Обзор алгоритмов триангуляции неявно заданной поверхности*

[1]*Бугров Н.В.*, [1]*Голубев В.И.*, [2]*Дижевский А.Ю.*, [1]*Какауридзе Д.Г.*,
[2]*Клименко А.С.*, [1]*Обоймов А.С.*, [2]*Фролов П.В.*
nikita.bugrov@gmail.com, vasily.golubev@gmail.com, mathlog@yandex.ru,
kakauridze.d@gmail.com, andy.klimenko@gmail.com, anton.oboimov@gmail.com,
pavel.frolov@gmail.com
[1]Московский физико-технический институт (государственный университет);
[2]Протвино, Институт физико-технической информатики

Алгоритмы создания реалистических изображений должны обеспечивать передачу свойств моделируемого объекта: объемность, расположение, передача полутонов, тени, освещение, текстуры поверхности и т.д. Генерация таких объемных изображений представляет собой очень сложную вычислительную задачу. В связи с этим на практике выполняют её декомпозицию: объекты разбиваются на составные части, и сложные изображения формируются из получаемых фрагментов. На практике, наиболее часто применяется разбиение изображений на треугольники. Это связано с тем, что треугольник - простейший полигон, вершины которого однозначно задают грань. Алгоритмы разбиения на треугольники существенно проще, чем при использовании других полигонов. Разбиение на треугольники существенно упрощает реализацию процедур рендеринга. Ну и наконец, на треугольники можно гарантированно разбить любую область, чего не скажешь о других полигонах. Процесс разбиения полигональной области со сложной конфигурацией в набор треугольников называется триангуляцией.
В данной статье рассматриваются алгоритмы триангуляции и визуализации неявно заданных поверхностей. Проводится исследование существующих алгоритмов триангуляции, отмечаются преимущества и недостатки различных подходов.

Введение

Под визуализацией неявно заданной поверхности понимается визуализация поверхности, заданной с помощью функции от трех аргументов и фиксированного значения этой функции - уровня.

$$(x, y, z)|f(x, y, z) = c) \tag{1}$$

где $f(x, y, z)$ - заданная функция, а c - заданный уровень.

Алгоритмы визуализации поверхности (surface rendering) строят изображение поверхности в трехмерном пространстве. Однако удобнее восстанавливать не саму поверхность, а поверхность аппроксимирующую искомую с помощью треугольников. Таким образом, сначала исходная поверхность аппроксимируется полигонами, затем производится визуализация полигонов с помощью графических библиотек.

Проблема визуализации поверхности, заданной различными способами, возникает во многих областях математики, физики, медицины:

— *Визуализация экспериментальных данных.* При проведении физических экспериментов очень часто возникает необходимость отобразить информацию сразу со всех датчиков. Например, при измерении температуры среды необходимо отобразить область с температурой, выше заданной.

— *Функциональное представление.* В некоторых математических задачах или расчетах необходимо визуализировать геометрический объект, заданный с помощью одной вещественной непрерывной описывающей функции нескольких переменных в виде $F(X) > 0$. Также может возникнуть общая задача, в которой описывающая функция задана с помощью множества точек, в которых известно её значение.

— *Медицина.* Использование компьютеров дало возможность развиваться новым направлениям томографической интроскопии, таким как компьютерная томография, магнитная резонансная томография и позитронная эмиссионная томография. С помощью томографической аппаратуры можно получить снимки множества сечений тела пациента, которые характеризуют особенности его анатомии и физиологии. Эти снимки с чрезвычайной четкостью показывают различные органы, причем изображения органов не налагаются друг на друга. Методы визуализации позволяют реконструировать трехмерную структуру органов по множеству параллельных сечений. Во многих случаях для установления диагноза врач зрительно анализирует изображения отдельных сечений объекта, полученных при томографическом обследовании. Однако, для некоторых клинических задач, подобных хирургическому планированию, необходимо понимать трехмерную структуру во всей ее сложности и видеть дефекты. Опыт показал, что «умозрительная реконструкция» объектов по изображениям их сечений чрезвычайно труд-

Работа выполнена при финансовой поддержке РФФИ, проект 12-07-33059.

на и сильно зависит от опыта и воображения наблюдателя. В таких случаях хотелось бы представить человеческое тело так, как его увидел бы хирург или анатом.

Аналогичные подходы применяются также для экспериментальных и модельных данных, которые получают в других областях, таких как динамика жидкостей, геология, метеорология, молекулярный анализ.

При решении задачи визуализации важную роль играет способ задания функции, которая описывает искомую поверхность. В большинстве прикладных задач функция задается таблично на регулярной сетке или имеет явное отображение, описываемое заданной формулой (1).

Но иногда возникают задачи, в которых нет явно заданного отображения, или таблица значений задана на нерегулярной сетке.

Такие задачи могут возникать во многих приложениях, например, в задаче измерения расстояния до поверхности с помощью облучения или в задаче реконструкции трехмерной структуры с помощью множества контуров-«срезов» в медицинских исследованиях.

В таких задачах используется следующий алгоритм действий [41]: поверхность S, заданная выборкой X, аппроксимируется касательными плоскостями, проходящими через каждую точку выборки X.

Затем искомая функция, задающая поверхность, считается следующим образом: для каждой точки пространства R функция в этой точке равна расстоянию до ближайшей касательной плоскости, взятому со знаком «+», если точка находится внутри объема, ограниченного построенными плоскостями, или со знаком «−», если точка находится вне этого объема. Затем проводится триангуляция поверхности, заданной с помощью получившейся функции.

Понятно, что задача построения триангуляции по исходному набору точек является неоднозначной, поэтому возникает вопрос, какая из возможных триангуляций лучше.

Триангуляцию называют оптимальной, если сумма длин всех ребер минимальна среди всех возможных триангуляций исходного множества точек.

Однако, задача построения такой триангуляции является NP-полной. В работах [50, 55] показано, что её сложность составляет $O(e^N)$, поэтому на практике используются приближенные алгоритмы. Одним из первых был предложен такой алгоритм [7].

Начало алгоритма.

Шаг 1. Генерируется список всех возможных отрезков, соединяющих пары исходных точек, и он сортируется по длинам отрезков.

Шаг 2. Начиная с самого короткого, последовательно выполняется вставка отрезков в триангуляцию. Если отрезок не пересекается с другими ранее вставленными отрезками, то он вставляется, иначе отбрасывается.

Конец алгоритма.

Заметим, что если все возможные отрезки имеют разную длину, то результат этого алгоритма однозначен, иначе он зависит от порядка вставки отрезков одинаковой длины.

Этот алгоритм является жадным, поэтому построенная им триангуляция тоже называется жадной.

Трудоемкость работы жадного алгоритма при некоторых его улучшениях составляет $O(N^2 \log N)$. В связи с такой большой трудоемкостью алгоритм на практике практически не применяется.

Методы решения задачи триангуляции

Кроме оптимальной и жадной, существуют и другие триангуляции и алгоритмы их построения. Все методы решения задачи триангуляции можно разбить на следующие три группы [11].

— *Ячеечные методы (cell-based).* В методах такого типа происходит разбиение области триангуляции на ячейки - параллелепипеды [52, 27, 58, 51, 56], треугольные пирамиды [25, 38, 65, 66] , тетраэдры, октаэдры [3] и т.д. Далее производится триангуляция поверхности в каждой ячейке отдельно. Причем каждая ячейка триангулируется одним из заданных ранее способов, т.е. значения координат для треугольников просто "подставляются"из заранее заданной таблицы. Для применения методов этого типа необходимо задать допустимую ошибку аппроксимации, на основе которой выбрать размер ячейки. После этого с помощью уже известных таблиц триангуляции получить искомое множество треугольников. При этом процедура триангуляции каждой ячейки сводится к анализу значений функции в вершинах этой ячейки - другими словами, определяется, какие вершины лежат "внутри"поверхности, а какие - "снаружи". На основе этого можно сделать вывод о достаточности определения функции только в вершинах ячеек. Наиболее известные ячеечные алгоритмы: метод Канейро (Caneiro) [25], метод, предложенный Гуэзеком [38], метод Скалы (Skala) [65], метод "Марширующих кубов"[52].
— *Метод предиктора-корректора (predictor-corrector).* Методы из этого класса основаны на добавле-

Рис. 1. Визуализация мочевого пузыря человека. Слева - данные, полученные с помощью УЗИ, справа - реконструированная поверхность.

нии к уже имеющемуся множеству точек триангуляции ещё одной, лежащей на касательной плоскости к заданной функции (это положение предиктора (predictor) - предсказанное) и затем передвижении её до визуализируемой поверхности (это положение корректора (corrector) - скорректированное).

При использовании методов [39, 43] из этого класса, необходимо знать значение функции во всех точках пространства и найти хотя бы одну точку, принадлежащую искомой поверхности. Метод заключается в итеративном увеличении количества треугольников - на каждой итерации метода к уже существующему множеству треугольников добавляется еще один, построенный на ребре крайнего треугольника и предсказанной (а затем скорректированной по кривизне поверхности) точки на поверхности.

— *Мозаичные методы (pre-tessellation methods & particle-based methods).* Суть таких методов [28, 22, 67] заключается в разбиении искомой поверхности на части для дальнейшей их триангуляции. Разбиение на части в pre-tessellation методах подразумевает разбиение поверхности на «примитивные» поверхности - фрагменты сфер и плоскостей. Разбиение на части в методах из плеяды particle-based менее «интеллектуально» - ищутся только фрагменты плоскостей. При этом возникает проблема «соединения» уже «протриангулированных» частей. Чаще всего этот процесс сводится к триангуляции Делоне, то есть, к подбору локальных по Делоне треугольников, соединяющих части искомой поверхности.

У всех этих методов есть как свои достоинства, так и недостатки. Основа методов первого типа - независимая триангуляция каждой ячейки с помощью таблиц триангуляции - является одновременно их сильной и слабой стороной. Высокая скорость работы этих методов делает их наиболее привлекательными по отношению к другим методам и дает возможность использовать их в интерактивных приложениях, но большим минусом считается их относительная индифферентность к поведению функции вне выбранного множества точек. Другими словами это невозможность правильно визуализировать локальные искривления - масштаб треугольников всегда пропорционален размеру ячейки. Такие методы идеально подходят для визуализации трехмерных скалярных полей, заданных на регулярной сетке.

Методы второго и третьего типа применимы только при визуализации полей определенных в каждой точке той части пространства, которое нас интересует. Большим плюсом таких методов можно считать их зависимость от локального искривления функции - в таких методах мелкие детали не "пропадут". Несмотря на сильную потерю в скорости по сравнению с методами первой группы и ограничениями на дифференцируемость функции и связность поверхности, они привлекают высоким "качеством"получаемой поверхности.

Алгоритмы построения триангуляции Делоне

Впервые задача построения триангуляции Делоне была поставлена в 1934 г. в работе советского математика Б.Н.Делоне [2]. Триангуляцию называют триангуляцией Делоне, если внутрь окружности, описанной вокруг любого построенного треугольника, не попадает ни одна из заданных точек триангуляции (рисунок 2).

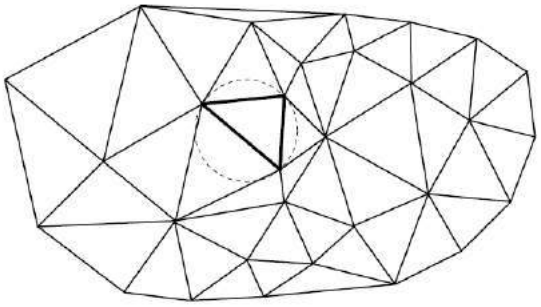

Рис. 2. Триангуляция Делоне.

Ее трудоемкость составляет $O(NlogN)$ арифметических операций. Существуют алгоритмы, достигающие этой оценки в среднем и худшем случаях. Кроме того, известны алгоритмы, позволяющие в ряде случаев достичь в среднем $O(N)$. Подробно многие алгоритмы рассмотрены и классифицированы, например, в работе [13]. Там же приводится оценки их эффективности и трудоемкости.

Приведем список наиболее известных алгоритмов построения триангуляции Делоне, с оценкой их эффективности.

1. Итеративные алгоритмы
   (а) Простой итеративный алгоритм
   (б) Алгоритмы с индексированием поиска треугольников
   (в) Алгоритмы с кэшированием поиска треугольников
   (г) Алгоритмы с изменённым порядком добавления точек

   Наиболее эффективным из итеративных алгоритмов, согласно [14], является алгоритм с динамическим кэшированием поиска треугольников, его трудоёмкость в среднем составляет (N).
2. Алгоритмы построения триангуляции Делоне слиянием
   (а) Алгоритм слияния «Разделяй и властвуй»
   (б) Рекурсивный алгоритм с разрезанием по диаметру
   (в) Полосовые алгоритмы слияния

   Наиболее эффективным из алгоритмов слияния, согласно [14], является алгоритм невыпуклого полосового слияния, его трудоёмкость в среднем составляет (N).
3. Алгоритмы прямого построения триангуляции Делоне
   (а) Пошаговый алгоритм
   (б) Пошаговые алгоритмы с ускорением поиска соседей Делоне

   Наиболее эффективным из алгоритмов прямого построения триангуляции Делоне, согласно [14], является пошаговый клеточный алгоритм, его трудоёмкость в среднем составляет (N).
4. Двухпроходные алгоритмы построения триангуляции Делоне
   (а) Двухпроходные алгоритмы слияния
   (б) Модифицированный иерархический алгоритм
   (в) Линейный алгоритм
   (г) Веерный алгоритм
   (д) Алгоритм рекурсивного расщепления
   (е) Ленточный алгоритм

   Наиболее эффективным из двухпроходных алгоритмов построения триангуляции Делоне, согласно [14], является двухпроходный алгоритм невыпуклого полосового слияния, его трудоёмкость в среднем составляет (N).

В настоящее время многие продолжают работать над усовершенствованием известных и созданием новых алгоритмов. Это обусловлено неустойчивостью ряда известных алгоритмов и неудовлетворительным временем их работы на реальных наборах данных.

Ниже описаны некоторые, наиболее быстрые и при этом наиболее простые в реализации согласно работе [13], алгоритмы построения триангуляции Делоне.

Итеративные алгоритмы

Все итеративные алгоритмы имеют в своей основе идею последовательного добавления точек в частично построенную триангуляцию Делоне. Формально этот процесс может быть описан так.

Пусть имеется триангуляция Делоне на множестве из (n-1) точек. Очередная n-я точка добавляется в уже построенную структуру триангуляции следующим образом.

Шаг 1. Вначале производится локализация точки, т.е. находится треугольник (построенный ранее), в который попадает очередная точка. Если точка не попадает внутрь триангуляции, то находится треугольник на границе триангуляции, ближайший к очередной точке.

Шаг 2. Если точка попала на ранее вставленный узел триангуляции, то такая точка обычно отбрасывается, иначе точка вставляется в триангуляцию в виде нового узла. При этом, если точка попала на некоторое ребро, то оно разбивается на два новых, а оба смежных с ребром треугольника также делятся на два меньших. Если точка попала строго внутрь какого-нибудь треугольника, он разбивается на три новых. Если точка попала вне триангуляции, то строится один или более треугольников. Затем проводятся локальные проверки вновь полученных треугольников на соответствие условию Делоне, и выполняются необходимые построения.

Сложность данного алгоритма складывается из трудоемкости поиска треугольника, в который на очередном шаге добавляется точка, трудоемкости построения новых треугольников, а также трудоемкости соответствующих перестроений структуры триангуляции в результате неудовлетворительных проверок пар соседних треугольников полученной триангуляции на выполнение условии Делоне.

При построении новых треугольников возможны две ситуации: добавляемая точка попадает либо внутрь триангуляции, либо вне ее. В первом случае строятся новые треугольники, и число выполняемых алгоритмом действий фиксировано. Во втором - необходимо построение дополнительных внешних к текущей триангуляции треугольников, причем их количество может в худшем случае равняться $n-3$. Однако за все шаги работы алгоритма будет добавлено не более $3N$ треугольников, где N - общее число исходных точек. Поэтому в обоих случаях общее затрачиваемое время на построение треугольников составляет $O(N)$.

Любое добавление новой точки в триангуляцию теоретически может нарушить целостность условия Делоне, поэтому после добавления точки обычно сразу же производится локальная проверка триангуляции на условие Делоне. Эта проверка должна охватить все вновь построенные треугольники и соседние с ними. Количество таких перестроений в худшем случае может привести к полному перестроению всей триангуляции, поэтому трудоемкость перестроений составляет $O(N)$. Однако среднее число таких перестроений па реальных данных составляет только около трех [15].

Таким образом, наибольший вклад в трудоемкость итеративного алгоритма дает процедура поиска очередного треугольника. Именно поэтому все итеративные алгоритмы построения триангуляции Делоне отличаются друг от друга главным образом только процедурой поиска очередного треугольника.

Простой итеративный алгоритм

В простом итеративном алгоритме поиск очередного треугольника реализуется следующим образом. Берется любой треугольник, уже принадлежащий триангуляции (например, выбирается случайно), и последовательными переходами по связанным треугольникам ищется искомый треугольник.

При этом в худшем случае приходится пересекать все треугольники триангуляции, поэтому трудоемкость такого поиска составляет $O(N)$. Однако в среднем для равномерного распределения в квадрате нужно совершить только $O(\sqrt{N})$ операций перехода [64]. Таким образом, трудоемкость простейшего итеративного алгоритма составляет в худшем случае $O(N^2)$, а в среднем - $O(N^{3/2})$ [46].

Во многих практически важных случаях исходные точки не являются статистически независимыми, при этом n-я точка находится вблизи $(n+1)$-й. Поэтому в качестве начального треугольника для поиска можно брать треугольник, найденный ранее для предыдущей точки. Тем самым иногда удается достичь на некоторых видах исходных данных трудоемкости построения триангуляции в среднем $O(N)$.

Алгоритмы с кэшированием поиска треугольников

При реализации этих алгоритмов строится кэш - специальная структура, позволяющая за время (1) находить некоторый треугольник, близкий к искомому. При этом, измененные треугольники из кэша не удаляются (предполагается, что каждый удаленный треугольник как запись в памяти компьютера превращается в новый треугольник и поэтому допустимость ссылок на треугольники не нарушается при работе алгоритма), один и тот же треугольник может многократно находиться в кэше, а некоторые треугольники могут вообще там отсутствовать [15].

Основная идея кэширования заключается в построении некоторого более простого планарного разбиения плоскости, чем триангуляция, в котором можно быстро выполнять локализацию точек. Для каждого элемента простого разбиения делается ссылка на треугольник триангуляции. Процедура поиска сводится к локализации элемента простого разбиения, переходу по ссылке к треугольнику и последующей локализации искомого треугольника алгоритмом из простого итеративного алгоритма триангуляции Делоне. В качестве такого разбиения проще всего использовать регулярную сеть квадратов (Рис. 3). Например, если данное планарное разбиение полностью покрывается квадратом $[0;1][0;1]$, то его можно разбить на m^2 равных квадратов. Занумеруем их всех естественным образом двумя параметрами $i,j = 0..\overline{(m-1)}$. Тогда по данной точке (x,y) мы можем найти квадрат $[x/m]$, $[y/m]$, где [...] - знак взятия целой части числа.

Кэш в виде регулярной сети квадратов наиболее хорошо работает для равномерного распределения исходных точек и распределений, не имеющих высоких пиков в функции плотности. В случае же если заранее известен характер распределения, можно выбрать какое-то иное разбиение плоскости (например, в виде неравномерно отстоящих вертикальных и горизонтальных прямых).

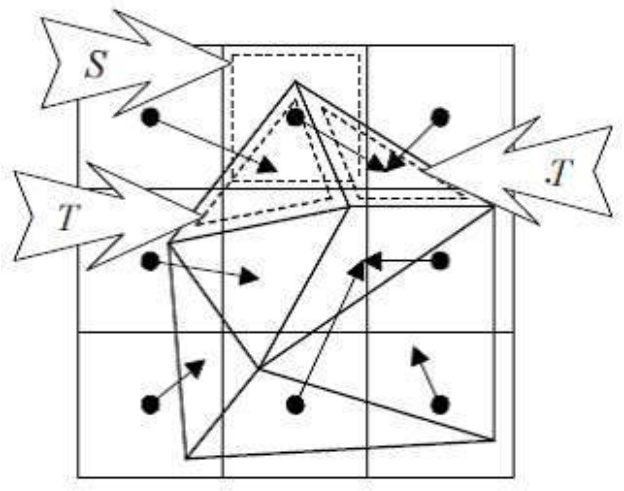

Рис. 3. Локализация точек в кэше (S - найденный квадрат, - связанный с квадратом треугольник, $'$ - конечный треугольник).

Итеративный алгоритм со статическим кэшированием поиска

В алгоритме триангуляции со статическим котированием поиска необходимо выбрать число m и завести кэш в виде двумерного массива r размером $m \times m$ для ссылок на треугольники [15]. Первоначально этот массив надо заполнить ссылками на самый первый построенный треугольник. Затем, после выполнения очередного поиска, в котором был найден некоторый треугольник , начиная поиск с квадрата (i, j), необходимо обновить информацию в кэше: r_{ij} := ссылка на T. Размер статического кэша следует выбирать по формуле $m = s \times N^{3/8}$, где s - коэффициент статического кэша. На практике значение s следует взять $\approx 0,6 - 0,9$.

Первое время, пока кэш не обновится полностью, поиск может идти довольно долго, по потом скорость повышается. Этого недостатка лишен следующий алгоритм.

Итеративный алгоритм с динамическим кэшированием поиска

В алгоритме триангуляции с динамическим кэшированием поиска необходимо завести кэш минимального размера, например 2×2. По мере роста числа добавленных в триангуляцию точек необходимо последовательно увеличивать его размер в два раза, переписывая при этом информацию из старого кэша в новый. При этом для увеличения кэша надо выполнить следующие пересылки (r - старый кэш, r' - новый): $\forall i, j = \overline{0, m-1}$: $r'_{2i,2j}, r'_{2i,2j+1}, r'_{2i+1,2j}, r'_{2i+1,2j+1} := r_{i,j}$.

Данный алгоритм кэширования позволяет одинаково эффективно работать на малом и большом количестве точек, заранее не зная их числа.

Увеличение размера динамического кэша в два раза следует производить каждый раз при достижении числа точек в триангуляции $n = r \times m^2$, где r - коэффициент роста динамического кэша, а m - текущий размер кэша. На практике значение коэффициента роста динамического кэша следует выбрать $\approx 3 - 8$.

Трудоемкости алгоритмов триангуляции с кэшированием как и всех итеративных алгоритмов составляют в худшем случае $O(N^2)$, а в среднем на равномерном распределении для статического кэширования - $O(N^{9/8})$ и для динамического кэширования - $O(N)$ [15].

Для большинства случайных распределений исходных точек данный алгоритм работает значительно быстрее всех остальных алгоритмов [15]. Однако на некоторых реальных данных, в которых последовательные исходные точки находятся вблизи друг друга (например, точки изолиний карт рельефа), алгоритм динамического кэширования может тратить большее время, чем другие алгоритмы. Для учета такой ситуации в алгоритм следует добавить дополнительную проверку. Если очередная добавляемая точка находится от предыдущей точки на расстоянии большем, чем некоторое число $\triangle$ (порядка текущего размера клетки кэша), то поиск необходимо начать с треугольника из кэша, иначе нужно начать с последнего построенного треугольника. В такой модификации алгоритм динамического кэширования становится непревзойденным по скорости работы на большинстве реальных данных.

Итеративный алгоритм с послойным сгущением

В итеративном алгоритме триангуляции с послойным сгущением [19] необходимо разбить плоскость с точками на $n = (2^u + 1) \times (2^v + 1)$ элементарных ячеек-квадратов одинакового размера. Каждый квадрат нумеруется от 0 до 2^u по горизонтали и от 0 до 2^v по вертикали. Далее вводится понятие слоя. Считается, что точка принадлежит слою i, если оба номера ее квадрата кратны 2^i (тогда все исходные точки образуют слой 0, слой $i + 1$ будет подмножеством слоя i, а максимальный номер слоя $k = min(u, v)$). По значениям пар номеров все точки слоя i делятся на четыре подмножества:

1. угловые точки (оба их номера кратны $2i + 1$) - это слой $i + 1$;
2. внутренние точки (оба их номера не кратны $2i+$ $+ 1$);
3. X-граничные точки (только номер по координате X кратен $2i + 1$);
4. Y-граничные точки (только номер по координате У кратен $2i + 1$).

Добавление точек в триангуляцию надо производить послойно от слоя с максимальным номером до нулевого. Внутри слоя нужно вначале внести все точки второго типа, затем третьего и в конце - четвертого.

На рисунке 4 приведен пример разбиения плоскости на квадраты при $u = 3$, $v = 2$ и $n = 9x{\times}5$ по этапам:

(а) все точки слоя 1;
(б) внутренние точки слоя 0;
(в) X-граничные точки слоя 0;
(г) Y-граничные точки слоя 0.

На рисунке 4 числа (от 1 и больше) определяют порядок выбора квадратов (и соответственно точек внутри них) на каждом этапе ранее обработанные квадраты затемнены.

Для произвольных наборов точек такой алгоритм (как и все другие итеративные алгоритмы) имеет трудоемкость (N^2). Если исходные точки имеют равномерное распределение, то трудоемкость алгоритма в среднем случае будет $O(N)$. Кроме того, в [19] показывается, что если исходные точки удовлетворяют некоторым фиксированным (не вероятностным) ограничениям, то трудоемкость алгоритма будет и в худшем случае $O(N)$.

Преимуществом данного алгоритма является ещё и то, что равномерно последовательно вставляя в триангуляцию узлы, удается избавиться от ситуаций построения длинных узких треугольников, которые в дальнейшем перестраиваются. За счет этого данный алгоритм быстрее выполняется на реальных данных, чем многие другие алгоритмы.

Ячеечные алгоритмы триангуляции

Алгоритм «марширующие кубы»

Самым ранним ячеечным алгоритмом построения триангуляции поверхности является алгоритм "марширующие кубы представленный Лоренсеном [52] в 1987 году. Этот алгоритм производит разбиение области пространства, содержащей исходную поверхность, на кубические ячейки и аппроксимирует пересечение исходной поверхности и каждой кубической ячейки разбиения треугольниками. Для случая синтеза изображения по плоским сечениям вершинами каждого куба будут по четыре точки на паре соседних сечений (на каждом сечении вершины образуют квадрат), расположенные как бы друг над другом. Алгоритм, предложенный Лоренсеном, можно разоить на два этапа:

1. Разбиение области G пространства R3 на конечное множество ячеек, поиск ячеек, пересекаемых искомой поверхностью.

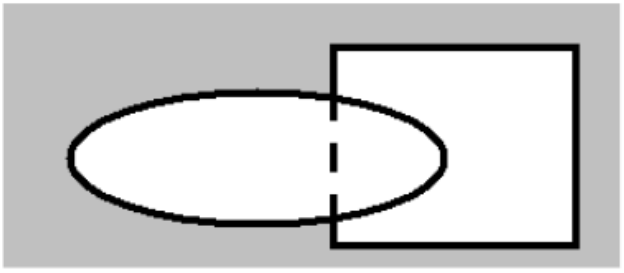

Рис. 5. На схеме квадрат обозначает ячейку, овал - некий изгиб поверхности.

2. Аппроксимация поверхности в найденных ячейках.

Две эти подзадачи являются независимыми. Рассмотрим их подробнее.

Первый этап.

Основные проблемы этого этапа заключается в следующем:

1. Разбить область G на ячейки.
2. Выбрать ячейки, которые пересекаются с искомой поверхностью.

После того как область G будет разбита на ячейки, значения функции, в общем случае, задающей поверхность будут известны только в вершинах этих ячеек. Таким образом, ячейка является главной структурной единицей во всех алгоритмах на этом этапе. В тех задачах, в которых функция, задающая поверхность задана таблицей на регулярной сетке, проблема разбиения области G на ячейки сразу отпадает, ввиду однозначности её решения - ячейка должна быть параллелепипедом - для того, чтобы знать значения функции в вершинах ячейки. Если же функция задана явно, то ячейку можно выбрать произвольной формы и размера. Однако следует учесть некоторые проблемы связанные с аппроксимацией искомой поверхности в ячейке. Если размер ячейки будет очень большим, то возможна большая потеря точности.

Как видно из рисунка 5, при большом размере ячейки некоторые части искомой поверхности просто не будут видны. Однако выбирать ячейки очень маленького размера не очень хорошо с точки зрения быстродействия. Поэтому размер ячейки надо выбирать не меньше допустимой погрешности построения искомой поверхности.

Форма ячейки в алгоритме «Марширующие кубы» - параллелепипед. Однако это не единственно возможный вариант. Форма ячейки определяет дальнейшую триангуляцию ячейки. Пусть форма ячейки - многогранник с N вершинами, тогда сопоставим каждой ячейке N - битовый индекс, а каждой

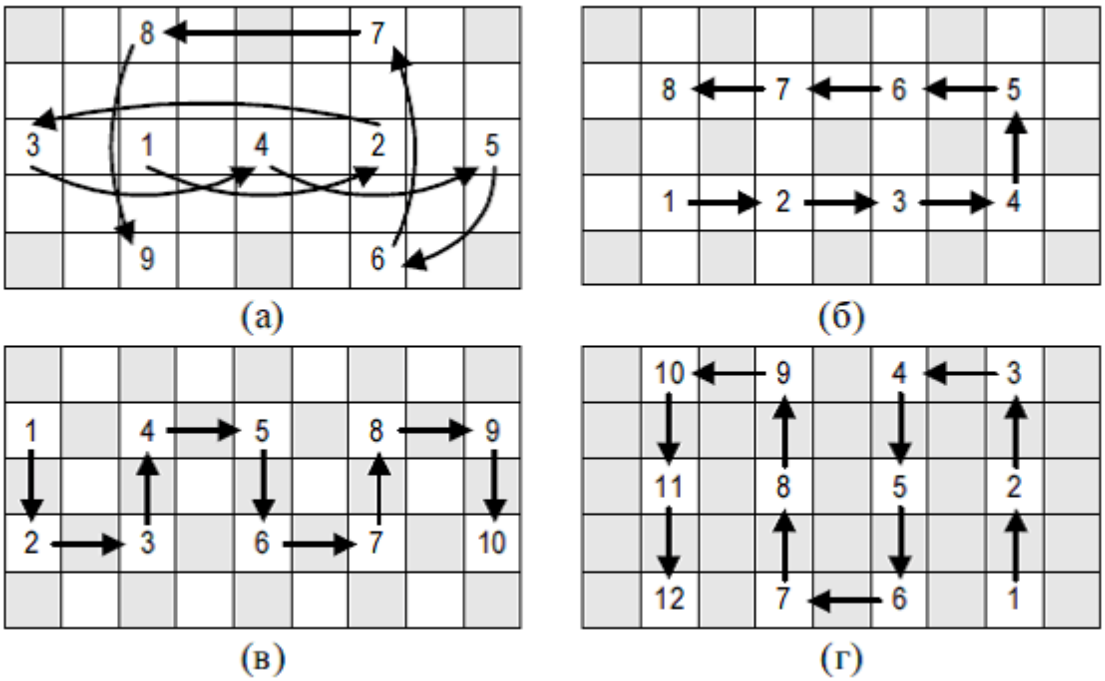

Рис. 4. Порядок выбора точек в итеративном алгоритме с послойным сгущением.

вершине - один бит в индексе. Причем, если вершина ячейки находится вне объема ограниченного искомой поверхностью, то значение этого бита «0», иначе - «1». Тогда количество разных типов триангуляции будет $2N$. Отсюда видно, что использовать в качестве ячейки, например, икосаэдр не оптимально. Многогранник с наименьшим количеством вершин - треугольная пирамида. Именно она используется в качестве ячейки в алгоритмах Канейро, MT6, Скалы.

Итак, допустим, что область G уже разбита на ячейки. Тогда главной проблемой становится поиск ячеек пересекаемых искомой поверхностью. Пусть - множество ячеек, тогда C_v - множество ячеек пересекаемых поверхностью $F(P) = v$. Тогда можно считать, что поверхность пересекает ячейку, если существуют такие $_1$, и $_2$ - вершины ячейки, что

$$F(p_1) < v < F(p_2) \tag{2}$$

Это условие выполняется, если справедливо неравенство:

$$\min_i F(p_i) < v < \max_i F(p_i) \tag{3}$$

Где p_i, p_j - вершины ячейки.

Таким образом, проблема свелась к следующему: из множества ячеек выбрать подмножество C_v ячеек, удовлетворяющих условию (3).

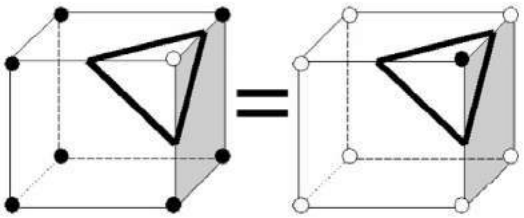

Рис. 6. Способы триангуляции.

Рассмотрим далее проблему аппроксимации поверхности в ячейке.

Второй этап.

Как уже было сказано, пространство разбивается на ячейки, и отбираются только те ячейки, в которых надо производить аппроксимацию. Таким образом, задачей второго этапа является аппроксимация поверхности в одной ячейке. Наиболее оптимальный способ аппроксимации - триангуляция. Посчитаем, сколько способов триангуляции имеет параллелепипед. Пусть имеется 8-битовый индекс. Тогда сопоставим каждой вершине один бит в индексе. Причем, если вершина ячейки находится вне объема ограниченного искомой поверхностью, то значение этого бита «0», иначе - «1». Тогда количество разных типов триангуляции будет $2^8 = 256$. Однако из рисунка 6 видно, что способ триангуляции с индексом (i) совпадает со способом триангуляции с индексом $(i \neq j)$.

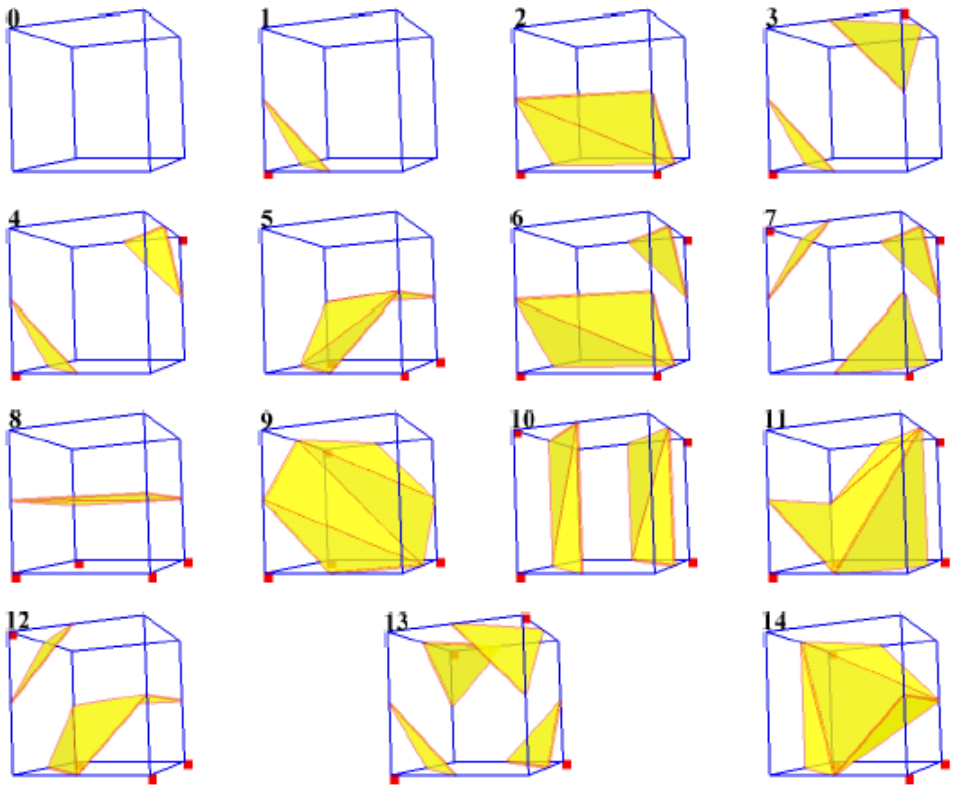

Рис. 7. Способы триангуляции.

Итого получается 128 различных способов триангуляции. Однако, используя симметрию и вращение эти 128 способов можно свести к 15 (рисунок 7).

Получив способ триангуляции, можно уже аппроксимировать поверхность в ячейке. К этому моменту уже известно количество треугольников, а для каждого треугольника известны ребра ячеек, на которых лежат его вершины. Остается найти точку на ребре ячейки, в которой поверхность ее пересекает. В случае явно заданной функции точку можно с большой точностью найти методами поиска корня, а в случае заданной таблично функции искомая точка находится с помощью линейной интерполяции двух вершин.

Алгоритм Канейро

Алгоритм, предложенный Канейро [25, 24] (также называемый "марширующие тетраэдры 5"[3]), основанный на разбиении пространства на треугольные пирамиды, как и алгоритм "Марширующих кубов состоит из двух этапов:

1. Разбиение пространства на конечное множество ячеек, затем поиск ячеек пересекаемых искомой поверхностью;
2. Аппроксимация поверхности в найденных ячейках.

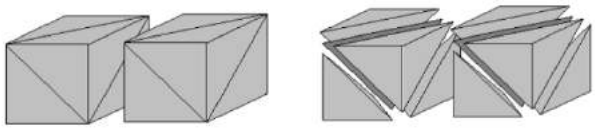

Рис. 8. Разбиение параллелепипеда на треугольные пирамиды.

Первый этап

Как уже было сказано, алгоритм использует в качестве ячеек треугольные пирамиды. Для этого пространство разбивается на параллелепипеды в соответствии с сеткой, на которой задана функция, а затем каждый параллелепипед разбивается на треугольные пирамиды. Такой же подход применяется в алгоритмах Скалы. Разбиение параллелепипеда на треугольные пирамиды по методу Канейро показано на рисунке 8.

Однако при подобном разбиении «швы» «разрезов» не совпадают. Другими словами, стороны треугольников, полученных в результате триангуляции соседних ячеек, не будут совпадать, что повлечет за собой появление «дырок». Для решения этой проблемы предлагается разбивать параллелепипеды в «шахматном порядке» - по очереди меняя

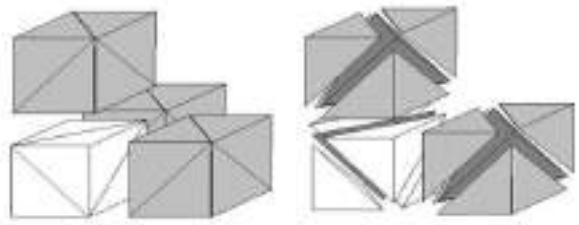

Рис. 9. Разбиение параллелепипеда на треугольные пирамиды.

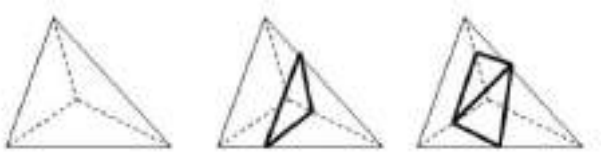

Рис. 10. Способы триангуляции треугольной пирамиды.

шаблон разбиения: с показанного на рисунке 8 на зеркальный, как показано на рисунке 9.

Второй этап

Задача второго этапа - аппроксимация поверхности в ячейке. Для алгоритмов Канейро, Скалы, второй этап один и тот же - производится триангуляция треугольной пирамиды в соответствии со значениями функции в вершинах.

Подсчитаем, число способов триангуляции треугольной пирамиды. Пусть имеется 4-битовый индекс. Тогда сопоставим каждой вершине один бит в индексе, таким же образом, как и для параллелепипеда. Тогда количество разных типов триангуляции будет $2^4 = 16$. Однако, используя симметрию и вращение, число способов можно свести к 3 (рисунок 10).

Алгоритм Скалы

Алгоритм Скалы [65] был разработан для визуализации трехмерных скалярных полей, заданных с помощью функции, определенной в каждой точке пространства. Однако, метод разбиения пространства на ячейки таков, что дает возможность использовать этот алгоритм для визуализации скалярных полей заданных на регулярной сетке.

Идея алгоритма заключается в том, чтобы строить тетраэдры не в ячейке, а на стыке двух ячеек. Для каждой ячейки добавляется дополнительная вершина - её центр.

Для разбиения пространства на ячейки метод Скалы использует узлы регулярной сетки, находящиеся в вершинах параллелепипеда, полученного тем же способом, что и в предыдущих рассмотренных методах, и дополнительную точку, находящуюся

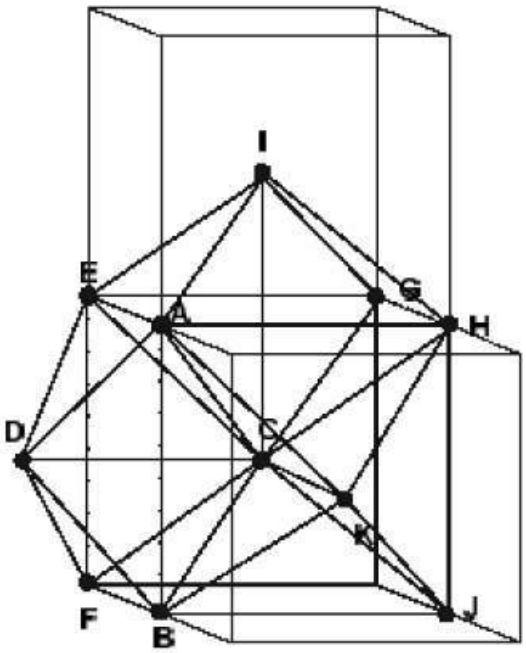

Рис. 11. Построение ячейки у параллелепипеда.

на пересечении диагоналей этого параллелепипеда. Значение функции в этой точке предлагается считать как линейную интерполяцию значений функции в вершинах параллелепипеда. Для каждого параллелепипеда, полученного из узлов регулярной сетки, строится ячейка способом, показанным на рисунке 11. При таком разбиении для каждой ячейки используются «срединные» точки «соседних» параллелепипедов. На рисунке 11 это точки I, K, D. Итог этого разбиения - 12 треугольных пирамид (DEAC, DABC, DFBC, DFEC, IEAC, IAHC, IGHC, IEGC, KAHC, KHJC, KJBC, KABC).

Алгоритм «Марширующие тетраэдры 6»

Алгоритм «Марширующие тетраэдры 6» был предложен Гуезеком [38] как альтернатива алгоритму Канейро. Основное отличие этих двух методов в том, что для алгоритма «MT6» отпадает необходимость смены шаблонов с прямого на зеркальный и обратно. Это достигается путем симметричного разбиения параллелепипеда на 6 тетраэдов, показанного на рисунке 12.

Сравнительный анализ вышеописанных четырех алгоритмов проводится, например, в работе [11].

Сравнение можно проводить по 4 критериям:

— Скорость работы;
— Ошибка аппроксимации;
— Количество сгенерированных треугольников;

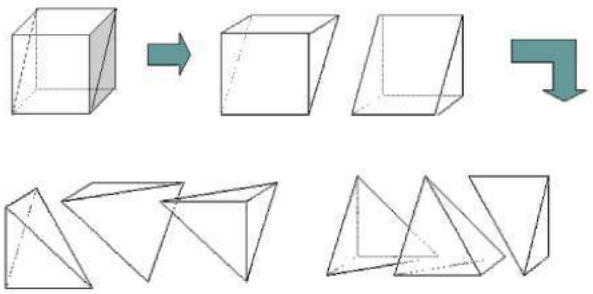

Рис. 12. Разбиение параллелепипеда на 6 тетраэдров.

— «Качество» сгенерированных треугольников.

При этом, рассмотренные алгоритмы ячеистого типа имеют одну и ту же основу. Поэтому скорость и ошибка аппроксимации у них различаются несущественно. Сложность этих алгоритмов составляет $O(N)$. Поэтому их достаточно сравнить по двум параметрам: количество треугольников и их "качество".

Для сравнения «качества» треугольников в работе [11] вводится некий параметр - «мера правильности треугольника» - отношение меньшей стороны треугольника к большей стороне. Таким образом, мера правильности треугольника может принимать значения от нуля до единицы (для равностороннего или правильного треугольника). Чем «компактнее» треугольник - тем правильнее освещение (рис. 13). Если учесть тот факт, что большое количество треугольников невыгодно, то получается, что «идеальный» треугольник - тот, у которого максимальная площадь и минимальный периметр. Это равносторонний треугольник. Таким образом, мера правильности треугольника обуславливает корректность освещения.

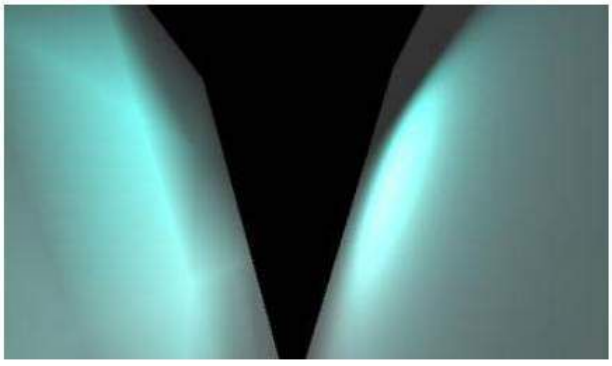

Рис. 13. Слева поверхность состоит из 32 треугольников, справа - из 20000.

Результаты испытаний этих алгоритмов показывают:

— Алгоритм Скала генерирует неоправданно большое количество треугольников;

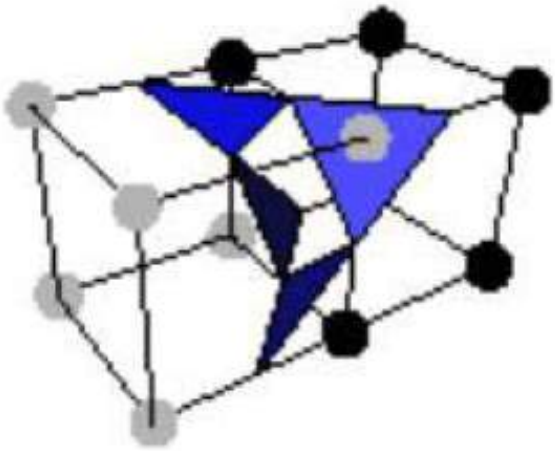

Рис. 14. Топологические неточности, возникающие из-за большого количества шаблонов триангуляции.

— Алгоритм MT6 показывает результаты хуже, чем алгоритм Канейро (количество треугольников, генерируемых алгоритмом MT6 больше, чем количество треугольников, генерируемых алгоритмом Канейро, а среднее качество хуже);
— Алгоритм "Марширующие кубы"генерирует значительно меньшее количество треугольников, чем другие алгоритмы.

Таким образом, можно сказать, что из рассмотренных алгоритмов алгоритм "Марширующие кубы"имеет лучшие показатели качества по выбранным критериям.

Однако, Блюменталь (Bloomenthal) в работе [23] показал существование топологических неточностей алгоритма "марширующие кубы"для граней куба (face ambiguity) у построенных поверхностей и предложен метод решения данной проблемы разбиением ячейки на 12 тетраэдров.

Дёрст (Durst) в своей работе [33] показал также существование внутренних топологических неточностей (internal ambiguity). Из-за большого количества шаблонов триангуляции может получиться ситуация как на рисунке 14.

Ещё один вариант неточностей показан в работе [58]. К примеру, оба фрагмента поверхностей на рисунке 15 при триангуляции алгоритмом "Марширующие кубы"дадут шаблон 4 (см. рисунок 7).

Решения этих проблем основаны на изменении шаблонов триангуляции и попытке «предугадать» (отсюда пошло название - asymptotic decider) как на самом деле себя ведет поверхность в ячейке. Однако такой подход в несколько раз увеличивает количество треугольников, генерируемых алгоритмом «Марширующие кубы». К тому же, такой алгоритм уже нельзя считать интерактивным.

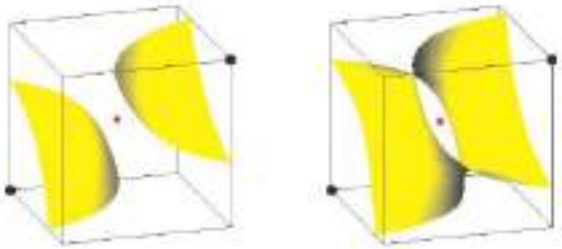

Рис. 15. Два варианта различных топологий, дающих шаблон 4 алгоритма «марширующие кубы»

Нильсон и Хамман [58] предложили метод решения топологических неточностей для граней куба, учитывая билинейность функции на грани куба как следствие трилинейности в кубе. Черняев в работе [26] предложил систематизированный метод решения топологических проблем и вывел дополненную таблицу для случаев метода марширующие кубы. Для решения внутренней проблемы использовалось свойство билинейности аппроксимирующей функции на сечении куба, параллельном его грани. В методе для решения возникает необходимость решения квадратного уравнения. Для некоторых сложных случаев пересечения поверхности и куба построение набора треугольников требует использование дополнительной вершины (центра куба). Метод строит топологически связную треугольную сеть. Метод Черняева был программно реализован Льюинером [48] в 2003 году. Для реализации метода использовалась таблица из 730 случаев, вместо оригинальных 256 для алгоритма «марширующие кубы».

Однако, несмотря на существование топологической неточности в генерируемой поверхности, алгоритм «Марширующие кубы» широко используется на практике, т.к. вероятность проявления ошибок такого рода достаточно мала.

Алгоритмы, разбивающие пространство на тетраэдры и аппроксимирующие поверхность внутри каждого тетраэдра и были разработаны в противоположность алгоритму «марширующие кубы», так как при пересечении поверхности и тетраэдра топологических неточностей не возникает. Однако их недостатком заметно более мелкая триангуляция и, следовательно, большое количество треугольников. Для решения этой проблемы в работе [3] были разработаны ещё три алгоритма разбиения, на базе алгоритмов «MT5» и «MT6», генерирующие тетраэдры лучшего качества, чем оригинальные. Эти алгоритмы рассмотрены ниже.

Алгоритм «марширующие призмы»

Область пространства заполняется призмами с правильными треугольниками в их основании [4]. Возможно 64 случая пересечения призмы и поверхности, которые хранят в специальной таблице. Эти 64 случая можно свести к 9 при помощи поворотов и симметрии относительно плоскости, параллельной основанию призмы и проходящей через середину вертикальных ребер, а также с учетом возможности обращения знака функции. На рисунке 16 показаны все эти 9 случаев. Отмечены вершины, на которых функция имеет положительные значения, на других вершинах значение отрицательное.

Триангуляция пятивершинными пирамидами

Область пространства разбивается на кубические ячейки [3]. Каждая ячейка разбивается на 6 пятивершинных пирамид: вершинами пирамид являются центр ячейки и по 4 вершины с каждой из 6 граней куба. Далее, строятся треугольники на пересечении пирамиды и поверхности. Возможны 32 варианта пересечения пирамиды и поверхности, которые можно свести к 6 неэквивалентным, т.к. у пирамиды 5 вершин. На рисунке 17 представлены 6 неэквивалентных вариантов. Отмечены вершины, на которых исходная функция принимает положительные значения. Для программной реализации данного способы триангуляции используется нумерация ребер как в методе Скалы и таблица из 32 случаев пересечения пирамиды и поверхности.

Алгоритм «Марширующие октаэдры»

Область пространства заполняется октаэдрами [3]. Заполнение можно производить следующим способом: область пространства заполняется кубами и строятся ребра от центров кубов к их вершинам. В итоге получается сеть октаэдров, у которых восемь ребер - это построенные ребра для каждой грани куба, с использованием его центра и центра смежного с ним куба. На рисунке 18 представлены варианты пересечения поверхности и октаэдра.

После заполнения области пространства октаэдрами производится аппроксимация треугольниками пересечения поверхности и каждого октаэдра.

Заполнение пространства шаблоном октаэдров с дополнительными 3-мя диагоналями аналогично заполнению пространства по Скале за исключением того, здесь добавляются еще 9 ребер (6 диагоналей на 3-х гранях и 3 отрезка, соединяющие центр куба с тремя соседними). В итоге в узле будет 20 ребер, в отличие от 11 из алгоритма Скалы.

Данный алгоритм обладает рядом преимуществ перед алгоритмом Скала, который дополнительно разбивает каждый октаэдр на четыре тетраэдра:

— построение сети треугольников для каждого октаэдра использует одно обращение к таблице

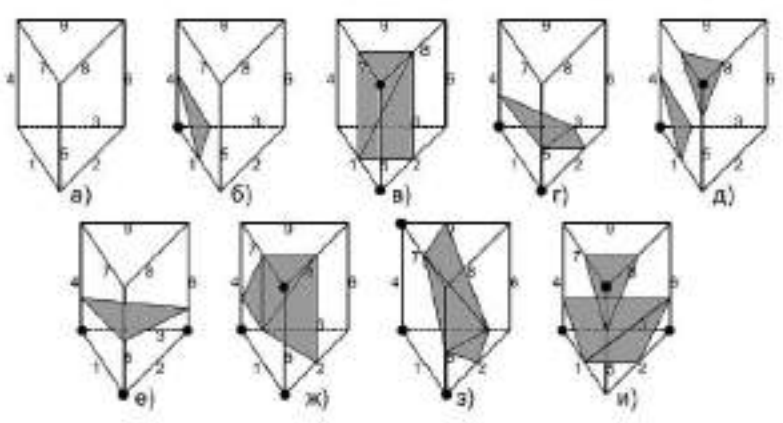

Рис. 16. Возможные случаи пересечения призмы и поверхности.

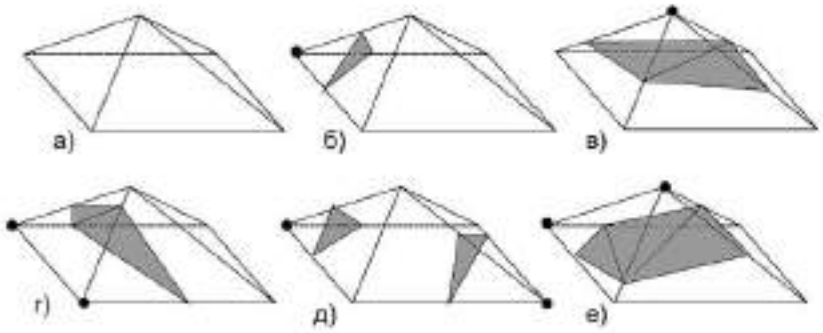

Рис. 17. Варианты пересечения пирамиды и поверхности.

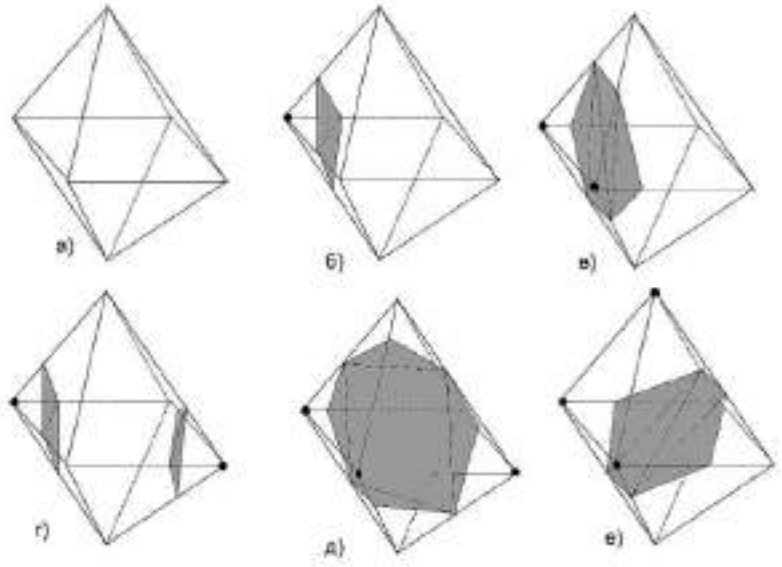

Рис. 18. Варианты пересечения пирамиды и поверхности.

случаев вместо четырех (т.к. в алгоритме Скала используется четыре тетраэдра);
— можно построить сеть, оптимальную по определенному параметру.

В работе [3] показано, что получаемая алгоритмом сеть треугольников лучше а её построение производится в среднем в 1,5 раза быстрее, чем по алгоритму Скала.

Триангуляция сверхбольших поверхностей

Как уже упоминалось и показывалось, существенной проблемой ячеечных алгоритмов является их индифферентность, то есть, невозможность всегда правильно визуализировать поведение поверхности внутри ячейки. Уменьшение размера ячейки может решить эту проблему, однако тогда возникает другая - нехватка памяти. Поверхность, исходные данные которой не умещаются в оперативной памяти компьютера, называют сверхбольшой [8]. Если поверхность не умещается в оперативную память, то в видеопамять он не поместится тем более. Поэтому визуализация такой поверхности обычными способами займет очень много времени.

Скорость вывода поверхности на экран прямо пропорциональна числу выводимых треугольников. В действительности, как правило, в область видимости попадает относительно небольшая часть поверхности. Но при использовании обычных алгоритмов построения триангуляции, например, триангуляции Делоне, скорость их работы не пригодна для использования при визуализации в реальном режиме времени, так как приходится несколько раз в секунду перестраивать довольно большую часть триангуляции. Поэтому, обычные алгоритмы построения триангуляции Делоне как метод моделирования и визуализации поверхности не пригодны для больших объёмов исходных данных.

Одной из первых попыток работы с большими триангуляциями можно считать алгоритмы построения упрощённых триангуляций [14].

Задача построения упрощенных триангуляций также является NP-сложной [14]. Поэтому на практике используются приближенные алгоритмы, которые можно разделить в соответствии с используемой стратегией на два основных класса, работающих «сверху вниз» и «снизу вверх» [35].

Стратегия «сверху вниз» начинает работу с простой аппроксимирующей модели, состоящей из одного или нескольких треугольников, покрывающих исходную триангуляцию. Далее в триангуляцию последовательно добавляются новые точки до тех

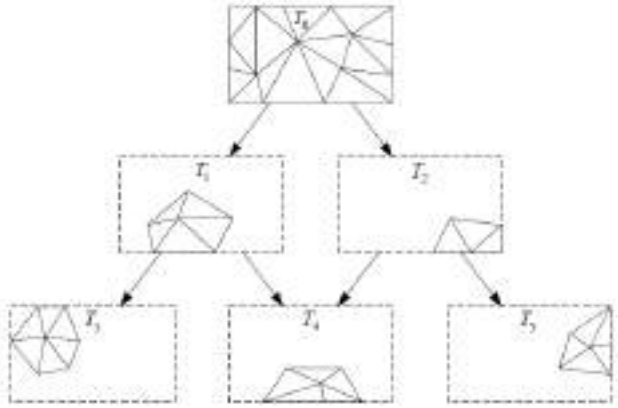

Рис. 19. Граф мультитриангуляции.

пор, пока не будет достигнуто требуемое разрешение. Одним из таких алгоритмов является алгоритм "Селектор Делоне описанный в [35].

В стратегии «снизу вверх» работа начинается с исходной триангуляции, и число элементов триангуляции постепенно уменьшается до тех пор, пока не будет достигнуто требуемого количества узлов либо не будет достигнуто заданного допустимого отклонения упрощенной триангуляции от исходной.

Из двух стратегий «сверху вниз» и «снизу вверх» следует отметить, что первая из них, как правило, работает точнее при одинаковом наборе изменяющих операций (удаление вставка узлов). Однако последняя стратегия в среднем работает быстрее и в ней можно использовать другие операции (например, коллапс рёбер и треугольников), что также может в ряде случаев повысить качество работы [47]. Кроме того, заметим, что для реализации первой стратегии достаточно процедур, предоставляемых любым итеративным алгоритм триангуляции, в то же время для второй необходимы дополнительные алгоритмы для удаления точек, коллапса рёбер и треугольников, которые имеют свои сложности в реализации.

Стоит также отметить, что использование алгоритмов упрощения изначально подразумевает необходимость нахождения некоторого компромисса между быстродействием обработки поверхности и качеством получаемых результатов. Решение данной проблемы основано на построении и использовании особой структуры - мультитриангуляции (MT) [7, 8], которую можно определить как особую структуру, представляющую собой набор фрагментов триангуляции, образующих ориентированный граф без циклов (рисунок 19).

Мультитриангуляция позволяет представлять триангуляцию с различной степенью детализации, в зависимости от предложенного критерия. Поэтому, в зоне, представляющей наибольший интерес с по-

зиции критерия, детализация триангуляции максимальна, а в оставшихся областях понижена. Обычно зона интереса является небольшой по сравнению со всей триангуляцией, поэтому за счёт снижения разрешения в большей части триангуляции удаётся достичь нужного количества примитивов без значительной потери качества. Более того, структура мультитриангуляции предоставляет эффективный способ перестроения триангуляции в связи со сменой критерия, что делает возможным динамическую визуализацию в реальном режиме времени.

На базе МТ разработаны алгоритмы, позволяющие строить триангуляцию переменного разрешения произвольных поверхностей [40]. Реализация одного из таких алгоритмов существует в DirectX 8.0 API (Progressive Mesh) и поэтому имеет широкое распространение в графических системах, работающих с помощью DirectX. Однако все разработанные алгоритмы работают, если вся структура МТ находится в ОЗУ компьютера, а для решения практически важных задач часто необходимо отобразить поверхность, большую, чем та, что умещается в оперативной памяти.

Основная идея построения структуры МТ для работы с большими объемами входных данных заключается в том, что в памяти можно держать только те фрагменты, которые вероятнее всего будут использованы. т.е. те, которые войдут в результирующий список треугольников для вывода на экран. То есть, структура МТ должна позволять в реальном режиме времени загружать нужные фрагменты и выгружать ненужные.

Применяя алгоритмы построения МТ [29], структура МТ получается таковой, что при детализации одного треугольника может возникнуть необходимость детализировать еще некоторый ряд произвольных треугольников. Это значит, что для работы алгоритма выбора результирующего списка треугольников необходимым условием будет наличие в памяти всей структуры МТ. Поэтому для работы алгоритма на больших объемах входных данных структура графа МТ должна быть изменена. Чтобы уменьшить взаимозависимости треугольников, нужно попытаться упорядочить граф МТ. В работах [7, 8] предлагается вариант упорядочения, при котором граф МТ должен состоять из нескольких независимых подграфов, что позволит оперативно работать с каждым из них. Для этого предлагается следующий алгоритм построения МТ:

— *Шаг 1.* Все множество исходных точек триангуляции разбивается на части, каждая из которых представляет собой некоторый прямоугольник.
— *Шаг 2.* Внутри каждого прямоугольника осуществляется построение МТ алгоритмом, описанным в [25].

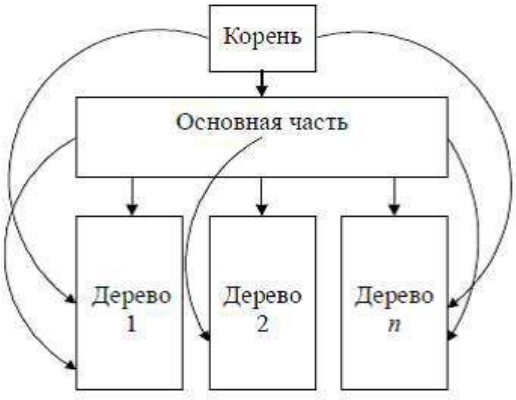

Рис. 20. Структура мультитриангуляции.

— *Шаг 3.* Оставшаяся триангуляция упрощается до самого грубого фрагмента.

На выходе алгоритма получается структура, схематично изображенная на рисунке 20.

Как видно из рисунка 20, получается несколько частей: «деревья МТ» и основная часть (по сути, дерево 0), которые можно загружать независимо друг от друга.

Возникает вопрос, на какие прямоугольники разделить исходное множество точек, так как это существенно влияет на скорость и корректность работы алгоритма. Число прямоугольников определяет число деревьев в МТ, если это число слишком мало, то возможно появление областей с недостаточной детализацией, а если слишком велико - объем памяти, занимаемый МТ будет необоснованно большим.

Хорошего соотношения между качеством визуализации и объемом занимаемой памяти МТ можно добиться при использовании так называемой сбалансированной структуры, т.е. разделенной МТ, в которой число вершин во всех деревьях и в основной части одинаково. Алгоритм построения сбалансированного дерева описан в [8].

Чтобы была возможность выгружать фрагменты, когда они не нужны, и загружать, когда они нужны, необходимо сохранять структура МТ на жесткий диск, а затем загружать её. При работе со сверхбольшими МТ необходимо загружать структуру МТ частично: лишь те части, которые вероятнее всего будут удовлетворять критерию выбора треугольников. Например, деревья МТ, которые находятся вблизи наблюдателя, необходимо загружать с максимальной степенью детализации, и чем дальше дерево от наблюдателя - тем ниже степень детализации.

Чтобы определить, с какой степенью детализации следует загружать каждое из деревьев МТ, необхо-

димо воспользоваться пространственным поиском объектов на плоскости. Эффективным способом такого поиска является, например, *R*-дерево [12].

R-дерево стоится при загрузке структуры МТ с диска. В качестве его элементов выступает структура, состоящая из номера дерева и указателя на открытый файл, содержащий дерево МТ. Таким образом, после построения R-дерева достаточно загрузить лишь основную часть МТ, остальные деревья будут загружаться в соответствии с алгоритмом, подробно описанным в [30, 54].

Алгоритм работает следующим образом. При помощи *R*-дерева производится поиск всех деревьев МТ, попавших в зону интереса, после чего, все деревья, выгруженные в память, но не попавшие в зону интереса из памяти полностью выгружаются. А далее, для всех деревьев, попавших в зону интереса, устанавливается запрашиваемый уровень детализации, в зависимости от критерия визуализации, и в соответствии с ним дерево загружается в память.

Что касается преимущества данного алгоритма перед простым линейным просмотром всех деревьев, то оно заключается в том. что для больших поверхностей зона интереса очень невелика, таким образом, в каждый момент времени обрабатывается лишь небольшое число деревьев.

Определив для каждого из деревьев МТ уровень детализации, нужно загрузить дерево в соответствии с ним. Для загрузки дерева с заданным уровнем детализации используется следующий алгоритм [30].

Начало алгоритма.

— *Шаг 1.* Если действительный уровень детализации равен требуемому, то алгоритм заканчивает работу. Иначе переход на Шаг 2.
— *Шаг 2.* Если действительный уровень детализации больше требуемого, то переход на Шаг 3, иначе на Шаг 4.
— *Шаг 3.* Если *Число загруженных фрагментов* дерева больше, чем *Порог загрузки*× *Запрашиваемый уровень детализации* × *Максимальное число фрагментов в дереве*, то всем ссылкам на нижний фрагмент треугольников из покрываемой части лишних фрагментов присваивается указатель на сток МТ. Удаляем все лишние фрагменты и освобождаем память.
— *Шаг 4.* Если *Число загруженных фрагментов* дерева меньше, чем *Порог загрузки*× *Запрашиваемый уровень детализации* × *Максимальное число фрагментов в дереве*, то производится выделение памяти под нужные фрагменты, после чего они загружаются из файла.

Конец алгоритма.

В данном алгоритме используется *Порог загрузки*, необходимый для повышения скорости работы. Выигрыш получается из минимизации обращений к жесткому диску.

Таким образом, вышеописанные алгоритмы построения и визуализации сверхбольших поверхностей позволяют интерактивно работать с моделями, существенно превосходящими объем доступной оперативной памяти.

Адаптивная триангуляция

Идея методов адаптивной триангуляции заключается в том, чтобы делать уровень детализации (и шаг получаемой сетки) обратно пропорционален расстоянию до наблюдателя. Построению адаптивной триангуляции посвящено множество работ, начиная с алгоритмов, представленных П. Линдстромом (P. Lindstrom) и др. [49] и М. Дюшейн (M. Duchaineau) и др. [32]. Подробный обзор различных многомасштабных моделей для визуализации сложных поверхностей в реальном времени представлен в работах [60, 61]. Для применения алгоритмов в системах реального времени ведутся исследования методов сжатия, например [53]. В работе [53] представлен метод геометрических карт отсечений (geometry clipmaps), в котором применяется техника сжатия изображений с потерями, описанная в [54]. Триангуляция, порождаемая методом, является регулярной и зависит только от положения камеры в пространстве. Благодаря этому удается достичь высокой эффективности использования GPU.

С другой стороны, из-за того что триангуляция не адаптирована к локальным особенностям поверхности, она оказывается чрезмерно избыточной. Более существенным недостатком метода является то, что он не может гарантировать заданную точность аппроксимации, что будет особенно заметно на сложных поверхностях. Ещё один метод, в котором применяется сжатие, называется C-BDAM и представлен в работе [36]. Метод использует двухэтапный алгоритм сжатия изображений с помощью вейвлет-преобразования [68], позволяющий гарантировать заданную точность восстановления. В методе C-BDAM для каждого фрагмента упрощенной модели используются регулярные триангуляции, не адаптированные к особенностям поверхности, что увеличивает нагрузку на GPU. Ещё один алгоритм, представленный в работе [31], предлагает метод кодирования адаптивной триангуляции и позволяет добиться умеренной степени сжатия в 8-9 раз. Вопрос сжатия в [31] не рассматривается.

В работе [51] представлена многомасштабная модель рельефа местности, которая позволяет эф-

фективно кодировать адаптивную триангуляцию поверхности, используя единый подход. В основе метода лежит идея, что вейвлет-коэффициенты, полученные при сжатии сетки высот, могут быть также использованы и для построения адаптивной триангуляции. Близкие подходы были реализованы в методах М. Гросса (M. Gross) и др. [37], а также А. Переберина [9]. В этих двух методах вейвлет-коэффициенты используются только для построения адаптивной триангуляции, а сжатие не применяется. Что еще более важно, описанные методы не могут обеспечить заданную точность восстановления. В качестве решения данной проблемы предлагается двухэтапный метод кодирования адаптивной триангуляции, при котором вейвлет-коэффициенты образуют базовую информацию, которая дополняется остаточной информацией, гарантирующей, что адаптивная триангуляция аппроксимирует поверхность с заданной погрешностью.

На первом этапе построения сжатого многомасштабного представления исходная сетка высот последовательно фильтруется и прореживается, в результате чего получается многоуровневая пирамида (рис. 21а), каждый следующий уровень которой представляет собой сетку с возрастающим в два раза шагом дискретизации по каждому направлению, аппроксимирующую исходную с уменьшающейся точностью. Такая же структура данных используется в методе геометрических карт отсечений [53]. Для построения пирамиды используется нормализованный низкочастотный вейвлет-фильтр анализа Коэна-Добеши-Фово 9/7.

Каждый уровень пирамиды разбивается на квадратные блоки, имеющие одинаковое количество элементов, которые объединяются в иерархическую структуру квадродерева (рис. 21б). Построенная иерархия сжимается нисходящим рекурсивным алгоритмом, начиная от блоков на самом грубом уровне приближения (уровень 0). При этом для каждого блока кодируется информация, которая позволяет уточнить его сетку высот и получить сетки высот 4 дочерних блоков.

Согласующая область состоит из нескольких контуров (трех на рисунке 22). Для обеспечения согласованности сеток высот достаточно одного контура, но для того чтобы элементы карты нормалей на границах были вычислены одинаково, требуется большее количество. Каждый контур сжимается без использования вейвлет-преобразования. Элементы контура рассматриваются как одномерная последовательность отсчетов высоты, которые квантуются и кодируются.

В работе [20] вводится дополнительно специальная структура нуль-деревьев вейвлет-коэффициентов,

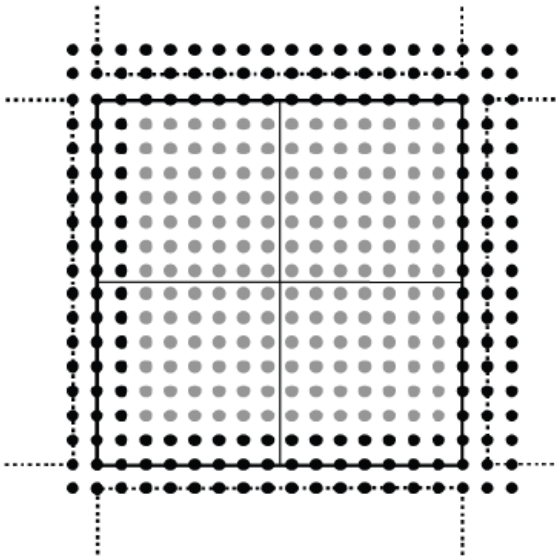

Рис. 22. Согласующая область.

полученная при сжатии сетки высот, используемая как базовая информация. Она определяет сетку высот и триангуляцию, которые почти везде удовлетворяют заданной погрешности. Чтобы гарантировать заданную точность восстановления, базовый слой дополняется уточняющей остаточной информацией. Степень сжатия сетки высот приведенным методом не уступает лучшим из известных алгоритмов и превосходит их, вместе с тем, в отличие от предшествующих методов, представление эффективно кодирует и адаптивную триангуляцию. При этом система способна обеспечить скорость визуализации адаптивной модели не менее 100 кадров в секунду даже в разрешении HD 1920×1280.

Алгоритм Финча и Бишопа.

Алгоритм, описываемый в работе [34], позволяет значительно снизить затраты памяти, т.к. вычисляет сетку для текущего положения камеры без использования предварительно полученных данных. Эффективность алгоритма достигается за счет использования регулярной сетки и применения заранее определенных простых способов ее огрубления при удалении камеры от отображаемого участка поверхности. Снижение затрат памяти происходит за счет вычисления сетки для текущего положения камеры без использования предварительно полученных данных.

Алгоритм использует понятие уровня детализации (LOD). На каждом уровне LOD на поверхности задается равномерная квадратная сетка. Шаг разбиения свой для каждого уровня и представляет собой степень 2: для $LOD = 0$ шаг является минимальным, для $LOD = 1$ длина стороны каждой ячейки

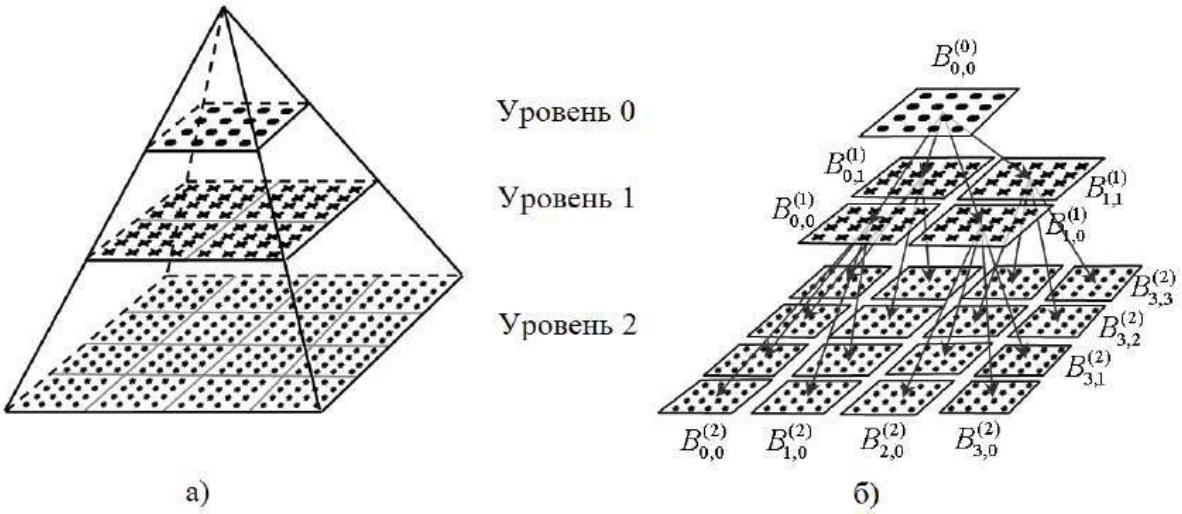

Рис. 21. а) Фильтрованная пирамида; б) квадродерево блоков рельефа.

в два раза больше, чем для $LOD = 0$, и т.д. Ячейки для $LOD = LOD_{max}$ являются самым грубым разбиением и называются страницами. Количество уровней является параметром алгоритма.

Результатом работы алгоритма является триангуляция, представленная в виде последовательности *strip*. Каждый отдельный *strip* направлен вдоль оси Ox поверхности (минорное направление), а последовательность *strip* упорядочена вдоль оси Oy (мажорное направление). В процессе триангуляции происходит обработка не всей поверхности, а только области, видимой в данный момент на экране. Проекция этой области на плоскость Oxy называется $footprint$.

Strip может состоять из треугольников различных уровней детализации. Текущий LOD определяет размер квадрата, который затем разбивается на треугольники определенным образом. На каждом шаге текущий уровень детализации может быть повышен или понижен на 1, или оставлен прежним по сравнению с соседними в мажорном и минорном направлении квадратами, в соответствии с допустимой ошибкой в текущей точке. Функция ошибки является функцией расстояния от точки взгляда наблюдателя до визуализируемой точки поверхности (в данной реализации функция ошибки линейно зависит от этого расстояния). Затем треугольники пришиваются к *strip*, и процесс повторяется, пока *strip* не достигнет границы $footprint$.

Особенностью алгоритма является то, что уровни детализации соседних в минорном и мажорном направлениях ячеек не могут отличаться больше, чем на 1.

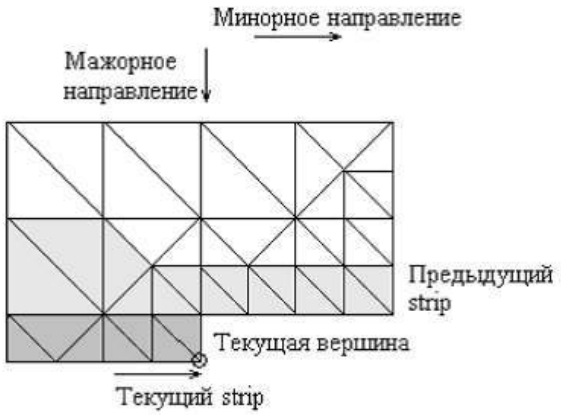

Рис. 23. Генерирование стрипов.

Существует пять способов разбиения ячейки (рис. 24).

Три из них (a, b, c) соответствуют случаю, когда уровень детализации текущего *strip* не меняется, а разбиение зависит от LOD предыдущего *strip* в данном месте. Четвертый способ (d) соответствует случаю повышения уровня детализации (рис. 25). Следующая ячейка будет генерироваться со стороной, в два раза меньшей. Так как по высоте она тоже в два раза меньше, то возникает необходимость в создании еще одного *strip*. Его построение выполняется рекурсивно.

Пятый случай (e) - случай понижения уровня детализации. Текущий *strip* заканчивается, а следующий - в этом месте увеличивает шаг в два раза (рис. 26).

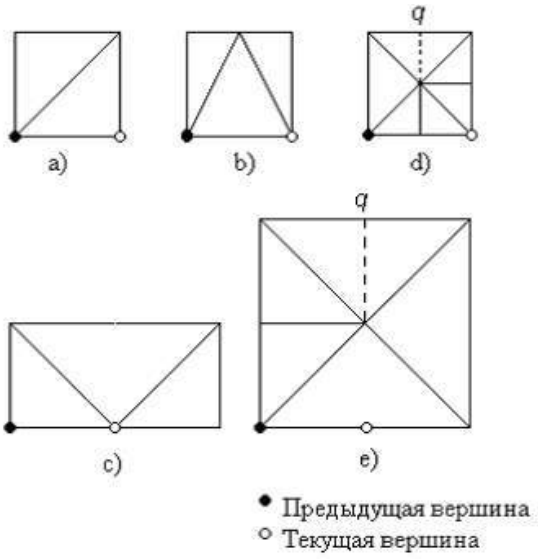

Рис. 24. Способы разбиения ячеек.

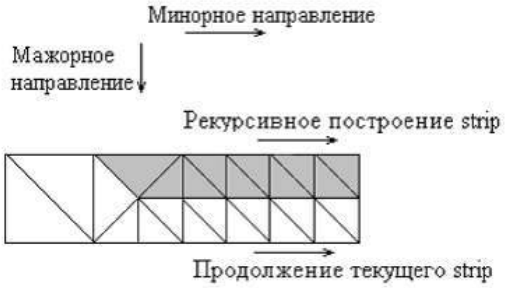

Рис. 25. Улучшение детализации.

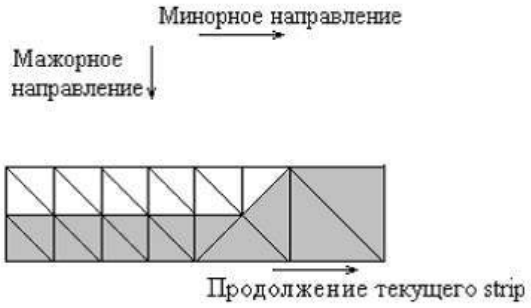

Рис. 26. Огрубление детализации.

Современные направления исследований и разработок

Моделирование и реалистичная визуализация поверхностей является ключевой составляющей многих приложений разной направленности. И основным требованием к подсистеме визуализации подобных приложений является достижение реалистичности виртуальной поверхности (или рельефа) при сохранении интерактивной частоты отображения (25 кадров в секунду и более).

Нетривиальной задачей при построении трехмерных моделей по последовательности дальностных изображений, представленных в виде отдельных триангуляционных сеток (оболочек), является задача объединения этих триангуляционных сеток [1]. Ее решение подразумевает построение единой связной оболочки, отвечающей исходным данным, без дублирования участков поверхности, видимых с нескольких видов. Важным требованием к алгоритмическому решению задачи является высокая скорость обработки данных, поскольку для насыщенных сцен со сложными объектами может потребоваться много видов с большим количеством описывающих треугольников (порядка 10^5, 10^6 и более).

На современном этапе акцент при увеличении производительности приложений сместился в сторону активного использования параллельных систем и сопроцессоров [1]. В качестве сопроцессоров к CPU (Central Processing Unit) обычно выступают GPU (Graphics Processing Unit), реже сопроцессоры на базе FPGA (Field Programmable Gateway Array). В первом случае повышение производительности достигается благодаря широкому использованию массивного векторного параллелизма, во втором - благодаря конвейеризации и специализации вычислителей, настраиваемых под конкретную задачу.

Графические потоковые процессоры GPU характеризуются большой степенью параллелизма, обладая сотнями и даже тысячами вычислительных ядер. Например, современные GPU несут на своем борту более 1000 потоковых процессоров каждого типа, что позволяет существенно повысить производительность за счет параллельного использования всех процессоров. Однако, использование потоковых процессоров накладывает большое число ограничений на выполняемые алгоритмы (в основном, из-за изначальной ориентированности на задачи 3D-графики). Эти ограничения делают невозможным прямой перенос алгоритмов обработки изображений для выполнения на потоковых процессорах. Среди этих ограничений отсутствие рекурсии, прямого доступа к памяти на запись, неопределенный результат чтения из результирующего изображения и др. [16].

Между тем, превосходство потоковых над обычными процессорами в десятки раз по вычислительной мощности делает перспективным обработку двумерных изображений именно на потоковых процессорах [17].

Следует учитывать, что разработка приложений, использующих GPU или FPGA, требует дополни-

тельных временных затрат и специфических знаний. Наибольшую сложность представляет использование FPGA, так как в этом случае требуется разработка не только программной, но и аппаратной части (логической схемы, размещаемой в устройстве), что в свою очередь предполагает знания в области схемотехники. При этом именно FPGA в ряде случаев позволяют достигать ускорения вычислений там, где традиционные параллельные архитектуры не дают прироста производительности.

Современные графические API (OpenGL, DirectX) позволяют программировать потоковые процессоры на языке высокого уровня (GLSL для OpenGL и HLSL для DirectX), что значительно упрощает процесс разработки, сокращает время. Из двух указанных графических API наиболее удобной является OpenGL. Незначительно отставая от своего конкурента в технических возможностях, OpenGL обладает несомненным плюсом - кроссплатформенностью, что позволяет разрабатывать приложение под широкий круг ОС: Windows, Unix, MacOSи др.

До недавнего времени предпринималось сравнительно мало попыток распараллеливания алгоритмов восстановления поверхностей, из которых стоит отметить [45], где используется октодерево в качестве структуры данных для хранения обрабатываемых точек. В работе [44] исследуется масштабируемость алгоритма восстановления на основе поверхностей Пуассона при использовании адаптивной сетки в качестве ускоряющей структуры.

Параллельный алгоритм построения триангуляции Делоне на плоскости при помощи графического процессора рассматривается в работе [18]. В этой работе рассматривается технология NVIDIA CUDA - неграфические вычисления на графических процессорах. Там же показана разница между CPU и GPU в параллельных расчётах, приведен обзор возможностей NVIDIA CUDA и области применения параллельных расчётов на GPU.

В работе [6] предлагается параллельный алгоритм восстановления поверхности для архитектур с массовым параллелизмом. За основу разработанного алгоритма взят алгоритм «марширующих кубов». В работе описывается разработанная эффективная структура для входного набора данных, параллельные алгоритмы построения этой структуры, удобный формат хранения данных, а так же предложена архитектура системы для реализации в рамках технологии гетерогенных вычислений OpenCL.

Перспективной также является идея использования потоковых процессоров для обработки изображений в браузерных приложениях в связи с их ограниченным доступом к вычислительным возможностям центрального процессора. Прямое исполнение кода на центральном процессоре из браузерного приложения невозможно без использования дополнительных модулей для браузера - плагинов. Механизм поддержки плагинов варьируется от браузера к браузеру, что вносит дополнительные трудности и требует их разработки под каждый поддерживаемый браузер.

Долгое время обработку изображений в браузерных приложениях можно было реализовать с помощью двух основных подходов: с использованием java-апплета и с использованием Flash-компонента. Так, например, для обработки изображений в java-апплетах широко используется библиотека ImageJ [21].

При использовании обоих подходов требуется установка специальных плагинов, что является во многих случаях нежелательным или невозможным (по соображениям безопасности, ввиду отсутствия плагина для нужной платформы или браузера, ввиду недостаточной квалификации пользователя для установки и т.д.). Кроме того, в обоих случаях обработка изображений производится байт-кодом, выполняемым на виртуальной машине Flash или Java. Таким образом, скорость обработки изображений будет значительно уступать скорости программ на компилируемых языках, таких как C/C++, и тем более скорости обработки изображений на графических потоковых процессорах.

С появлением стандарта HTML5 стала возможна обработка изображений непосредственно на языке javascript с использованием элемента canvas [42]. Преимущество метода заключается в отсутствии необходимости установки каких-либо плагинов, однако данный метод затрудняет использование многоядерности, а программа обработки изображений выполняется на виртуальной машине javascript. Эти факторы негативно отражаются на производительности подхода: в [42] показывается, что его скорость в несколько раз уступает скорости обработки изображений с помощью ImageJ.

В настоящее время широко распространена технология PixelBender [71], реализованная в Adobe Flash начиная с версии 10.0. Эта технология позволяет в ограниченном объеме использовать возможности потоковых процессоров, разрабатывая программы для них на специальном языке, специфичном для Flash. Однако, технология реализована внутри Flash, а значит требует установки плагина и имеет все перечисленные проблемы технологий, требующих плагинов. Минусом является также и сам специальный язык, который ограничивает доступ к вычислительным возможностям потоковых процессоров, ограничивает простор для оптимизации и т.д.

10 февраля 2011 года была опубликована спецификация открытого стандарта WebGL [72] - реализации OpenGL ES 2.0 для использования в браузерных приложениях. WebGL дает возможность разрабатывать OpenGL-программы непосредственно на javascript, включая использование вершинных и пиксельных шейдеров. При этом становится доступной вся функциональность потоковых процессоров. Математическая модель обработки изображений с учетом особенностей и возможностей WebGL описывается в [16]. Там же описываются математические модели различных инструментов для обработки изображений.

Существенным плюсом в пользу WebGL является ещё и то, что для его использования не требуется установки дополнительных модулей, поддержка технологии реализуется непосредственно в браузере. Всё это делает технологию перспективной для организации обработки изображений в браузерных приложениях.

Заключение

В данной работе был сделан обзор основных алгоритмов триангуляции и визуализации поверхностей. Описаны основные алгоритмы, принципы их реализации. Следует отметить ячеечные методы, преимущество которых перед другими алгоритмами заключается еще и в том, что методы этого типа часто довольно просты в реализации, и предоставляют возможность визуализации "нетривиально"заданных скалярных полей. Так, к примеру, создать регулярную сетку на основе нерегулярной значительно проще, нежели восстановить функцию в каждой точке пространства. Это же относится и к проблеме восстановления поверхности по "срезам возникающей в томографии.

Учитывая, что алгоритмы триангуляции являются неотъемлемой частью практически всех программных продуктов 3D-графики, интенсивно ведутся работы по их усовершенствованию. Также интенсивно ведутся работы по их аппаратной реализации в графических акселераторах.

Благодарности.

Авторы выражают благодарность доктору физико-математических наук, профессору Станиславу Владимировичу Клименко за поддержку данной работы и Российскому фонду фундаментальных исследований за финансирование исследований в рамках гранта РФФИ мол_а_вед_2012 12-07-33059.

Литература

[1] *Андреев А. Е., Силкин И. М., Шафран Ю. В.* Прогнозирование производительности при реализации алгоритмов на гибридных архитектурах с сопроцессорами. // Современные проблемы науки и образования : электрон. журнал. —2012. —№ 3. — http://www.science-education.ru/103-6389.

[2] *Делоне Б. И.* О пустоте сферы // Изв. АН СССР. ОМЕН. 1934, 4. 793-800.

[3] *Дижевский А Ю.* Визуализация трехмерных объектов и геометрические аспекты выявления особенностей. // Дис. . . . канд. техн. наук. 2011.

[4] *Дижевский А Ю.* Общий подход к реализации методов построения триангуляции неявно заданных поверхностей, использующих разбиение пространства на ячейки. // Вычислительные методы и программирование, 2007. —Т. 8. — С. 286-296

[5] *Еврухимов В. Л., Капустин А. Д., Малашкина А. В.* Реализация алгоритма визуализация местности в режиме реального времени. // Материалы конф. по комп. граф. и визуализации «ГрафиКон'99»., — Москва, 1999.

[6] *Козлов Д., Турлапов В.* Алгоритм восстановления поверхности из облака точек на графическом процессоре. // ГрафиКон'2010: 20-я Международная Конференция по Компьютерной Графике и Зрению, 20-24 сентября 2010, Санкт-Петербург, Россия. — http://www.graphicon.ru/proceedings/2010/conference/RU/Se5/44.pdf.

[7] *Мирза Н С.* Разработка алгоритмов построения и визуализации больших поверхностей на основе мультитриангуляции. // Дипломная работа, Томск, 2004 г.

[8] *Мирза Н С., Скворцов А В., Чаднов Р В.* Визуализация сверхбольших поверхностей // Вестник ТГУ, 2006. —№290, март, — С. 209-219.

[9] *Переберин А. В.* Многомасштабные методы синтеза и анализа изображений. // Дисс. . . . канд. физ.-мат. наук. —М.: ИПМ им. М.В. Келдыша, 2002. — 138 с.

[10] *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение. // Пер. с англ. М.: Мир, 1989.

[11] *Семенихин А.* Сравнительный анализ методов интерактивной триангуляции сеточных функций. // On-line журнал "Графика и мультимедиа вып. 6, 2004. — http://cgm.graphicon.ru/issue6/triangulation_comp/.

[12] *Скворцов А. В.* Глобальные алгоритмы R-деревьев // Геоинформатика: Теория и практика. Вып. 1. Томск: Изд-во Том. ун-та. 1998.

[13] *Скворцов А. В.* Обзор алгоритмов построения триангуляции Делоне. // Вычислительные методы и программирование. 2002, 3.

[14] *Скворцов А. В.* Триангуляция Делоне и ее применение. // Томск: Изд-во Том. ун-та. 2002. 128 с.

[15] *Скворцов А. В., Костюк Ю. Л.* Эффективные алгоричмы построения триангуляции Делоне. // Геоинформатика. Теория и практика. Вып. 1. Томск: Изд-во Томского ун-та. 1998.

[16] *Сморкалов А.* Обработка изображений в браузерных приложения на потоковых графических процессорах. // ГрафиКон'2011: 21-я Международная конференция по компьютерной графике и зрению:

Москва, МГУ имени М.В. Ломоносова, 26-30 сентября 2011 г.: Труды конференции. —М.: МАКС Пресс, 2011. —270 с., —стр. 243-246.

[17] *Сморкалов А. Ю., Морозов М. Н.* Поддержка дискретных вейвлет-преобразований для компрессии в системе обработки изображений на потоковых графических процессорах. // Материалы всероссийской научно-практической конференции "Информационные технологии в профессиональной деятельности и научной работе". —Йошкар-Ола: МарГТУ, 2010. —С. 91-96.

[18] *Суржко А. С., Терпугов В. Н.* Построение двухмерной триангуляции Делоне при помощи графического процессора. // Актуальные проблемы механики, математики, информатики: сб. тез. науч.-практ. конф. (Пермь, 12–15 октября 2010 г.) // гл. ред. В.И. Яковлев; Перм. гос. ун-т. —Пермь, 2010.

[19] *Фукс А. Л.* Предварительная обработка набора точек при построении триангуляции Делоне. // Геоинформатика. Теория и практика. Вып. 1. Томск: Изд-во Томского ун-та. 1998. 48-60.

[20] *Юсов Е А., Турлапов В Е.* Эффективное кодирование адаптивной триангуляции рельефа в контексте иерархического вейвлет-сжатия сетки высот. // Вестник Нижегородского университета им. Н.И. Лобачевского, 2010. —№5(1). — С. 209-219.

[21] *Abramoff M. D., Magelhaes P. J., Ram S. J.* Image Processing with ImageJ. // Biophotonics International, volume 11, issue 7, pp. 36-42, 2004.

[22] *Bern M., Eppstein D.* Mesh Generation and Optimal Triangulation. // Computing in Euclidean Geometry Eds. World Scientific, 1992, —Pp. 23-90

[23] *Bloomenthal J.* Ail Implicit Surface Polvgonizer. // In P. Heckbert editor. Graphics Gems IV. Academic Press, Boston, 1994. Pp. 324-349.

[24] *Carneiro B. P., Silva C. T., Kaufman A. E.* Tetra-Cubes: An algorithm to generate 3D isosurfaces based upon tetrahedra. // SIGGRAPH'96. Pp. 205-210.

[25] *Bernardo P. Carneiro, Arie E. Kaufman* Tetra-Cubes: An algorithm to generate 3D isosurfaces based upon tetrahedra. // SIGGRAPH'96, pp. 205-210.

[26] *Chernyaev E. V.* Marching Cubes 33: Construction of Topologically Correct Isosurfaces. // Teclmical Report CERN CN 95-17. CERN Institute for High Energy Physics, 1995.

[27] *Cignoni P.* Speeding up Isosurface Extraction Using Interval Trees. // IEEE Transaction on visualization and CG, vol. 3, —no. 2, April-June 1997.

[28] *Crespin B., Guitton P., Schlick C.* Efficient and accurate tessellation of implicit sweeps. // Proceedings of CSG'98, 1998.

[29] *De Floriani L., Magillo P., Puppo E.* Building and traversing a surface at variable resolution. // Proc. Conf. On Visualization '97. 1997.

[30] *De Floriani L., Marzano P., Puppo E.* Multiresolution Models for Topographic Surface Description. // The Visual Computer. 1996. Vol. 12. N. 7. P. 317-345.

[31] *Dick C., Schneider J., Westermann R.* Efficient Geometry Compression for GPU-based Decodingin Realtime Terrain Rendering. // Computer Graphics Forum. 2009. V. 28, No 1. P. 67-83.

[32] *Duchaineau M., Wolinsky M., Sigeti D. E.* Roaming Terrain: Real-time Optimally Adapting Meshes. // In Proc. Visualization 97. IEEE, Computer Society Press, Los Alamitos, California, 1997. P. 81-88.

[33] *Durst M.* Letters: Additional Reference to Marching Cubes. // Computer Graphics, 1988. —vol.,22, —No. 2, —Pp. 72-73.

[34] *Finch M., Bishop L.* Sorted Height Field Rendering. // `http://www.ndl.com/wpPeCГers/ndlter.html`.

[35] *Fowler R. J., Lirtle J. J.* Automatic extraction of irregular network digital terrain models. // Computer Graphics. 1979. —Vol. 13. —N. 3. —P. 199-207.

[36] *Gobbetti E., Marton F., Cignoni P. et al.* CBDAM - Compressed Batched Dynamic Adaptive Meshes for Terrain Rendering. // Computer Graphics Forum, 2006. V.,25. —No,3.

[37] *Gross M. H., Staadt O. G., Gatti R.* Efficient Triangular Surface Approximations Using Wavelets and Quadtree Data Structures. // IEEE Trans, on Visualization and Computer Graphics. V. 2, No. 2. June, 1996. Pp. 130-143.

[38] *Gueziec A.* Exploiting Triangulated Surface Extraction Using Tetrahedral Decomposition. // IEEE Transactions on Visualization and Computer Graphics, Vol. 1, Issue 4, Pp. 328 - 342, December, 1995.

[39] *Hilton A., Stoddart A. J., Illingworth J., Windeatt T.* Marching triangles: Range image fusion for complex object modeling // International Conference on Image Processing, 1996. `ftp://ftp.ee.surrey.ac.Uk/pub/vision/papers/16-icip96.ps.Z`.

[40] *Hoppe H.* Progressive Meshes. // Computer Graphics, 1996.

[41] *Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W.* Surface reconstruction from unorganised points. // SIGGRAPH'92 proceedings, 26(2), —Pp. 71-78.

[42] *Kai Uwe Barthel, Karsten Sehulz* ImageJ in the web? - Image processing in the browser using HTML5. // In proceedings of ImageJ Conference, 2010.

[43] *Karkanis T., A. James Stewart* Curvature-Dependent Triangulation of Implicit Surfaces. // IEEE Computer Graphics and Applications, March 2001. —V. 21 —No. 2, —Pp. 60-69.

[44] *Kazhdan M., Bolitlio M., Hoppe H.* Poisson surface reconstruction. In Proceedmgs of SGP'06, 61-70 // In Proceedmgs of SGP'06, 2006, —Pp. 61-70

[45] *Kun K., Minmin G., Xin H., Baimng G.* Higlily parallel surface reconstruction. // Microsoft research Asia.

[46] *Lee D., Schuchtcr B.* Two algorithms for constructing a Delaunay triangulation. // Int. Jour. Coinp. and Inf. Sc. 1980. No 3, Pp. 219-242.

[47] *Lee J.* Comparison of existing methods for building triangular irregular network models of terrain from grid digital elevation models. // Int. Journal of GIS. 1991. Vol. 5. N. 3. P. 267-285.

[48] *Lewiner T., Lopes H., Vieira A., Tavares G.* Efficient implementation of Marching Cubes' cases with topological guarantees. // Journal of Graphics Tools. 2003. —Vol. 8. —No. 2. —pp. 1-15.

[49] *Lindstrom P., Koller D., Ribarsky W. et al.* Realtime, Continuous Level of Detail Rendering of Height Fields. // Proc. SIGGRAPH 96. ACM SIGGRAPH, 1996. —P. 109-118.

[50] *Lingas A.* The Greedy and Delaunay triangulations are not bad... // Lect. Notes Comp. Sc. 1983. —158. —Pp. 270-284.

[51] *Lopes A., Brodlie K.* Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. // IEEE Transactions on Visualization and Computer Graphics, 2002. —Pp. 16-29

[52] *Lorensen W. E., Cline H. E.* Marching Cubes: A high resolution 3D surface construction algorithm. // CG vol.21, no.4, July 1987.

[53] *Losasso F., Hoppe H.* Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids. // ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004). —23(3). —Pp. 769-776.

[54] *Malvar H.* Fast Progressive Image Coding without Wavelets. // Data Compression Conf. (DCC'00). — P. 243-252.

[55] *Manaclier G., Zobrist A.* Neither the Greedy nor the Delaimay triangulation of planar point set approximates the optimal triangulation. // Inf. Proc. Let. 1977. —Vol. 9. —N. 1. —P. 31-34.

[56] *Matveev S. V* Approximation of Isosurface in the Marching Cube: Ambiguity problem. // Proceedings IEEE Visualization '94, —pp. 288-292

[57] *Montani C., Scateni R., Scopigno R.* Discretized Marching Cubes. // Visualization'94 Proceedings.//IEEE Computer Society. IEEE Computer Society Press. 1994. pp. 281-287

[58] *Nielson G. M., Hamann B.* The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes. // IEEE Visualization. Proceedings of the 2nd conference on Visualization '91, 1991. —pp. 83-91

[59] *Ohtake Y., Belyaev A. G.* Dynamic meshes for accurate polygonization of implicit surfaces with sharp features. // Proceedings of the International Conference on Shape Modeling & Applications, Page: 74, 2001. — `http://cis.k.hosei.ac.jp/~F-rep/OhtakeSmi01.pdf`.

[60] *Pajarola R.* Overview of Quadtree-Based Terrain Triangulation and Visualization. // Technical Report UCIICS-02-01, I&C Science, University of California Irvine, 2002.

[61] *Pajarola R., Gobbetti E.* Survey on Semi-Regular Multiresolution Models for Interactive Terrain Rendering. // Visual Computer, 2007. —V. 23. —No. 8. —Pp. 583-605.

[62] *Puppo E.* Variable resolution triangulations. // Computational Geometry, 1998. —V. 11.

[63] *Rocchini C., Cignoni P, Ganovelli F.* Marching Intersections: an Efficient Resampling Algorithm for Surface Management. // International Conference on Shape Modeling & Applications, 2001. `http://smi2001.ima.ge.cnr.it/abstracts/47.pdf`.

[64] *Shapiro M.* A note on Lee and Schachters algorithm for Delaunay triangulation. // Inter. Jour, of Сотр. and Sciences, 1981. —10, —N 6. —Pp. 113-118.

[65] *Skala V.* Precission of iso-surface extraction from volume data and visualization. // Conference on Scientific Computing, 2000. —Pp. 368-378

[66] *Treece G. M., Prager R. W., Gee A. H.* Regularised marching tetrahedra: improved iso-surface extraction. // Technical Report CUED/F-INFENG/TR 333, Cambridge University Engineering Dept, September 1998. `http://citeseer.ist.psu.edu/3098regularised.html`.

[67] *Witkin A. P., Heckbert P. S.* Using Particles to Sample and Control Implicit Surfaces. // Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pp: 269-277, 1994. `www.cs.cmu.edu/~aw/pdf/particles-reprint.pdf`.

[68] *Yea S., Pearlman W.* A Wavelet-Based Two-stage Near-Lossless Coder. // Proc. 1C IP. 2004. —Pp. 2503-2506.

[69] *Yusov E.* Adaptive Context Modeling for Efficient Image and Elevation Data Compression. // Proc. of the 20th International Conference on Computer Graphics and Vision «GraphiCon'2010». St. Petersburg, Sept. 20-24, 2010. —P. 22-29.

[70] *Yutaka Ohtake, Belyaev A. G.* Dual/Primal mesh optimization for polygonized implicit surfaces. // ACM Symposium on Solid Modeling and Applications, pp. 171-178, 2002. `http://cis.k.hosei.ac.jp/~F-rep/SM02ob.pdf`.

[71] `http://www.adobe.com/content/dam/Adobe/en/devnet/pixelbender/pdfs/pixelbender_guide.pdf` — Adobe Pixel Bender Guide.

[72] `https://www.khronos.org/registry/webgl/specs/1.0/` — The Khronos Group. WebGL 1.0 Specification.