также теоретически не должно возникнуть препятствий в использовании морф-анимаций. Используя модификацию [14] размеры карт в атласе могут быть много больше граней каркаса из SSDM, что позволяет лучше адаптировать частоту дискретизации под плотность пикселей на дальних дистанциях. Также в [14] предлагается ряд других усовершенствований позволяющих в сумме добиться десятикратного улучшения качества. Далее, конверсия моделей в формат иерархического атласа полностью автоматизирована и может быть при нужде адаптирована эвристиками под конкретное приложение. К сожалению имплементация самого алгоритма построения иерархического атласа достаточно сложна. Потребление видеопамяти алгоритмом хоть и может быть теоретически ограничено любым значением благодаря виртуализации, но требуемая для достижения фиксированного качества необходимая память растёт квадратично. Однако в воксельном рейкастинге же память растёт кубически. Наконец самое большое преимущество этой технологии – независимость выбора частоты дискретизации геометрии, частоты дискретизации текстур, а также фактического количества частей из которого состоит вся поверхность, иначе говоря количества запусков графического пайплайна. Заменяя эвристики выбора этих параметров можно адаптировать алгоритм под конкретные данные. Как один из недостатков можно упомянуть что выбор частоты дискретизации хоть и достаточно мелкогранулярен благодаря квадродеревьям, но всё равно проигрывает воксельному рейкастингу где частоту можно независимо выбирать для каждого экранного пикселя. Ещё одним недостатком всех упомянутых схем является применимость их лишь к определённому роду геометрии. Например использование их с моделью кованого металлического забора с особо сложным рисунком вряд ли даст прирост в производительности или качестве по сравнению с традиционным рендерингом.

Наконец стоит отметить, что при приближении частоты дискретизации модели треугольниками к частоте дискретизации экрана пикселями начинают появляться проблемы алиасинга и "промахивания" треугольников мимо точки семплирования пикселей. В одном из

интервью Карис упомянул что для решения этой проблемы при использовании иерархического атласа для создания Nanite ([15]) их команде пришлось написать свой программный растеризатор.

	Гранулярность по скринспейсу	Скелетные анимации	Упрощение на высоких дистанциях	Связность
Статические уровни детализации	Нет	Да	Не ограничено	Явная
Иерархический атлас	Сильная	Да	Сильное	Неявная
SSDM	Средняя	Да	Слабое	Явная
Воксельныйрейкастинг	Сильная	Нет	Не ограничено	Неявная
Маскируемые ленты	Нет	Да	Слабое	Неявная

Таблица 1: Сравнение возможностей рассмотренных подходов

	Гранулярность по дистанции	Динамическая детализация	Неоднородные атрибуты	Предобработка
Иерархический атлас	Сильная	Текстуры, геометрия	Нет	Автоматическая
SSDM	Сильная	Геометрия	Нет	Ручная
Воксельныйрейкастинг	Средняя	Геометрия, топология, текстуры	Нет	Автоматическая
Маскируемые ленты	Средняя	Геометрия	Вершинные	Автоматическая

Таблица 2: Сравнение возможностей рассмотренных подходов (продолжение)

## 2. Разработанное решение

В базовом виде алгоритм из статьи [6] делится на два этапа: предобработка и рендеринг. Алгоритм предобработки принимает на вход дискретное многообразие (возможно с краем), хранимое в формате “треугольного супа” – неиндексированного набора треугольников, где каждый треугольник задан атрибутами его вершин. Этот формат был выбран в целях упрощения имплементации предобработки во внешней памяти.

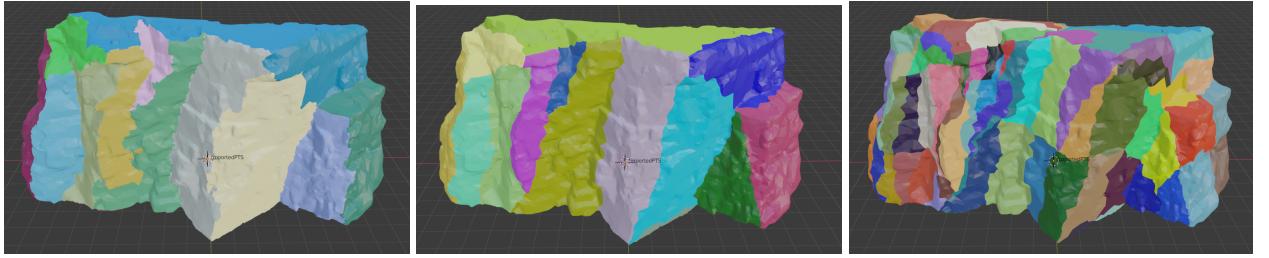


Рис. 1: Визуализация этапа предобработки (слева направо: кластеризация, распрямление границ, квадрангуляция)

## 2.1. Предобработка

Цель этапа предобработки – построить непрерывный атлас меша, а затем ресемплировать атрибуты в каждой карте атласа по равномерной сетке в параметрическом пространстве, тем самым получив семейство геометрических изображений. Предлагаемый алгоритм состоит из следующих этапов:

1. Раскладывание треугольников по бакетам
2. Кластеризация треугольников в рамках бакетов
3. Глобальная кластеризация
4. Распрямление границ кластеров
5. Квадрангуляция
6. Репараметризация
7. Ресемплинг

### 2.1.1. Раскладывание треугольников по бакетам

Этот и последующий этапы фактически являются оптимизациями основного алгоритма, необходимыми для работы с большими мешами. Пространство модели разбивается по прямоугольной равномерной трёхмерной сетке на “бакеты”, треугольники каждого бакета проходят следующий этап независимо, что позволяет запускать его параллельно на многоядерных процессорах.

### 2.1.2. Кластеризация

Цель кластеризации – разбить все треугольники на непересекающиеся связные множества (называемые кластерами), оптимизируя некоторые эвристики и сохраняя некоторые инварианты. В данной работе используются эвристики планарности, компактности и изменения иррегулярности. Новые кластеры строятся последовательным объединением имеющихся кластеров минимизируя целевые эвристики.

Пусть  $M$  – кластер, состоящий из вершин  $\{v_1, \dots, v_k\}$ . Для удобства будем считать, что вершины заданы в однородных координатах. Плоскость будем задавать однородным вектором  $n$ , задающим множество точек плоскости как решения  $n^\top v = 0$ . Рассмотрим среднеквадратичное отклонение вершин кластера от произвольной поверхности:

$$E_M(n) = \frac{1}{k} \sum_{i=1}^k (n^\top v_i)^2.$$

Обозначим точку минимума этой функции, то есть прямую приближающую кластер методом наименьших квадратов, за  $n_0$ . Тогда эвристикой планарности кластера назовём

$$E_{fit}(M) = E_M(n_0).$$

Конечно же вычисление этой эвристики заново на каждой итерации алгоритма приведёт к квадратичной асимптотике, что неудовлетворительно. Для эффективного вычисления в [16] предлагается использовать поверхности второго порядка. Раскрывая скобки в определении  $E_M$ ,

$$E_M(n) = \frac{1}{k} \sum_{i=1}^k n^\top (v_i v_i^\top) n = \frac{1}{k} n^\top \left( \sum_{i=1}^k v_i v_i^\top \right) n = \frac{1}{k} n^\top Q_M n,$$

где  $Q_M = \sum_{i=1}^k v_i v_i^\top$ . Так как  $Q_{M \cup N} = Q_M + Q_N$ , для вычисления  $E_M$  достаточно хранить для каждого кластера его форму  $Q_M$ , а при объединении кластеров складывать их.

Саму МНК плоскость же можно найти используя выборочную ковариационную матрицу

$$Z = \frac{1}{1-k} \sum_{i=1}^k (v_i - \bar{v})(v_i - \bar{v})^\top.$$

Первые три координаты  $n_0$  будут равны первым трём координатам собственного вектора  $Z$  соответствующего наименьшему собственному значению, а последняя координата может быть вычислена из равенства  $n_0^\top \bar{v} = 0$ . К счастью, матрица  $Z$  также может быть вычислена через форму  $Q_M$ :

$$Z = \sum_{i=1}^k v_i v_i^\top - k(\bar{v} \bar{v}^\top) = Q_M - \frac{Q_{M1:3,4} Q_{M1:3,4}^\top}{Q_{M4,4}},$$

где константа пропорциональности опущена так как не влияет на направления собственных векторов и положения в вариационном ряду собственных значений. Это наблюдение и позволяет вычислять ошибку планарности на каждой итерации алгоритма за  $O(1)$ .

Пусть площадь и периметр кластера  $M$  равны  $A$  и  $P$  соответственно. Тогда иррегулярность кластера называют величину

$$\gamma(M) = \frac{P^2}{4\pi A}.$$

Иррегулярность измеряет степень схожести кластера с диском. Эвристикой изменения иррегулярности для объединения кластеров  $N_1, N_2$  в кластер  $M$  называют

$$E_{shape} = \frac{\gamma(M) - \max(\gamma(N_1), \gamma(N_2))}{\gamma(M)}.$$

Эвристикой компактности же назовём

$$E_{comp}(M) = P^2.$$

Эвристики изменения иррегулярности и планарности взяты из работы [16], но попытка использовать подход описанные в этой работе как есть не увенчалась успехом: эвристика ориентации не вносила значительного вклада в результат на тестовых моделях, а эвристики изменения иррегулярности оказалось недостаточно чтобы передать кластерам окружную форму (отношение квадрата периметра к площади кластера могло оставаться близким к  $4\pi$  даже когда кластер имеет крайне вытянутую форму за счёт “ребристости” поверхности, так как ребристость увеличивает площадь не меняя периметра). Было принято решение отказаться от эвристики ориентации, а эвристику изменения иррегулярности заменить на некоторую другую метрику. Из вариантов были рассмотрены просто иррегулярность, квадрат периметра и сумма квадрата периметра и изменения иррегулярности. Последний вариант дал наиболее благоприятный результат: сам по себе квадрат периметра приводил к слишком ломанным границам между кластерами, а учёт изменения иррегулярности позволил сгладить этот эффект. Однако квадрат периметра не совпадает по размерности с остальными эвристиками, что потребовало подбора коэффициента вклада этой эвристики под конкретную модель. На использованных тестовых моделях (ландшафты [17]) наилучший результат дал коэффициент  $10^{-4}$  для компактности, 1.7 для планарности и 1 для изменения иррегулярности.

В процессе кластеризации на кластеры накладывается следующий топологический инвариант: пересечение любых двух кластеров гомеоморфно либо точке, либо отрезку, и при этом граница любого кластера гомеоморфна окружности. Этот инвариант позволит на этапе квадрангуляции получить разбиение, удовлетворяющее определению непрерывного атласа.

Используется жадный алгоритм кластеризации, строящий кластеры снизу-вверх. Поддерживается очередь с приоритетами из пар кластеров – претендентов на слияние. Приоритетом является сумма эвристик посчитанных для объединения кластеров. В ходе итерации алгоритм берёт потенциальный мердж с минимальной ошибкой из очереди, проверяет не нарушит ли он топологические инварианты, объединяет кластеры с использованием структуры данных “система непересекающихся множеств”, затем обновляет приоритеты потенциальных мерджей кластера этой итерации и его соседей. Алгоритм прекращает свою работу при достижении целевого количества кластеров или превышении дозволенной ошибки.

### 2.1.3. Глобальная и локальная кластеризация

Приведённый алгоритм кластеризации используется и на 3 и на 4 этапах, но в ходе 3 этапа каждый треугольник в бакете считается отдельным кластером, а в ходе 4 этапа изначальным набором кластеров берётся объединение результатов работы предыдущего этапа. Очевидно, на входе 3 этапа топологический инвариант выполнен. Каждая итерация алгоритма сохраняет инвариант, поэтому в рамках каждого бакета он тоже выполнен. Однако несложно построить пример ситуации когда инвариант нарушается при объединении бакетов. На рисунке

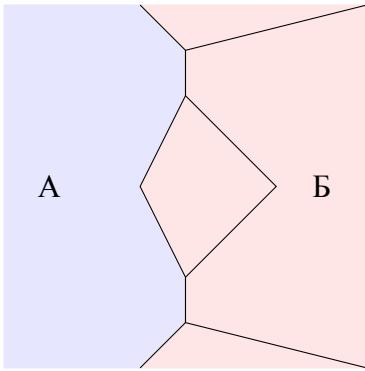


Рис. 2: Нарушение инварианта при наивном объединении бакетов

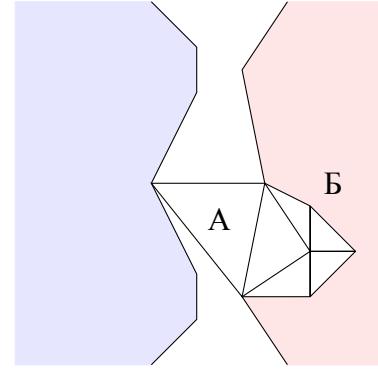


Рис. 3: Нарушение инварианта при объединении бакетов с разграничителем

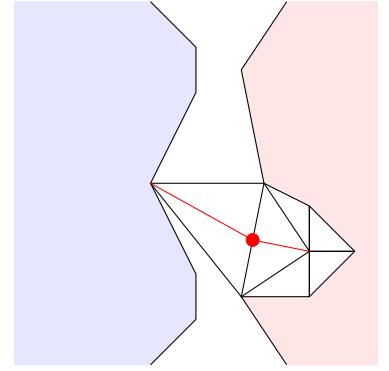


Рис. 4: Устранение нарушений при объединении бакетов с разграничителями

[2](#) синим и красным цветом обозначены кластеры двух разных бакетов соответственно. Кластеры А и Б пересекаются по несвязному множеству, тем самым нарушая инвариант, хотя в рамках каждого бакета инвариант выполнен. В работе [6] никак не уточняется этот момент, однако практика показывает что если бакеты достаточно велики относительно детализации меша, то эта проблема возникает достаточно редко. С целью устранения этой проблемы было решено не включать в бакет треугольники пересекающие его границы, а добавлять их как отдельные кластеры при переходе к 4 этапу. Этот подход позволяет частично избежать ситуации с рисунка [2](#), но инвариант всё ещё может быть нарушен как на рисунке [3](#): кластер границы А и кластер правого бакета Б пересеклись по несвязному множеству, двум точкам. Если в дополнение к этому подходу разделять треугольники нарушившие инвариант на 2 части, либо использовать граничные треугольники в проверке инварианта в каждом бакете, то проблема окончательно пропадает. В данной работе был выбран следующий подход. Рассмотрим множество треугольников пересекших границы бакетов (то есть таких, что не все вершины попали в один бакет. Это условие необходимо и достаточно, так как и треугольник и бакет – выпуклые множества). Ясно, что топологически эта фигура является двумерной поверхностью с  $n$  дырками. Обозначим за  $A_i$  множества вершин, образующие границы дырок. Рассмотрим рёбра такие, что их вершины принадлежат множеству  $A_i$ , но на ребро опираются 2 треугольника (т.е. ребро не лежит на границе дырки). Разделим все такие рёбра и смежные с ними треугольники пополам (рисунок [4](#)). Легко понять, что после этой операции любой треугольник пересекается со всей поверхностью кластера либо ровно по одной вершине, либо по ровно по одному ребру, чего и достаточно для соблюдения инварианта при объединении всех бакетов и граничного множества.

Ещё одним тонким моментом в процессе кластеризации является работа с границей многообразия. Авторы [18] не уточняют в своих определениях считается ли атлас, одна из карт которого имеет область определения пересекающуюся с границей меша по двум несвязным отрезкам, корректным (рисунок [5](#)). В данной работе считается, что вся граница многообразия является границей одной карты, что запрещает ситуации как на рисунке.

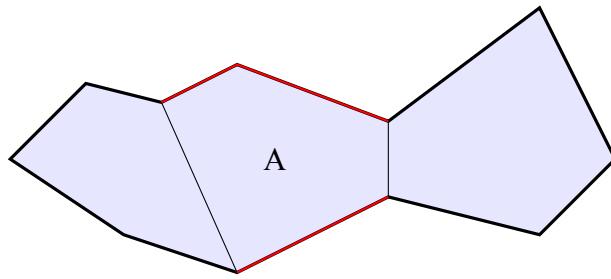


Рис. 5: Атлас с картой А пересекающей границу по несвязному множеству (отмечено красным)

#### 2.1.4. Распрямление границ

При использовании некоторых эвристик границы кластеров получаются достаточно ломанными, что приводит к артефактам на их границах при рендеринге. С целью борьбы с этой проблемой на этом этапе пары граничащих кластеров загружаются в память и граница между ними заменяется на кратчайший путь из конца в начало найденный алгоритмом Дейкстры [19]. В этот этап тоже достаточно легко вносится параллелизм. Так как операция распрямления границы мутирует пару файлов-кластеров, необходимо убедиться что никакие два исполнителя не будут параллельно распрямлять границы принадлежащие одному кластеру.

#### 2.1.5. Квадрангуляция

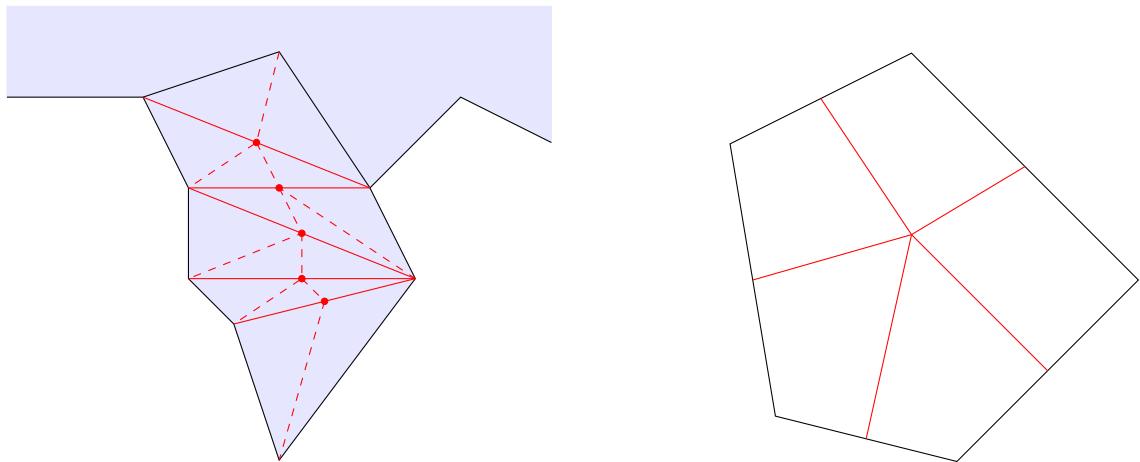


Рис. 6: Разделение рёбер для построения путей при квадранголяции. Сплошные красные – рёбра, подлежащие разделению, красные точки – новые вершины, красный пунктир – дополнительные рёбра

Рис. 7: Квадрангуляция. Чёрные рёбра – схематичное представление исходного кластера как многоугольника, красным – пути разрезания кластера на четырёхугольники

Легко увидеть, что в результате кластеризации каждый кластер геометрически является  $n$ -угольником. Вершинами назовём точки, в которых пересекаются более чем 2 кластера, считая воображаемую область за границей меша отдельным кластером. Рёбрами назовём

множества из тех точек, в которых пересекаются ровно 2 кластера. В ходе этого этапа в каждом кластере выделяется вершина называемая центром, а затем проводятся кривые из центра к серединам рёбер. Кривые строятся алгоритмом  $A^*$  [20] так, чтобы кривые шли как можно дальше друг от друга, от границы всего кластера, и не пересекались. Для достижения этого эффекта веса рёбер взвешиваются единицей делённой на расстоянием до нежелаемого множества. Порядок выбора рёбер для проведения кривых следующий: первая кривая произвольная, а затем в самом большом по количеству рёбер регионе из получившегося разбиения выбирается центральное ребро. Этот подход позволяет итоговым четырёхугольникам быть примерно одинакового масштаба и пропорций. В случае если это построение не возможно, предлагается разделять мешающие треугольники. Авторы не уточняют используемый алгоритм поиска мешающих треугольников, а также способ разделения, поэтому в данной работе было решено использовать следующий подход. На каждой итерации проведения кривой в регионе, ищутся все рёбра такие, что само ребро не лежит на границе, но при этом обе его вершины лежат на границе. Далее эти рёбра делятся пополам, как и опирающиеся на них треугольники. На рисунке 6 красным цветом обозначены описанные рёбра, а пунктиром обозначено дополнительное построение получающееся в результате последовательного разделения красных рёбер. После этого разделения пути гарантированно будут существовать.

Пусть дан граф триангуляции диска, вершины  $O, M$  лежат на границе, и любой путь между ними проходит ещё хотя бы один раз по границе. Покажем, что тогда найдётся ребро не лежащее на границе, но обе вершины которого лежат на ней (назовём такие рёбра плохими). Тогда по контрапозиции получим корректность алгоритма. Доказывать будем по индукции. Базой будут такие графы, в которых нет смежных внутренних вершин. Если же в графе есть смежные внутренние вершины, стянем ребро между ними. Каждому пути в старом графе можно сопоставить путь в новом графе, при этом не потеряв посещения границы на них. Значит по предположению индукции в графе есть плохое ребро. Но операция стягивания ребра не могла добавить новых плохих рёбер. Значит плохое ребро было и в старом графе. Пусть в графе нет смежных внутренних вершин. Если в графе любая граничная вершина имеет степень не больше трёх, очевидно граф выглядит как  $n$ -угольник и в нём есть пути из любой вершины границы в любую другую не проходящие по границе, что противоречит условию. Значит найдётся вершина на границе со степенью хотя бы 4. Рассмотрим её соседей в порядке обхода. Заметим, что 2 подряд идущих соседа не могут быть внутренними вершинами. Значит кроме первой и последней вершины в порядке обхода мы найдём ещё одну вершину лежащую на границе. Эта вершина и изначальная и будут образовывать целевой плохое ребро. Таким образом в любом графе с описанными свойствами найдётся плохое ребро, что и требовалось доказать.

Из рисунка 7 видно, что в ходе этой операции многогранник будет разбит на четырёхугольники. Более того, легко увидеть, что полученное разделение будет удовлетворять определению непрерывного атласа. Пары четырёхугольников полученных из одного кластера пересекаются по построению либо по общей вершине, либо по общему ребру. Сами кластеры из поддерживаемого инварианта пересекаются аналогичным образом. А так как новые вер-

шины ставятся на серединах рёбер кластеров одинаковым образом, из чего четырёхугольники разных кластеров тоже пересекаются либо по общему ребру, либо по общей вершине.

Стратегия выбора центра предлагаемая [18] достаточно произвольна. Выбирается одно из рёбер многогранника, из его центра проводятся кривые в центры других рёбер, а затем центры проведённых кривых в порядке обхода принимаются за новый многогранник, у которого на 1 ребро меньше, после чего процедура запускается рекурсивно пока не останется вырожденного многогранника из меньше чем трёх точек, центр которого берётся за центр всего кластера.

Отметим, что в случае если в кластере все вершины лежат на границе, алгоритм не будет работать корректно, так как не удастся найти центр. Эту проблему не сложно решить добавив новую произвольную вершину, но в данной работе не возникло в этом нужды так как столь малые кластеры не удовлетворительны для дальнейшей работы метода, и алгоритм кластеризации не оставляет таких кластеров при должном критерии остановки.

### 2.1.6. Репараметризация

После этапа квадрангulationи меш разделён на области определения карт для итогового атласа. Далее необходимо построить согласованные гомеоморфизмы для каждого четырёхугольника на единичный квадрат, то есть построить параметризацию. Задача построения параметризации широко известна в области компьютерной графики, и к её решению было придумано много подходов. Большая часть из них берут за исходную параметризацию вложение Татта [21], а затем оптимизируют некоторый целевой функционал, не нарушающий инъективности исходной параметризации. В данном алгоритме важно, при ресемплинге частота дискретизации была пропорциональна кривизне исходной поверхности в каждой точке, поэтому был выбран подход из статьи [22]. Для корректности получившегося атласа в процессе построения параметризации вершины, соответствующие углам четырёхугольника, были закреплены на углах единичного квадрата, а вершины лежащие на границе были распределены по границам квадрата пропорционально длине рёбер, и также зафиксированы.

### 2.1.7. Ресемплинг

Цель ресемплинга – построить геометрические изображения содержащие информацию об атрибутах меша в его точках, соответствующих точкам на равномерной прямоугольной сетке в пространстве параметризации. Заметим, что при таком построении для двух соседних карт их пересечение будет семплировано в оба геометрических изображения соответствующих этим картам. Таким образом если спроецировать области геометрических текстур на исходную модель, соседние изображения будут иметь пересечение в 0.5 пикселя. Для удобства построения квадродеревьев в следующем разделе в качестве частоты ширины и высоты геометрических изображений берутся исключительно числа вида  $2^n + 1$ .

Также для адаптивного рендеринга необходимо сгенерировать mip-уровни для итоговых геометрических текстур. Авторы статьи [18] предлагают использовать фильтрацию с

достаточно сложной обработкой граничных случаев, однако так как каждый следующий мип-уровень в 4 раза меньше предыдущего, делать ресемплинг для каждого уровня заново занимает всего в  $\approx 1.3$  раза дольше ресемплинга самого высокочастотного мип-уровня, поэтому в данной работе был выбран именно этот подход.

Небольшой проблемой упомянутой в [6] стали правила растеризации. При семплировании по описанной выше схеме граничные семплы приходятся ровно на границу патча. В такой ситуации растеризатор не считает что семплы нижней и правой грани попали внутрь патча, поэтому не генерирует фрагментов для них. Эта проблема легко устраняетсярендерингом этих граней отдельно от основных треугольников. Однако метод предложенный в [6] преобразует эти грани не так, как основную часть модели, что может привести к неточностям и артефактам. При использовании такого же преобразования возникает проблема с правым нижнем пиксели: согласно правилам растеризации он вновь не будет закрашен. Но так как вся информация об атрибутах в этой точке уже известна из изначальной модели, в данной работе мы заполняем это недостающее значение уже после основного ресемплинга.

## 2.2. Рендеринг

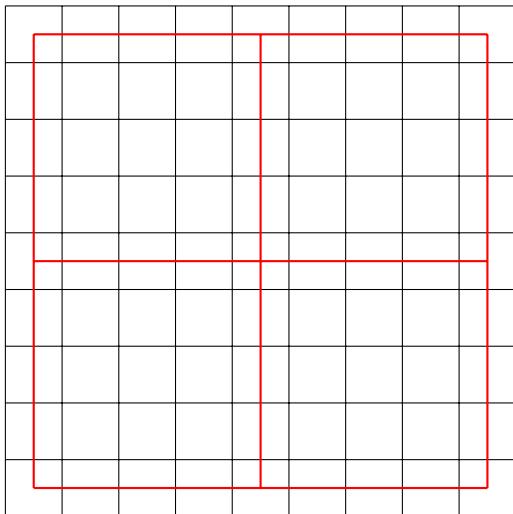


Рис. 8: Расположение вершин квадродерева в параметрическом пространстве.

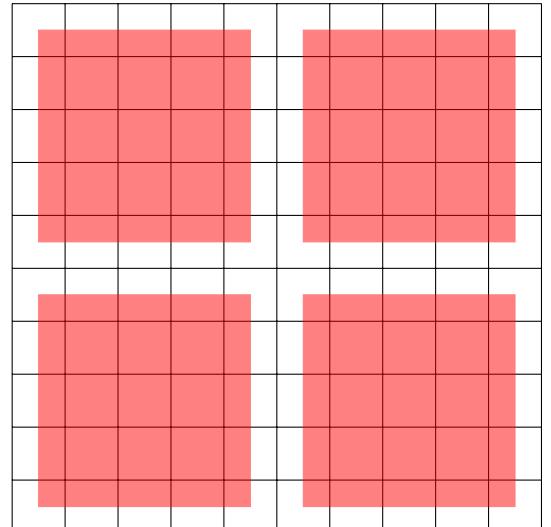


Рис. 9: Области семплирования страниц в кэше

Как было упомянуто выше, для мелкогранулярного выбора уровня детализации по модели данный алгоритм использует квадродеревья в рамках каждой карты. На каждом уровне дерева пространство параметризации делится на 4 равных квадрата. Из этого следует что граница разделения проходит по середине центрального пикселя геометрического изображения, соответственно новым вершинам дерева соответствуют регионы геометрического изображения пересекающиеся по центральным пикселям и имеют размер  $2^{n-1} + 1$  (рисунок 8).

Для адаптации иерархии был выбран простейший алгоритм из [6] постепенно измельчающий разрез квадродерева в попытке сэкономить полигоны адаптируя части квадродерева по отдельности. Адаптация мип-уровня вершин квадродерева происходит посредством

вычисления ограничивающей призмы, проецированием её в пространство камеры и поиском ограничивающего её прямоугольника. Логарифм корня площади этого прямоугольника в пикселях с учётом некоторой константы пропорциональности берётся за целевую плотность треугольников для данной вершины.

### 2.2.1. Виртуализация

Для виртуализации геометрических изображений был выбран простой алгоритм LRU-кэширования. Размер наименьшего mip-уровня  $K$  берётся за размер страницы кэша, каждое геометрическое изображение разрезается на страницы с наложением в один пиксель (аналогично соответствуя регионов пространства параметризации вершинам квадрограмма), по мере нужды страницы соответствующих изображений загружаются в кэш хранимый на видеокарте (без наложения). Наличие отступа в полпикселя у каждой страницы с каждой стороны позволяет использовать билинейную фильтрацию для кэша без возникновения артефактов на границах. На рисунке 9 красным обозначены сэмплируемые области страниц, интерполяция пикселей из разных страниц в остальной области не влияет на отображаемый результат.

Для вычисления позиции страницы определённого изображения  $I$  на определённом mip-уровне  $M$  в кэше используются таблицы индирекции.  $I, M$ -таблица имеет размер  $2^{M-K}$  по высоте и ширине и в ячейках содержит либо номер ячейки в кэше содержащей соответствующую страницу, либо служебное значение  $-1$ , означающее что страница отсутствует в кэше.

Для стабилизации частоты кадров количество страничек загружаемых за один кадр в кэш ограничено. Из-за этого может возникать нужда в момент рендеринга получить данные из страницы не находящейся в данный момент в кэше. В таком случае вычисляется в какой странице mip-уровня на 1 меньше целевого находится текущая точка поверхности и повторяется процедуру лукапа страницы. Процесс повторяется пока не будет найдена доступная в кэше страница. Чтобы таковая всегда была, изображения наименьшего mip-уровня никогда не извлекаются из кэша. Таким образом не зависимо от состояния кэша всегда есть возможность отобразить модель в каком-то качестве, возможно меньшем чем желаемое.

Важным моментом является факт дублирования граничной информации страниц – при вычислении элемента таблицы лукапа для точки параметрического пространства лежащей на границе между несколькими страницами можно выбрать любую из них. Однако неосторожная обработка этого момента может привести к разрывам. В данной работе всегда выбирается страница целиком лежащая в регионе текущей вершины атласа, а разрывы устраняются в рамках описанного ниже алгоритма.

### 2.2.2. Устранение разрывов

При наивном рендеринге граничащих вершин квадрограммов с разными mip-уровнями геометрических изображений будут образовываться разрывы из-за разной частоты дискре-

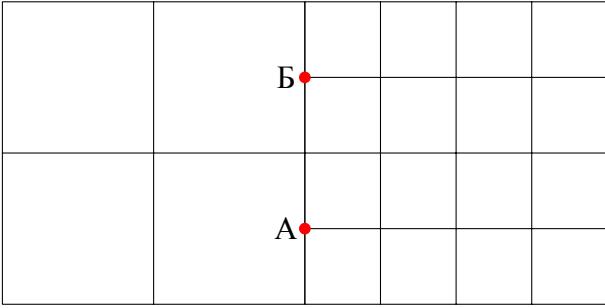


Рис. 10: Потенциальные разрывы

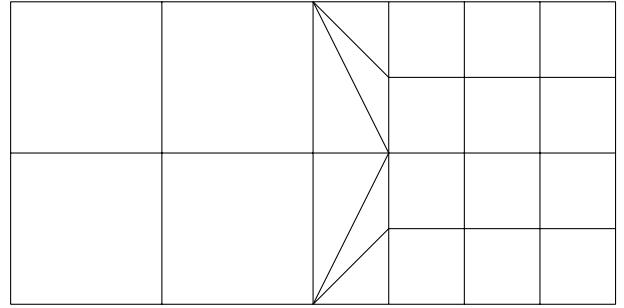


Рис. 11: Устранение разрывов

тизации границы. На рисунке 10 вершины А и Б со стороны правой вершины могут не совпасть по координатам с серединой соответствующих рёбер левой вершины. Для устранения этой проблемы необходимо схлопывать рёбра со стороны вершины с большим мип-уровнем согласно рисунку 11. Однако может образовываться ситуация в которой вершина с высоким мип-уровнем граничит по одному ребру с несколькими вершинами различных более низких мип-уровней. В этой ситуации необходимо взять минимум из всех мип-уровней вершин опирающихся на это ребро и выбрать мип-уровень границы именно таким.

В данной работе используется следующий алгоритм поиска мип-уровней границ. Рассматриваются только вершины из активного среза. При изменении мип-уровня вершины перебрать все её рёбра, для каждого ребра найти всех опирающихся на него соседей и выбрать наибольшую по физическому размеру вершину. Для этой вершины находим всех соседей со стороны граничащей с изначальной вершиной, находим минимум из их мип-уровней и устанавливаем всем им мип-уровень границы на найденный минимум.

Для поиска соседних вершин входящих в срез, как отмечают в [6], необходимо хранить в корнях деревьев 4 ссылки на соседние квадродеревья (их количество таково из свойств областей определений карт), а также ссылки на родителя в каждой вершине. Однако этой информации не достаточно. Как и при работе с обычными многообразиями, для переноса информации между различными картами необходимо иметь *отображение склейки*, позволяющее отобразить параметрические координаты одной карты в параметрические координаты другой для точек лежащих в обоих картах. Для рассматриваемых нами атласов эти отображения всегда являются композицией переноса на целое число вдоль одной из осей и поворота на угол кратный  $\frac{\pi}{2}$ . Матрицы, задающие эти отображения, легко предподсчитываются сопоставлением координат вершин соответствующих углам геометрических изображений.

В данной работе дополнительно предподсчитываются для каждой вершины и каждой её стороны вершина, с которой достаточно начинать поиск соседей. В случае если с целевой стороны находится край текущей карты, этой вершиной является корень соседнего дерева. В ином случае такой вершиной служит первый из родителей текущей в порядке подъёма такой, что все вершины опирающиеся на грань текущей являются его потомками.

Наконец отметим, что при поиске мип-уровней границ необходимо учитывать в каком разрешении доступны страницы геометрического изображения на GPU на данном кадре. Для этого достаточно включать в поиск минимального мип-уровня для грани мип-уровни

страниц, опирающихся на эту грань.

### 2.2.3. Графический конвейер

Для рендеринга иерархического атласа хорошо подходят тесселляционные шейдеры. Каждая модель рендерится при помощи одного вызова отрисовки с использованием инстансинга. Инстансы соответствуют выбранным вершинам квадродеревьев атласа, а количество вершин в каждом инстансе равно 4. Для каждого из них на видеокарту передаются выбранный mip-уровень, mip-уровни для сторон (именно за счёт них происходит описанное выше склонгивание), размер и позиция вершины в параметрическом пространстве, а также номер квадродерева, используя примитив данных “patch list”. В вершинном шейдере по номеру вершины выбираются её координаты как одного из углов единичного квадрата. Далее шейдер управления тесселляцией выставляет режим “quad” и уровни тесселляции в соответствии с mip-уровнями текущего инстанса. После тесселляции оценочный шейдер получает на вход одну из протесселлированных вершин расположенных равномерно (не считая границ) по единичному квадрату и используя их координаты, номер квадродерева и позицию в параметрическом пространстве вычисляет позицию вершины в параметрическом пространстве и обращаясь к соответствующей странице кэша по алгоритму, описанному выше, находит необходимые вершинные атрибуты, выставляет позицию вершины и передаёт остальные атрибуты во фрагментный шейдер для расчёта освещения.

## 3. Результаты

Имплементация описанного выше решения была написана на C++20 с использованием библиотек Eigen, Function2, GLFW3, Cxxopts и Vulkan. Исходный код имеет размер порядка 10000 строк и доступен по ссылке [https://github.com/Mrkol/thesis\\_vgi](https://github.com/Mrkol/thesis_vgi).

В качестве тестовых моделей были выбраны 3D-сканы Quixel Megascans [17] в максимальном качестве. В следующей таблице приведено время работы этапов предобработки, а также количество полигонов и название использованных моделей. В столбец ”подготовка” объединены этапы (1-2), в столбец ”кластеризация” этапы (3-4), в ”квадрангуляцию” – (5-6), а последние 2 столбца соответствуют этапам (7) и (8). В разных строках указаны запуски с разной целевой ошибкой кластеризации, в столбце ”кластеры” указано итоговое количество кластеров. Замеры происходили на Intel Core i5-470 (3.20Ghz) и видеокарте NVIDIA GeForce RTX 2070 SUPER.

Как видно из таблицы, время затраченное на репараметризацию сильно зависит от количества кластеров. Более того, при малом числе кластеров их форма как правило становится достаточно иррегулярной, что приводит к значительно большему времени на этапе квадрангуляции из-за необходимости делить рёбра. С другой стороны большее число кластеров приводит к более медленному рендерингу из-за необходимости адаптировать большее число квадродеревьев.

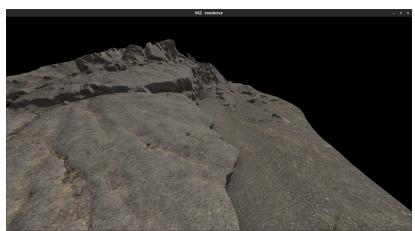


Рис. 12: Оригинальная модель. Вид с ракурса 1

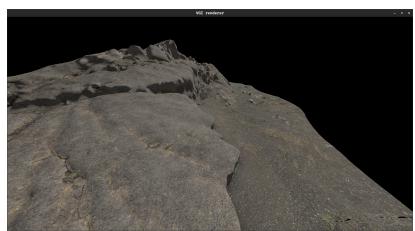


Рис. 13: Предлагаемый метод. Вид с ракурса 1

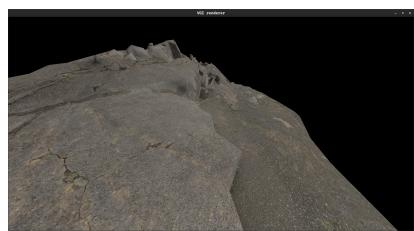


Рис. 14: Уровни детализации. Вид с ракурса 1



Рис. 15: Оригинальная модель. Вид с ракурса 1 (сетка)



Рис. 16: Предлагаемый метод. Вид с ракурса 1 (сетка)



Рис. 17: Уровни детализации. Вид с ракурса 1 (сетка)

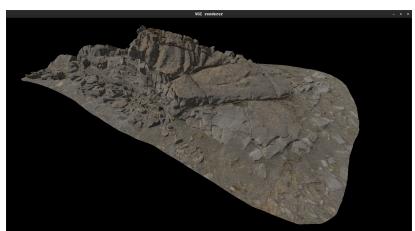


Рис. 18: Оригинальная модель. Вид с ракурса 2

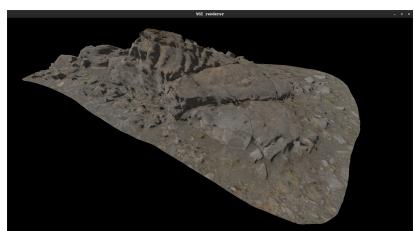


Рис. 19: Предлагаемый метод. Вид с ракурса 2

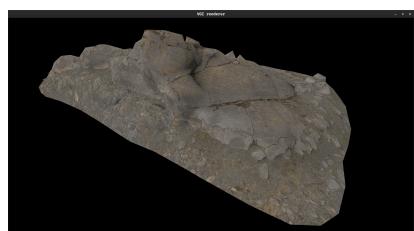


Рис. 20: Уровни детализации. Вид с ракурса 2



Рис. 21: Оригинальная модель. Вид с ракурса 2 (сетка)

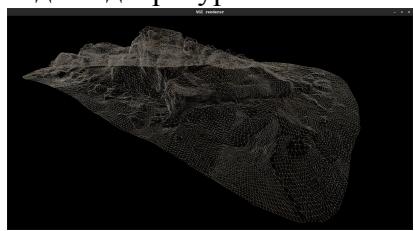


Рис. 22: Предлагаемый метод. Вид с ракурса 2 (сетка)

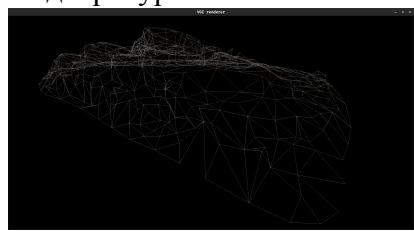


Рис. 23: Уровни детализации. Вид с ракурса 2 (сетка)

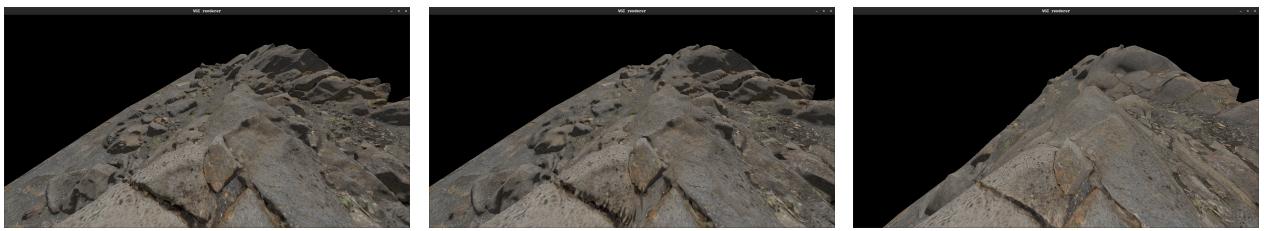


Рис. 24: Оригинальная модель. Вид с ракурса 3

Рис. 25: Предлагаемый метод. Вид с ракурса 3

Рис. 26: Уровни детализации. Вид с ракурса 3



Рис. 27: Оригинальная модель. Вид с ракурса 3 (сетка)

Рис. 28: Предлагаемый метод. Вид с ракурса 3 (сетка)

Рис. 29: Уровни детализации. Вид с ракурса 3 (сетка)

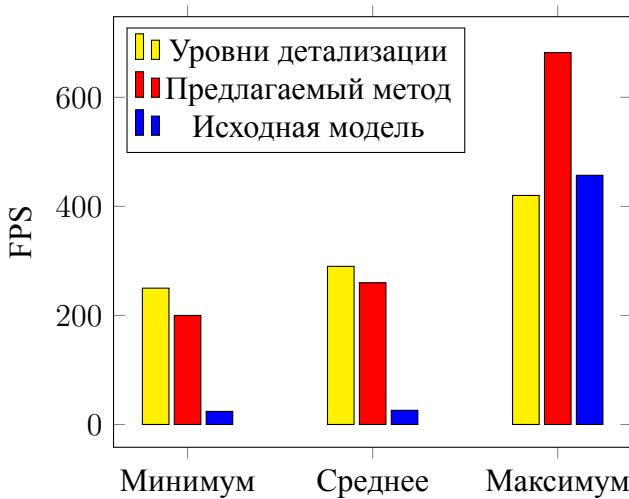


Рис. 30: Производительность рендеринга модели “Rock Cliffs (uchwaffda)” в кадрах в секунду (FPS, frames per second). Больше – лучше

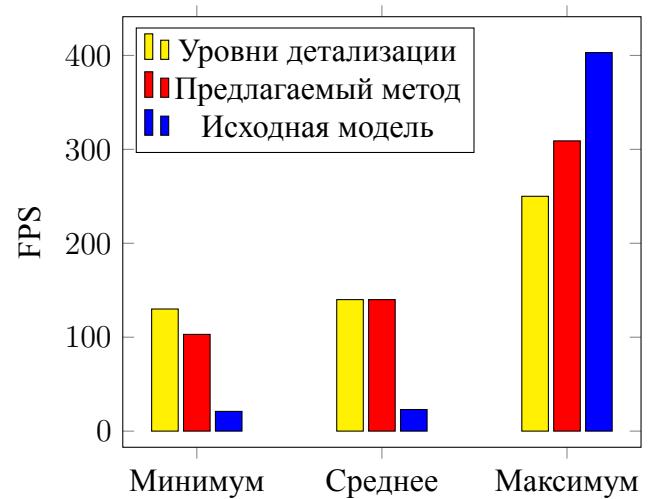


Рис. 31: Производительность рендеринга модели “Rock Assembly Rough (sjzbj)” в кадрах в секунду (FPS, frames per second). Больше – лучше

	Полигоны (млн.)	Кластеры	Подготовка (мин.)	Кластер. (мин.)	Квадранг. (мин.)	Репарам. (мин.)	Ресемплинг (мин.)
Rock Cliffs (uchwaffda)	3 3	130 164	2.1 2.1	0.8 0.8	40 40	105 83	0.2 0.2
Rock Sandstone (rlbx3)	5 5	30 224	3.4 3.5	1.4 1.4	116 34	700 200	0.5 0.3

Таблица 3: Производительность алгоритма предобработки

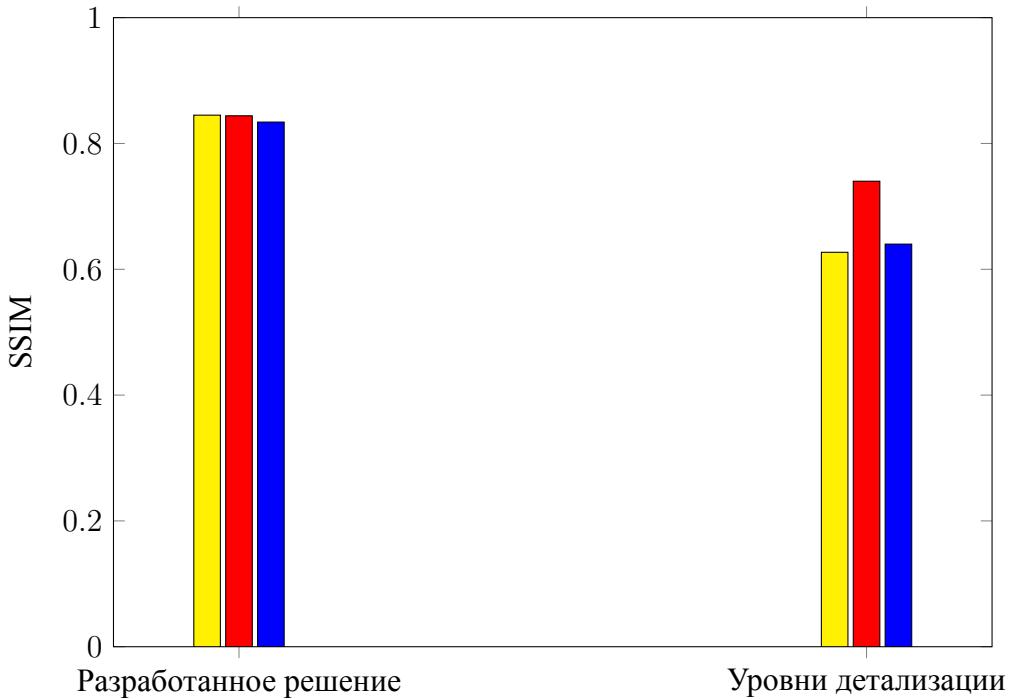


Рис. 32: SSIM-качество отрендеренных изображений с трёх ракурсов (обозначены цветами).  
Больше – лучше

Производительность и качество рендеринга же измерялись на процессоре Intel Core i7-8550U и его встроенной видеокарте. Производительность алгоритма при движении камеры показана на рисунках 30-31. Для сравнения были взяты исходная высокополигональная модель отрендеренная и предоставляемые Quixel её уровни детализации. Из графиков видно, что использование иерархического атласа не сильно ухудшает производительность по сравнению с наивным использованием уровней детализации, но при этом в разы быстрее рендеринга полной исходной модели.

Противопоставим эти показатели итоговому качеству картинки. Для его измерения была использована метрика SSIM [23], разработанная специально для сравнения видимого качества семантически одинаковых изображений. Сравниваемые изображения, а также изображения каркасов соответствующих моделей, представлены на рисунках 12-29. Изображения полученные при помощи разработанного решения, а также полученные с помощью наивных уровней детализации, были сопоставлены с исходной высокополигональной моделью с трёх

разных ракурсов. Результаты представлены на рисунке 32. Таким образом разработанное решение позволяет добиться качества на 20 процентных пунктов лучше наивного подхода, при этом незначительно проигрывая ему в производительности.

## 4. Заключение

В рамках проведённой работы были уточнены детали устройства метода рендеринга с использованием иерархических атласов, предложен ряд усовершенствование в алгоритме отображения, а также разработано комплексное программное решение имплементирующее метод. Разработанное решение позволяет добиться значительного прироста в качестве картинки по сравнению с наиболее распространённым методом, уровнями детализации. Производительность решения удовлетворительна на этапе предобработки и сравнима с наивным подходом на этапе рендеринга.

В силу большой модульности и сложности метода предобработки, при дальнейших исследованиях планируется повышать производительность и качество практически всех его частей. Планируется реализовать некоторое количество альтернативных подходов и сравнить получаемое качество, а также время предобработки. Среди прочего планируются исследования в направлении поддержки неоднородных вершинных атрибутов посредством проведения границ кластеров по границам разрывов, ускорение этапа репараметризации используя иные алгоритмы оптимизации, использование иных методов кластеризации, а также внедрение модификаций предлагаемых в [14]. Планируется пересмотрение этапа расправления границ. В результате этой операции эвристика планарности кластеров сильно ухудшается, что в некоторых ситуациях приводит к недостаточной частоте семплирования выступов моделей. Более тонкий учёт геометрии поверхности на этапе сглаживания может помочь избавится от этой проблемы. Отметим, что аналогичный эффект проявляется и на этапе квадрангуляции, но в меньшей степени.

Ещё одним крупным направлением для дальнейших исследований являются алгоритмы адаптации при рендеринге иерархического атласа. Также планируется внедрение параллелизма в текущую имплементацию рендеринга (вынесение обновления виртуального кэша в отдельный поток), и наконец рассмотрение возможности перенесения вычислений нагрузжающих центральный процессор в текущей имплементации (адаптация, вычисление мип уровней для границ) на графический процессор.

По мнению автора подход виртуализации весьма перспективен в применении к современным приложениям реального времени, так как способен удовлетворить всё растущие требования к визуальному качеству, а также позволяет более эффективно использовать ресурсы GPU, а поэтому заслуживает дальнейшего изучения и усовершенствования.

## Список литературы

- [1] Blinn James F. Simulation of Wrinkled Surfaces // *SIGGRAPH Comput. Graph.* — 1978. — Aug. — Vol. 12, no. 3. — P. 286–292. — Access mode: <https://doi.org/10.1145/965139.507101>.
- [2] Policarpo Fábio, Oliveira Manuel M., Comba João L. D. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces // Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games. — New York, NY, USA : Association for Computing Machinery. — 2005. — I3D '05. — P. 155–162. — Access mode: <https://doi.org/10.1145/1053427.1053453>.
- [3] Szirmay-Kalos László, Umenhoffer Tamás. Displacement Mapping on the GPU — State of the Art // *Computer Graphics Forum*. — 2008. — Vol. 27, no. 6. — P. 1567–1592. — <https://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2007.01108.x>.
- [4] Assarsson Ulf, Moller Tomas. Optimized View Frustum Culling Algorithms for Bounding Boxes // *Journal of Graphics Tools*. — 2000. — Vol. 5, no. 1. — P. 9–22. — <https://doi.org/10.1080/10867651.2000.10487517>.
- [5] Coorg Satyan, Teller Seth. Real-time occlusion culling for models with large occluders // Proceedings of the 1997 symposium on Interactive 3D graphics. — 1997. — P. 83–ff.
- [6] Niski Krzysztof, Purnomo Budirijanto, Cohen Jonathan. Multi-grained level of detail using a hierarchical seamless texture atlas // Proceedings of the 2007 symposium on Interactive 3D graphics and games. — 2007. — P. 153–160.
- [7] Epic Games, Inc. Unreal Engine. — 2020. — Access mode: <https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5>.
- [8] Karis Brian. Virtual Geometry Images. — 2009. — Access mode: <http://graphicrants.blogspot.com/2009/01/virtual-geometry-images.html> (online; accessed: 2020-12-13).
- [9] Lee Aaron, Moreton Henry, Hoppe Hugues. Displaced Subdivision Surfaces // Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. — USA : ACM Press/Addison-Wesley Publishing Co. — 2000. — SIGGRAPH '00. — P. 85–94. — Access mode: <https://doi.org/10.1145/344779.344829>.
- [10] Karis Brian. More Geometry. — 2009. — Access mode: <http://graphicrants.blogspot.com/2009/01/more-geometry.html> (online; accessed: 2020-12-13).
- [11] Bunnell Michael. Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping // GPU Gems 2 / ed. by Pharr Matt. — Addison-Wesley, 2005. — P. 109–122.
- [12] Olick Jon. Current and Next Generation Parallelism in Games. — 2008. — SIGGRAPH.

- [13] Rendering continuous level-of-detail meshes by Masking Strips / Rípolles Oscar, Chover Miguel, Gumbau Jesus, Ramos Francisco, and Puig-Centelles Anna // *Graphical Models*. — 2009. — Vol. 71, no. 5. — P. 184 – 195. — Access mode: <http://www.sciencedirect.com/science/article/pii/S152407030900023X>.
- [14] Feature-preserving triangular geometry images for level-of-detail representation of static and skinned meshes / Feng Wei-Wen, Kim Byung-Uck, Yu Yizhou, Peng Liang, and Hart John // *ACM Transactions on Graphics (TOG)*. — 2010. — Vol. 29, no. 2. — P. 1–13.
- [15] Epic Games, Inc. Nanite Technology. — 2020. — Access mode: <https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5>.
- [16] Garland Michael, Willmott Andrew, Heckbert Paul S. *Hierarchical Face Clustering on Polyg-onal Surfaces* // Proceedings of the 2001 Symposium on Interactive 3D Graphics. — New York, NY, USA : Association for Computing Machinery. — 2001. — I3D '01. — P. 49–58. — Access mode: <https://doi.org/10.1145/364338.364345>.
- [17] Epic Games, Inc. Quixel Megascans. — Access mode: <https://quixel.com/megascans> (online; accessed: 2021-05-27).
- [18] Purnomo Budirijanto, Cohen Jonathan, Kumar Subodh. *Seamless Texture Atlases*. — 2004. — 01. — Vol. 71. — P. 67–76.
- [19] Dijkstra E. W. A Note on Two Problems in Connexion with Graphs // *Numer. Math.* — 1959. — Dec. — Vol. 1, no. 1. — P. 269–271. — Access mode: <https://doi.org/10.1007/BF01386390>.
- [20] Hart Peter E., Nilsson Nils J., Raphael Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths // *IEEE Transactions on Systems Science and Cybernetics*. — 1968. — Vol. 4, no. 2. — P. 100–107.
- [21] Tutte W. T. How to Draw a Graph // *Proceedings of the London Mathematical Society*. — 1963. — Vol. s3-13, no. 1. — P. 743–767. — <https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s3-13.1.743>.
- [22] Texture Mapping Progressive Meshes / Sander Pedro V., Snyder John, Gortler Steven J., and Hoppe Hugues // Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. — New York, NY, USA : Association for Computing Machinery. — 2001. — SIGGRAPH '01. — P. 409–416. — Access mode: <https://doi.org/10.1145/383259.383307>.
- [23] Image quality assessment: from error visibility to structural similarity / Wang Zhou, Bovik Alan C, Sheikh Hamid R, and Simoncelli Eero P // *IEEE transactions on image processing*. — 2004. — Vol. 13, no. 4. — P. 600–612.