Имплементация виртуализации геометрии в рендеринге реального времени

Санду Р.А.[1]

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

31 мая 2021 г.

Содержание

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: максимальное число треугольников в рамках стандартного для интерактивных приложений бюджета в 16 миллисекунд на кадр (60FPS) остаётся порядка десятка миллионов. С другой стороны современные программы для скульптурирования, а также различные технологии 3D сканирования, позволяют получить модели количество полигонов которых почти не ограничено. Бюджет приложения можно насытить 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 “лодов” (от англ. LOD, level of detail, уровень детализации) совместно с различными “мапами” (от англ. map, карта). Лод – результат геометрического упрощения исходной модели содержащий меньше полигонов. Как правило генерируется несколько различных лодов для модели с числом треугольников отличающимся на порядок, а затем при рендеринге динамически выбирается одна из них в зависимости от расстояния до наблюдателя. Различные мапы используются чтобы восстановить визуальное качество модели потерянное за счёт упрощения. Среди часто используемых мап – 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 лодирование 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 адаптивного рендеринга*. Ещё одной проблемой является потребление видеопамяти. Наивно загружать все модели сцены со всеми их лодами в видеопамять быстро израсходует её. Существует множество техник решающих эту проблему, однако их освящение выходит за рамки данной работы.

Виртуализация геометрии – один из подходов к решению задачи адаптивного 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 [4]. Причиной этому послужил анонс “Unreal Engine 5” [5] летом 2020 года. Было вы-

яснено, что реализованная в нём технология микрополигонального рендеринга “Nanite” тоже берёт своё начало в упомянутой выше статье. Целью данной работы положим современную эффективную имплементацию идей этой статьи, а также исследование деталей реализации опущенных в оригинальной статье.

1.1. Определения

- Мешем из n вершин называют пару $M = (I, A)$, числа $1, ..., n$ называют вершинами, $I \subset \{1, ..., n\}^3$ – индексное множество задающее треугольники (тройки вершин), а $A : \{1, ..., n\} \to \mathbb{R}^3 \times \mathbb{R}^n$ – 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). (ССЫЛКА!!!)

- 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 аттрибутов вершин и элементов матрицы и взятием множества треугольников соответствующе рисунку. (РИСУНОК!!!)

1.2. Обзор литературы

В своём блоге [6], [7] Брайан Карис, один из разработчиков технологии “Nanite”, сравнивает различные способы добиться рендеринга бóльшего количества геометрии. Основываясь на этих статьях сравним упомянутые Карисом техники с выбранным нами алгоритмом из статьи Ниски, Пурномо и Коэна [4].

Subdivision surfaces with displacement maps (далее SSDM) – 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 (дальше – 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 ограничится 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 добится 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 начинается 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, а также к измельчению каркаса приводят некоторые техники анимации. В своей статье Карис 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 ”красивость” 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 [8].

Другой способ упомянутый Карисом – воксельный рейкастинг. Эта 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, хотя появившаяся недавно аппаратная поддержка ускорения рейтрейсинга может позволить добиться гораздо лучше результата с использованием этой техники нежели 10 лет назад. Также Карис отмечает требовательность техники к видеопамяти и необходимость дополнительных надстроек для её экономии. Наконец отметим главный недостаток этой техники – она абсолютно не совместима с 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. Для дальнейшей информации смотреть [9].

Ещё один подход к динамическому уровню детализации представлен в [10]. Модель разбивается на набор полосок из треугольников и на этапе препроцессинга генерируется стратегия 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 высокими LOD’ами. 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.

Сравнивая алгоритм представленный в [4] с SSDM и воксельным рейкастингом, использование иерархического атласа фактически позволяет избавится от всех упомянутых проблем. Путём небольшой модификации алгоритма описанного в [11] легко достигается поддержка скелетных анимаций, поддержка морфов тривиальна. Используя эту модификацию 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 [11] 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 память растёт квадратично а не кубически в отличии от воксельного рейкастинга. 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 гово-

ря 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 алиазинга 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 ([12]) его команде пришлось написать свой программный растерайзер.

	Гранулярность по скринспейсу	Скелетные анимации	Упрощение на высоких LOD	Связность
Статические LOD-модели	Нет	Да	Не ограничено	Явная
Иерархический атлас	Сильная	Да	Сильное	Неявная
SSDM	Средняя	Да	Слабое	Явная
Воксельный рейкастинг	Сильная	Нет	Не ограничено	Неявная
Маскируемые ленты	Нет	Да	?	Неявная

	Гранулярность по дистанции	Динамический LOD	Неоднородные атрибуты
Иерархический атлас	Сильная	Текстуры, геометрия	Нет
SSDM	Сильная	Геометрия	Нет
Воксельный рейкастинг	Средняя	Геометрия, топология, текстуры	Нет
Маскируемые ленты	Средняя	Геометрия	Вершинные

	Предобработка
Иерархический атлас	Автоматическая
SSDM	Ручная
Воксельный рейкастинг	Автоматическая
Маскируемые ленты	Автоматическая

2. Разработанное решение

В базовом виде алгоритм из статьи [4] делится на два этапа: предобработка и рендеринг. Алгоритм работает с дискретными многообразиями с краем, с дополнительным условием ”непрерывности” атрибутов в зависимости от точки поверхности. Достаточно сложно строго сформулировать это условие в терминах мешей, так как меш является дискретным объектом, а его интерполяция по определению является непрерывной. (ВСТАВИТЬ КАРТИНКУ)

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. Преобразование исходной модели в ”треугольный суп”
2. Раскладывание треугольников по бакетам
3. Кластеризация треугольников в рамках бакетов
4. Глобальная кластеризация
5. Распрямление границ кластеров
6. Квадрангуляция
7. Репараметризация
8. Ресемплинг

2.1.1. Преобразование в ”треугольный суп”

Популярный в области компьютерной графики термин “triangle soup” подразумевает хранение меша без явной индексации вершин, то есть атрибуты вершины дублируются в рамках описания каждого опирающегося на неё треугольника. Этот формат хранения упрощает работу с подмножествами треугольников меша.

2.1.2. 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.

2.1.3. 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. Изначальная попытка использовать эвристики из статьи [13] не увенчалась успехом: эвристика ориентации не вносила значительного вклада в результат на тестовых моделях, а эвристики изменения иррегулярности оказалось недостаточно чтобы предать кластерам округлую форму (отношение квадрата периметра к площади кластера могло оставаться близким к 4π 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 эвристику компактности. 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 ломаным 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 этой эвристики под конкретную модель. На использованных тестовых моделях (ландшафты [14]) наилучший результат дал коэффициент 10^{-4} для компактности, 1.7 для планарности и 1 для изменения иррегулярности.

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.

2.1.4. 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 в котором в рамках каждого бакета инвариант выполнен, но при этом между бакетами он нарушается. (КАРТИНКА!!!) В статье [4] 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 (РИСУНОК), но инвариант всё ещё может быть нарушен (ДРУГОЙ РИСУНОК). 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 пересёкших 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 поверхность 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. 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 работаа с границей многообразия. Авторы [15] не уточняют в своих определениях считается ли атлас, одна из карт которого имеет область определения пересекающуюся с границей меша по двум несвязным отрезкам, корректным. (КАРТИНКА) В данной работе считается, что вся граница многообразия является границей одной карты, что запрещает ситуации как на рисунке.

2.1.5. 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.

2.1.6. Квадрангуляция

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 Дейкстры с потенциалами (ЦИТИРОВАТЬ) так, чтобы кривые шли как можно дальше друг от друга, от границы всего кластера, и не пересекались. 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 (КАРТИНКА). (ДОКАЗАТЕЛЬСВО, ЧТО ПУТЬ БУДЕТ СУЩЕСТВОВАТЬ)

Из рисунка видно, что в ходе этой операции многогранник будет разбит на четырёхугольники. Более того, (ДОК-ВО ЧТО ИЗ ИНВАРИАНТА ПОЛУЧИМ ХОРОШИЙ АТЛАС).

Стратегия выбора центра предлагаемая [15] 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.

2.1.7. Репараметризация

После этапа квадрангуляции 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 [16], 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 [17]. 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.

2.1.8. 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 мип-уровни для итоговых геометрических текстур. Авторы статьи [15] 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 ≈ 1.3 раза дольше ресемплинга самого высокочастотного мип-уровня, поэтому в данной работе был выбран именно этот подход.

2.2. Рендеринг

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 $2^{n-1}+1$. (КАРТИНКА)

Для адаптации иерархии был выбран простейший алгоритм из [4] 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 мип-уровня K берётся за размер страницы кэша, каждое геометрическое изображение разрезается на страницы с наложением в один пиксель (аналогично соответствию регионов пространства параметризации вершинам квадродерева), 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. (КАРТИНКА)

Для вычисления позиции страницы определённого изображения I на определённом мип-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 мип-уровня на 1 меньше целевого находится текущая точка поверхности и повторяем процедуру лукапа страницы. Процесс повторяется пока не будет найдена доступная в кэше страница. Чтобы таковая всегда была, изображения наименьшего мип-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 двумя страницами можно выбрать любую из страниц, однако может оказаться так, что одна из них доступна только в меньшем разрешении. Неосторожная обработка этого случая может привести к разрывам на границах патчей даже в случае корректной их обработки с точки зрения мип-уровней. Решение проблемы – при выборе страницы находящейся на границе патча всегда выбирать страницу находящуюся целиком внутри патча. (ПРОВЕРИТЬ, ПОЯСНИТЬ)

2.2.2. Устранение разрывов

При наивном рендеринге граничащих вершин квадродеревьев с разными мип-уровнями геометрических изображений будут образовываться разрывы из-за разной частоты дискретизации границы. Для устранения этой проблемы необходимо схлопывать рёбра со стороны вершины с бóльшим мип-уровнем согласно рисунку. (РИСУНОК) 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 необходимо и достаточно 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.

2.2.3. Графический пайплайн

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 мип-уровень, мип-уровни для сторон, 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 соотетствии с мип-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 (ССЫЛКИ!!!). Исходный код имеет размер порядка 10000 строк и доступен по ссылке https://github.com/Mrkol/thesis_vgi.

В качестве тестовых моделей были выбраны 3D-сканы Quixel Megascans [14] 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.

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

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 бóльшее число квадродеревьев.

4. План дальнейших исследований

В силу большой модульности и сложности метода предобработки, практически любая его часть может быть улучшена как по производительности, так и по качеству результата.

Для различных этапов предобработки планируется реализовать некоторое количество альтернативных подходов и сравнить получаемое качество, а также время предобработки. Из примечательных моментов – поддержка разрывных вершинных атрибутов посредством проведения границ кластеров по границам разрывов, ускорение этапа репараметризации используя иные алгоритмы оптимизации, а также исследование модификаций все техники предлагаемых в [11].

Ещё одним крупным направлением для дальнейших исследований являются алгоритмы адаптации при рендеринге иерархического атласа. Также планируется внедрение параллелизма в текущую имплементацию рендеринга, а также рассмотрение возможности перенесения вычислений нагружающих центральный процессор в текущей имплементации (адаптация, вычисление мип уровней для границ) на графический процессор.

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/pdf/10.1111/j.1467-8659.2007.01108.x.

[4] 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.

[5] Epic Games, Inc. Unreal Engine. — 2020. — Access mode: https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5.

[6] Karis Brian. More Geometry. — 2009. — Access mode: http://graphicrants.blogspot.com/2009/01/more-geometry.html (online; accessed: 2020-12-13).

[7] Karis Brian. Virtual Geometry Images. — 2009. — Access mode: http://graphicrants.blogspot.com/2009/01/virtual-geometry-images.html (online; accessed: 2020-12-13).

[8] Bunnell Michael. Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping // GPU Gems 2 / ed. by Pharr Matt. — Addison-Wesley, 2005. — P. 109–122.

[9] Olick Jon. Current and Next Generation Parallelism in Games. — 2008. — SIGGRAPH.

[10] Rendering continuous level-of-detail meshes by Masking Strips / Ripolles 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.

[11] 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.

[12] Epic Games, Inc. Nanite Technology. — 2020. — Access mode: https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5.

[13] Garland Michael, Willmott Andrew, Heckbert Paul S. Hierarchical Face Clustering on Polygonal 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.

[14] Epic Games, Inc. Quixel Megascans. — Access mode: https://quixel.com/megascans (online; accessed: 2021-05-27).

[15] Purnomo Budirijanto, Cohen Jonathan, Kumar Subodh. Seamless Texture Atlases. — 2004. — 01. — Vol. 71. — P. 67–76.

[16] 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.

[17] 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.