Обзор методов упрощения полигональных моделей на графическом процессоре

Гонахчян В.И.
ИСП РАН, Москва
pusheax@ispras.ru

Аннотация. Упрощение полигональных моделей является одной из распространенных методик, позволяющих увеличить скорость растеризации масштабных сцен, состоящих из большого количества сложных объектов. Традиционные алгоритмы, как правило, основанные на последовательном исключении ребер и граней, имеют высокую вычислительную сложность, что является препятствием для реализации ряда графических приложений на CPU. С развитием технологий программирования графического процессора открываются новые возможности для эффективной параллельной реализации данных алгоритмов. В работе обсуждаются некоторые известные алгоритмы упрощения полигональных моделей, использующие возможности распараллеливания независимых операций исключения ребер и спекулятивных оценок визуального качества редуцируемого полигонального представления. Сравниваются основные характеристики описанных алгоритмов и параллельных программ, а также даются рекомендации по их практическому использованию.

1. Введение

Трехмерная компьютерная графика — научная область, имеющая многочисленные приложения в визуализации научных и инженерных расчетов, САПР, анимации, играх. Как правило, необходимость реалистичной визуализации, желательно, в режиме реального времени приводит к довольно жестким требованиям к эффективности растеризации динамических пространственно–трехмерных сцен. В случае сложных сцен, состоящих из тысяч и миллионов элементов с индивидуальным геометрическим представлением, удовлетворить этим требованиям удается далеко не всегда даже с учетом перманентного технологического прогресса в развитии графических процессоров.

Одним из перспективных подходов к эффективной растеризации является использование альтернативных, как правило, упрощенных полигональных представлений для элементов сцен (LOD — аббревиатура от Level Of Details).

При растеризации сцены учитывается фактор близости элементов к камере обзора (более точно, угловое расстояние или пиксельное разрешение элементов), и оригинальная геометрическая модель каждого индивидуального элемента подменяется упрощенным представлением, которое обеспечивает необходимое визуальное качество итогового изображения. В частности, элементы с разрешением меньше одного экранного пиксела, вообще исключаются из обработки, а элементы, находящиеся на значительном удалении от камеры, заменяются полигональными моделями с меньшим количеством граней.

На рис. 1 тестовая геометрическая модель представлена четырьмя альтернативными полиэдрами с общим количеством треугольных граней 69666, 2500, 250 и 75 соответственно. При уменьшении количества граней визуальное качество модели заметно ухудшается. Однако по мере удаления от камеры это требование становится все менее критичным, и можно воспользоваться упрощенным представлением, которое бы существенно ускорило процесс растеризации.

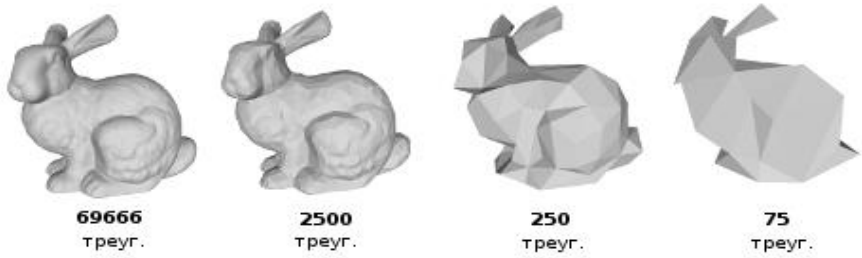

Рисунок 1. Уровни детализации полигональной модели Stanford bunny

На рис. 2 показана смена упрощенных представлений при удалении от камеры. Первый полиэдр содержит 20000 граней. Количество граней уменьшается в два раза при переходе к следующему представлению. Видно, что визуальное качество при этом остается примерно одинаковым.

Рассмотрим области применения методов упрощения полигональных сеток. В технологиях компьютерного зрения, в частности, технологиях лазерного сканирования используются программы для обнаружения, отслеживания, классификации объектов. Результатом их применения часто являются полигональные сетки с миллионами вершин. Примечательно, что необходимость их упрощения часто диктуется не только требованиями ускорить процесс визуализации, но и желанием увеличить точность представления исходных данных и снизить требования к вычислительным ресурсам, необходимым для работы с ними. При этом важно сохранить индивидуальные особенности объектов, связанные с их пропорциями,

наличием острых углов и дырок, для последующего отображения и распознавания.

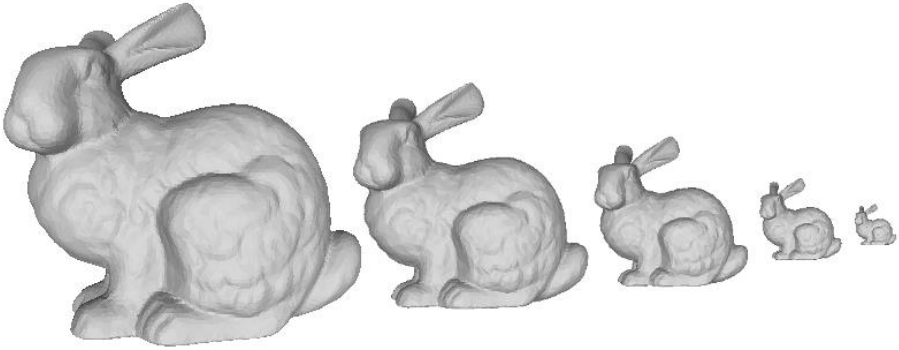

Рисунок 2. Смена уровней детализации полигональной модели при разных удалениях от экрана

Построение упрощенных геометрических моделей, обеспечивающих необходимое визуальное качество с точки зрения психофизики восприятия графических образов человеком, представляет собой нетривиальную задачу, порой требующую привлечения многих прикладных специалистов. Однако разработан мощный арсенал автоматических методов инкрементального упрощения исходной полигональной модели. В настоящей работе обсуждаются подобные возможности, при этом главное внимание уделяется методам, обладающим внутренним параллелизмом и допускающим эффективную реализацию на графических процессорах.

Традиционные автоматические методы упрощения моделей, как правило, основанные на последовательном исключении ребер и граней, имеют высокую вычислительную сложность, что является препятствием для реализации ряда графических приложений на CPU [5]. Вместе с тем, данные методы допускают распараллеливание независимых операций исключения ребер и спекулятивных оценок визуального качества редуцируемого полигонального представления. В работе [22] описана одна из первых программных реализаций параллельного алгоритма для составления многоуровневых сеток [18]. С развитием технологий программирования графического процессора открываются новые возможности для эффективной реализации данных алгоритмов в графических приложениях реального времени. Существующие инструменты программирования GPU, в частности, шейдерные языки CG, GLSL, HLSL и интегрированные GPGPU системы CUDA, OpenCL, предоставляют развитые средства реализации низкоуровневых операций с высокой степенью параллелизма.

Растеризация на GPU предполагает конвейеризацию вычислений с явным выделением следующих этапов:

- Этап обработки вершин, на котором к каждой вершине трехмерной модели применяются трансформации для позиционирования в системе координат сцены;
- Этап обработки фрагментов, на котором определяется цвет каждого пиксела с учетом источников света, материалов;
- Этап формирования итогового изображения с учетом видимости объектов сцены.

Более подробно графический конвейер обсуждается в [9].

Выделяют вершинные, геометрические и фрагментные шейдерные программы. Они соответствуют этапам графического конвейера. Организация шейдерных вычислений естественным образом ложится на архитектуру графического процессора. Вершины и пикселы обрабатываются параллельно по принципу SIMD (Single Instruction Multiple Data). Фрагментные процессоры имеют доступ к текстурам. Кадровый буфер может быть считан как текстура при последующих проходах отрисовки. Обычно шейдеры используются по прямому назначению, однако функциональная полнота операций графического процессора позволяет выполнять любые вычисления. Высокая степень параллельности сделала шейдеры хорошим инструментом для реализации различных операций с матрицами и массивами.

В отличие от шейдерного программирования в рамках концепции GPGPU можно использовать как графические, так и центральные процессоры (CPU). Существует две основных платформы для GPGPU программ: CUDA и OpenCL. CUDA — закрытая интегрированная система, разрабатываемая компанией Nvidia. OpenCL — открытый стандарт, разрабатываемый консорциумом Khronos Group.

В данной статье используются термины, применяемые в модели вычислений OpenCL. Устройство (device) состоит из вычислительных модулей (compute unit) с обрабатывающими элементами (processing element). Роль устройства может выполнять графический процессор или центральный процессор. Вычисления описываются на языке OpenCL C (подмножество ISO C '99) в ядрах (kernel) — специальных функциях. Ядра выполняются рабочими элементами (work item) в рабочих группах (work group). Рабочий элемент — экземпляр ядра программы с уникальным идентификатором. В рабочей группе есть локальная память (local memory) с возможностью быстрого доступа. Вычисления соответствуют модели SIMD (Single Instruction Multiple Data) — одна и та же программа выполняется несколькими вычислительными модулями с разными входными данными. В парадигме GPGPU обращение к глобальной памяти устройства и обмен данными между устройствами стоят относительно дорого.

В настоящей работе обсуждаются некоторые известные параллельные алгоритмы упрощения полигональных моделей, сравниваются их основные

характеристики, а также даются рекомендации по их практическому использованию.

2. Классификация методов упрощения

Алгоритмы упрощения сеток различаются по типам поверхностей, к которым они применяются. Одни работают только на многообразиях (manifold), другие также допускают полигональные сетки, не являющиеся многообразиями (non–manifold). Рассмотрим проблемы, которые возникают при упрощении топологии полигональной модели. Полигональная сетка, каждое ребро которой входит в два треугольника, является двумерным многообразием. Двумерное многообразие с границей допускает ребра, которые входят только в один треугольник. Немногообразные полигональные сетки имеют особенности, которые затрудняют их упрощение. В частности, существуют ребра, которые входят в более чем два треугольника, и вершины, которые являются единственными связующими точками для двух множеств треугольников. Такие участки полигональной сетки требуют отдельной обработки алгоритмом. Большинство алгоритмов пропускает данную часть модели. В случае, когда визуальное качество критично (например, для метода конечных элементов), алгоритмы могут завершиться с ошибкой при нахождении подобных участков. Чтобы избежать этого, требуется предварительно убедиться, что исходная модель является многообразием, и при необходимости преобразовать ее в требуемое представление. Однако есть алгоритмы, которые справляются и с немногообразными полигональными сетками [8].

Алгоритмы также различаются по способу удаления геометрии: прореживание (decimation), выборка (sampling).

Прореживание — итеративный метод удаления геометрии полигональной модели. За одну итерацию объединяются вершины, удаляется вершина, ребро или треугольник с последующей триангуляцией для заполнения дырки при необходимости. При достижении целевого числа полигонов алгоритм завершается.

Рассмотрим основные операции по удалению геометрии при прореживании. На рис. 3а изображена операция объединения вершин, в результате которой три вершины замещаются одной. При этом удаляются все ребра и треугольники с их участием, затем выполняется триангуляция. На рис. 3б изображена операция удаления вершины со всеми зависимыми ребрами и треугольниками. На рис. 3в представлена операция удаления ребра, в результате которой два треугольника становятся вырожденными. При схлопывании ребра (edge collapse) удаляется ребро и вырожденные треугольники. Схлопывание пары (pair collapse) работает таким же образом, но допускает отсутствие ребра между вершинами. На рис. 3г показана операция удаления треугольника с последующей триангуляцией.

Выборка позволяет за несколько шагов существенно сократить количество граней модели. При выборке исходную модель пересекают с трехмерной сеткой заданного размера и упрощают геометрию в каждой ячейке. Обычно используют равномерную сетку. Недостаток этого подхода заключается в том, что плотность вершин в моделях зачастую неравномерная. В результате качество может ухудшиться.

Помимо основных категорий в обзоре также рассматриваются следующие характеристики методов:

- Используемая операция упрощения,
- Точность представления исходной сетки,
- Сравнение производительности и визуального качества с реализацией на CPU.

3. Методы упрощения

Метод итеративного прореживания полигональной сетки [12] предназначен для упрощения результата алгоритма "marching cubes". За каждый проход принимается решение об удалении каждой вершины. Если вершину можно удалить без изменения топологии с допустимой точностью, то алгоритм удаляет ее и все связанные с ней треугольники. Это оставляет дырку в модели, которая затем заполняется треугольниками. Упрощение продолжается до тех пор, пока не закончатся подходящие вершины. Алгоритм, приведенный в работе [16], расширяет данный метод путем прореживания с постепенным уменьшением точности представления.

При кластеризации вершин [13] происходит наложение равномерной сетки на модель для упрощения топологии. В каждой ячейке оставляют одну вершину, которая входит в треугольники с наибольшей суммарной площадью. Этот метод имеет линейную сложность (O(n) для n вершин), но дает относительно низкое визуальное качество из-за изменения топологии и отсутствия возможности указать точность упрощения. Алгоритм, приведенный в работе [15], расширяет данный метод, используя более эффективное пространственное деление, которое называется кластеризацией с плавающей ячейкой (floating–cell clustering). В нем вершины сортируются по приоритету, и на месте вершины с наивысшим приоритетом создается ячейка, в которой производится упрощение. Далее выбирается следующая вершина, и процесс повторяется. Отсутствие сетки в этом алгоритме позволяет упрощать модели вне зависимости от положения и ориентации. Алгоритм, приведенный в работе [4], в отличие от [15], работает с моделью во внешней памяти, используя оперативную память только для хранения результата упрощения. Для оценки точности представления используется квадратичный многочлен [1].

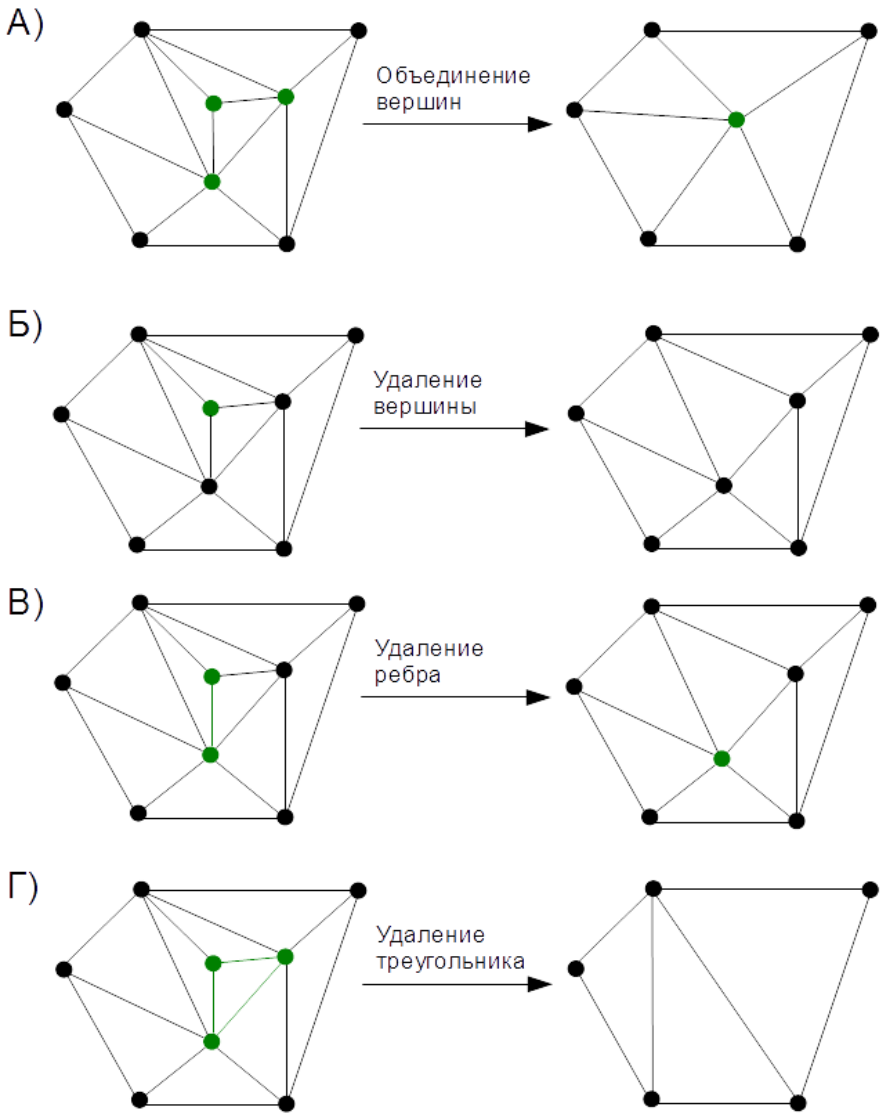

Рисунок 3. Базовые операции по изменению геометрии при прореживании

Метод воксельного упрощения [14] позволяет упрощать топологию постепенно с использованием техники обработки сигналов. Сначала на полигональную модель накладывается трехмерная сетка из вокселей. Каждому вокселю присваивается значение 1, когда он находится внутри модели, либо 0 — вне модели. Затем применяется фильтр нижних частот, который удаляет

характерные особенности на высоких частотах. Далее применяется алгоритм "marching cubes", который восстанавливает полигональное представление модели. Главным недостатком алгоритма является ухудшение визуального качества за счет удаления ряда характерных особенностей.

В работе [17] используются конверты упрощения (envelopes) для гарантирования точности упрощения. Создаются две граничные поверхности, которые смещены относительно исходной на константы δ и -δ. В случае самопересечения этих поверхностей алгоритм понижает значение константы. Затем производится упрощение и делается проверка того, что упрощенная поверхность содержится в пределах граничных поверхностей. Данный метод сохраняет топологию и работает только на ориентируемых многообразиях.

В работе [18] описан способ прореживания многообразия для создания многоуровневых сеток (progressive mesh). Многоуровневая сетка состоит из начального представления и набора операций по разделению вершин (vertex split). Операция разделения вершины является обратной по отношению к операции схлопывания ребра. Применение всех операций по разделению вершин восстанавливает исходную модель. При этом многоуровневые сетки рассчитаны на визуализацию в реальном времени частично восстановленной модели. В работе [21] представлена общая техника на основе метода [18], которая жертвует скоростью для того, чтобы принимать на вход произвольные полигональные сетки.

В работе [19] описан алгоритм, использующий квадратичный многочлен для определения точности представления в вершинах. Квадратичный многочлен задается матрицей размерности 4x4, которая позволяет найти сумму квадратов расстояний от вершины до плоскостей соседних треугольников. Сначала выполняются операции по удалению вершин, которые дают наивысшую точность упрощения. Алгоритм итеративно объединяет пары вершин, между которыми может и не быть ребра. Для определения точности упрощения в новой вершине выполняют сложение матриц, соответствующих квадратичным многочленам удаленных вершин. Это позволяет упрощать топологию при гарантированной точности представления исходной сетки. Невозможно подобрать одну точность упрощения, которая дает высокое визуальное качество для различных полигональных сеток. Алгоритм, приведенный в [20], является развитием данного метода для упрощения полигональных моделей с приписанными материалами.

4. Методы упрощения на графическом процессоре

Метод, предложенный в работе [2], совмещает выборку и прореживание полигональных моделей в реальном времени с помощью шейдерных программ для DirectX 10 [3], не сохраняет топологию, работает на многообразиях. За основу взят алгоритм из работы [4]. Предложен способ кластеризации вершин на графическом процессоре. Граничная рамка объекта равномерно разбивается на ячейки. В каждой ячейке оставляют одну вершину,

которая дает наивысшую точность упрощения согласно квадратичному многочлену для определения суммы квадратов расстояний до соседних граней [1]. Алгоритм работает в три прохода графического процессора, используя сохраненные буферы на последующих проходах. Высокая степень параллельности достигается путем применения независимых операций в каждой ячейке кластера. Использование операции схлопывания пары позволяет получить качественно лучшие результаты по сравнению с традиционной кластеризацией. Реализованный алгоритм дает прирост производительности от 15 до 22 раз по сравнению с версией на CPU. Этот метод можно использовать для упрощения автоматически сгенерированной геометрии или при загрузке моделей для повышения производительности. Упрощенные модели можно использовать для обнаружения столкновений.

Метод, предложенный в работе [6], выполняет прореживание полигональных моделей с помощью OpenCL, сохраняет топологию, работает на многообразиях. Исходная модель разбивается на сегменты по 700 треугольников, чтобы не выйти за пределы 32КБ локальной памяти, отведенной для вычислительного модуля. Далее выполняется параллельное упрощение на обрабатывающих элементах. Результат передается в основную память для встраивания в исходную модель. При упрощении используется операция схлопывания ребра. Для оценки стоимости схлопывания ребра используется объемная точность упрощения [7]. Следующее ребро выбирается без сортировки параллельно с помощью обрабатывающих элементов. При достижении заданной точности упрощение прекращается. Утверждается, что алгоритм дает прирост производительности, сравнимый с кластеризацией вершин (от 15 до 22 раз согласно [2]), но обладает низким визуальным качеством из-за поддержания границ сегментов в исходном виде для выполнения встраивания.

Метод, предложенный в работе [8], выполняет прореживание полигональных моделей с помощью шейдерных программ с использованием OpenGL, сохраняет топологию, работает на многообразиях и немногообразных полигональных сетках. Традиционные методы оценки точности упрощения основаны на форме объектов. В статье обсуждаются пользовательские критерии для задания приоритета удаления ребер. Если конечный элемент достигает предельного размера, то его упрощение прекращается вне зависимости от того, сколько в нем осталось граней. В статье описан гибридный подход, при котором одна часть операций выполняется центральным процессором, а другая — графическим процессором. Алгоритм выделяет независимые множества и сортирует их по значению дискретной кривизны (discrete curvature). Вершины независимого множества не должны быть непосредственными соседями (one–ring neighbourhood) других вершин этого же множества. Независимые множества сохраняются в текстурах графического процессора, который применяет критерий для схлопывания ребер и выполняет перестроение сетки. В качестве критерия для схлопывания ребра используется значение односторонней метрики Хаусдорфа. Для этого

каждой вершине назначается область предельной точности (envelope) [17]. После операции схлопывания ребра область становится активной. При пересечении каждой активной области хотя бы с одним треугольником упрощенной модели упрощение считается корректным. Далее эта область назначается упрощенному треугольнику. Применение критерия на графическом процессоре ограничивается независимым множеством вершин. Критерий по объему применяется на CPU. Если удаление ребра меняет топологию, то его пропускают. Немногообразные полигональные сетки обрабатываются центральным процессором. Утверждается, что алгоритм в 8 раз быстрее версии на CPU при упрощении моделей с большим количеством граней. Этот подход используется в методе конечных элементов, где важно работать с упрощенным представлением и требуется удаление характерных особенностей объекта.

Метод, предложенный в работе [10], выполняет прореживание полигональных моделей с помощью OpenCL, сохраняет топологию, работает на многообразиях. Применяется операция схлопывания ребра для постепенного упрощения модели. Для задания точности упрощения в вершине используют квадратичный многочлен [1]. Весь алгоритм, за исключением стадии инициализации данных, выполняется на графическом процессоре. На вход передается модель и желаемое количество вершин на выходе. Алгоритм поддерживает в оперативной памяти массивы с дополнительной информацией о вершинах. Для параллелизации работы определяются независимые множества модели. Независимое множество состоит из вершин, расстояние между которыми в графе вершин модели не меньше трех. На каждой итерации алгоритм находит независимые множества у всей модели, сортирует их параллельно с использованием сортировки "bitonic sorting", упрощает на вычислительных модулях GPU. Утверждается, что прирост производительности составляет 1.5–2.5 раза по сравнению с версией на CPU при сравнимом визуальном качестве.

Метод, предложенный в работе [11], выполняет прореживание полигональных моделей с помощью CUDA, сохраняет топологию, работает на многообразиях. В статье описан гибридный подход к упрощению моделей, который использует центральный и графический процессоры в заданном процентном соотношении. Задачи постепенно назначаются на свободные потоки CPU и GPU в цикле. Стоит отметить, что при этом не учитываются особенности работы графической системы, как это сделано в работе [6], идет расчет на оптимизацию вычислений в CUDA. При выборе следующего треугольника для параллельного упрощения используется механизм блокировки вершин на основе метода "test–and–set". При успешной блокировке треугольник помечается, и производится операция схлопывания ребра. Таким образом, гарантируется равномерность упрощения. Таблицы CPU–GPU при сравнении соотношений 1:0 и 0:1 показывают незначительный прирост производительности в 15% на больших моделях.

Все рассмотренные методы выполняют прореживание геометрии. В [2] используется гибридный подход — выборка с прореживанием. Во всех методах, кроме [2], в качестве операции упрощения используется схлопывание ребра. В [2] используется операция схлопывания пары. В результате все методы, за исключением [2], сохраняют топологию поверхности. В [8] в качестве исходной модели допускаются немногообразные полигональные сетки. Остальные характеристики приведены в табл. 1.

Статья	Реализация	Метод оценки точности упрощения	Производительность по сравнению с CPU
Decoro–Tatarchuk [2]	Шейдеры	Квадратичный многочлен	15–22 раза
Vad'ura [6]	OpenCL	Объемная мера	—
Hjelmervik–Leon [8]	OpenGL	Расстояние Хаусдорфа	8 раз
Papageorgiou–Platis [10]	OpenCL	Квадратичный многочлен	1.5–2.5 раза
Shontz–Nistor [11]	CUDA	—	15%

Таблица 1. Основные характеристики методов упрощения моделей на GPU

5. Заключение

Таким образом, рассмотрены некоторые известные алгоритмы упрощения полигональных моделей, использующие возможности распараллеливания независимых операций исключения ребер и спекулятивных оценок визуального качества редуцируемого полигонального представления. Проведено сравнение основных характеристик описанных алгоритмов и реализующих их параллельных программ. В частности, разобраны случаи, в которых технологии программирования графических процессоров имеют наибольший успех.

Шейдерные методы дают большой прирост производительности [2], но также имеют свои недостатки. Они предназначены для упрощения в реальном времени, что сильно ограничивает их область применения. Графический процессор в первую очередь рассчитан на растеризацию сцен, а не предобработку трехмерных данных, что вызывает сложности в написании и поддержке кода подобных методов. Также при использовании кластеризации

вершин, которая хорошо распараллеливается с помощью шейдеров, визуальное качество получается хуже, чем при прореживании.

Методы с использованием интегрированных сред OpenCL, CUDA проще в реализации. Допускается загрузка результатов вычислений в основную память. Однако при подходах, которые не учитывают особенности вычислительной модели графического процессора, прирост производительности меньше, чем у шейдерных методов [11], [10]. В [6] показано, что требуется учет низкоуровневых деталей (количества локальной памяти и потоковых процессоров), чтобы задействовать одновременно максимальное количество потоковых процессоров. При этом визуальное качество получается хуже последовательной реализации из-за наличия границ между сегментами с повышенной плотностью вершин.

Отметим, что в качестве основной операции по упрощению чаще всего выбирают схлопывание ребра. Это делается не только из соображений простоты, но и потому что при этой операции не требуется последующая триангуляция, как при удалении треугольников или вершин. Это хорошо подходит для графической системы, т. к. она плохо приспособлена для добавления дополнительных треугольников при заполнении дырок.

Не все алгоритмы допускают эффективную параллельную реализацию. Для обсуждаемых задач, как правило, удается выделить независимые операции или подзадачи, которые можно выполнить одновременно. Однако узким местом остается необходимость организации эффективного обмена данными, для чего часто развертываются дополнительные структуры данных в основной памяти [8], [10]. Сбалансированная загрузка центрального и графического процессоров также является проблемой с учетом специфики решаемых подзадач.

Список литературы

[1] M. Garland, P. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 209–216, 1997.

[2] C. DeCoro, N. Tatarchuk, "Real–time Mesh Simplification Using GPU," in *Proceedings of the symposium on interactive 3D graphics and games*, 2007.

[3] D. Blythe, "The Direct3D 10 System," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 724–734, 2006.

[4] P. Lindstrom, "Out–of–core simplification of large polygonal models," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 259–262, 2000.

[5] J. Rossignac, P. Borrel, "Multi–resolution 3D approximations for rendering complex scenes.," *Modeling in Computer Graphics: Methods and Applications*, pp. 455–465, 1993.

[6] J. Vad'ura, "Parallel mesh decimation with GPU," 2011.

[7] A. Gueziec, "Surface simplification inside a tolerance volume," *Second Annual International Symposium on Medical Robotics and Computer Aided Surgery*, pp. 123–139, 1995.

[8] J. Hjelmervik, J. Leon, "GPU–accelerated shape simplification for mechanical based applications," *Shape modeling international*, pp. 91–102, IEEE Computer Society, 2007.
[9] D. Shreiner, M. Woo, J. Neider, T. Davis, "OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2," 2005.
[10] A. Papageorgiou, N. Platis, "Triangular mesh simplification on the GPU," *NASAGEM Geometry Processing Workshop*, Computer Graphics International, 2013.
[11] S. Shontz, D. Nistor, "CPU–GPU algorithms for triangular surface mesh simplification," in *Proceedings of the 21st international meshing roundtable*, pp. 475–492, Springer, Berlin, 2013.
[12] W. Schroeder, J. Zarge, and W. Lorensen, "Decimation of Triangle Meshes," *ACM Siggraph Computer Graphics*, vol. 26, no. 2, pp. 65–70, 1992.
[13] J. Rossignac and P. Borrel, "Multi–Resolution 3D Approximations for Rendering Complex Scenes," *Geometric Modeling in Computer Graphics*, pp. 455–465, 1993.
[14] T. He, L. Hong, A. Kaufman, A. Varshney and S. Wang, "Voxel–Based Object Simplification," in *Proceedings of the 6th conference on Visualization '95*, pp. 296–303, 1995.
[15] K. Low and T. Tan, "Model Simplification Using Vertex–Clustering," in *Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 75–82, 1997.
[16] W. Schroeder, "A Topology Modifying Progressive Decimation Algorithm," in *Proceedings of the 8th conference on Visualization '97*, pp. 205–212, 1997.
[17] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification Envelopes," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 119–128, 1996.
[18] H. Hoppe, "Progressive Meshes," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 99–108, 1996.
[19] M. Garland and P. Heckbert, "Surface Simplification Using Quadric Error Metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 209–216, 1997.
[20] M. Garland and P. Heckbert, "Simplifying Surfaces with Color and Texture using Quadric Error Metrics," in *Proceedings of the conference on Visualization '98*, pp. 263–269, 1998.
[21] J. Popovic and H. Hoppe, "Progressive Simplicial Complexes," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 217–224, 1997.
[22] F. Dehne, C. Langis, G. Roth, "Mesh simplification in parallel," in *Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing*, pp. 281–290, 2000.

Survey of polygonal surface simplification algorithms on GPU

V.I. Gonakhchyan
ISP RAS, Moscow, Russia
pusheax@ispras.ru

Annotation. Rendering time depends on number of faces in polygonal mesh. Displaying large number of objects or objects with a lot of faces can result in performance degradation. One of the ways to overcome this problem is to use several levels of detail for polygonal mesh. Levels of detail with low quality are used when object is far away from camera and difference in number of faces is not apparent to viewer. Coarse meshes can be generated manually which is difficult and time consuming. For many applications (computer graphics, computer vision, finite element analysis) it's preferable to use automated methods for polygonal surface simplification to increase performance. Traditional methods for surface simplification are designed for CPU. GPU methods provide improvements in performance and have comparable visual quality. This survey covers algorithms on GPU. They are based on either shader programming or general purpose computing frameworks like OpenCL and CUDA. Shader methods are very restrictive. They are designed to render images in real–time. General purpose computing frameworks provide flexible APIs but require attention to hardware implementation details to avoid performance bottlenecks. Main features of algorithms are provided for comparison as the result. The most used operation for geometry simplification is edge collapse. It's difficult to avoid expensive data exchange between GPU and CPU. It's important to take into account computational model of GPU to increase number of stream processors working in parallel.

Key words: mesh simplification, level–of–detail, GPU programming

References

[1] Garland M., Heckbert P. "Surface simplification using quadric error metrics" in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 209–216, 1997.
[2] DeCoro C., Tatarchuk N. "Real–time Mesh Simplification Using GPU" in *Proceedings of the symposium on interactive 3D graphics and games*, 2007.
[3] Blythe D. "The Direct3D 10 System" in *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 724–734, 2006.
[4] Lindstrom P. "Out–of–core simplification of large polygonal models" in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 259–262, 2000.
[5] Rossignac J., Borrel P. "Multi–resolution 3D approximations for rendering complex scenes" *Modeling in Computer Graphics: Methods and Applications*, pp. 455–465, 1993.
[6] Vad'ura J. "Parallel mesh decimation with GPU" 2011.

[7] Gueziec A. "Surface simplification inside a tolerance volume" *Second Annual International Symposium on Medical Robotics and Computer Aided Surgery*, pp. 123–139, 1995.
[8] Hjelmervik J., Leon J. "GPU–accelerated shape simplification for mechanical based applications" *Shape modeling international*, pp. 91–102, IEEE Computer Society, 2007.
[9] Shreiner D., Woo M., Neider J., Davis T. "OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2", 2005.
[10] Papageorgiou A., Platis N. "Triangular mesh simplification on the GPU" *NASAGEM Geometry Processing Workshop*, Computer Graphics International, 2013.
[11] Shontz S., Nistor D. "CPU–GPU algorithms for triangular surface mesh simplification" in *Proceedings of the 21st international meshing roundtable*, pp. 475–492, Springer, Berlin, 2013.
[12] Schroeder W., Zarge J. and Lorensen W. "Decimation of Triangle Meshes" *ACM Siggraph Computer Graphics*, vol. 26, no. 2, pp. 65–70, 1992.
[13] Rossignac J. and Borrel P. "Multi–Resolution 3D Approximations for Rendering Complex Scenes" *Geometric Modeling in Computer Graphics*, pp. 455–465, 1993.
[14] He T., Hong L., Kaufman A., Varshney A. and Wang S. "Voxel–Based Object Simplification" in *Proceedings of the 6th conference on Visualization '95*, pp. 296–303, 1995.
[15] Low K. and Tan T. "Model Simplification Using Vertex–Clustering" in *Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 75–82, 1997.
[16] Schroeder W. "A Topology Modifying Progressive Decimation Algorithm" in *Proceedings of the 8th conference on Visualization '97*, pp. 205–212, 1997.
[17] Cohen J., Varshney A., Manocha D., Turk G., Weber H., Agarwal P., Brooks F. and W. Wright "Simplification Envelopes" in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 119–128, 1996.
[18] Hoppe H. "Progressive Meshes" in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 209–216, 1997.
[19] Garland M. and Heckbert P. "Surface Simplification Using Quadric Error Metrics" in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 209–216, 1997.
[20] Garland M. and Heckbert P. "Simplifying Surfaces with Color and Texture using Quadric Error Metrics" in *Proceedings of the conference on Visualization '98*, pp. 263–269, 1998.
[21] Popovic J. and Hoppe H. "Progressive Simplicial Complexes" in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 217–224, 1997.
[22] Dehne F., Langis C. and Roth G. "Mesh simplification in parallel" in *Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing*, pp. 281–290, 2000.