Алгоритм упрощения триангуляции посредством стягивания рёбер, сохраняющий топологию

Я. С. ПЕПКО
Московский государственный университет им. М. В. Ломоносова
e-mail: p.yaroslav.s.21@gmail.com

В. В. БОРИСЕНКО
Московский государственный университет им. М. В. Ломоносова
e-mail: vladimir_borisen@mail.ru

УДК 004.92+004.932

Ключевые слова: триангуляция изоповерхности, марширующие тетраэдры, алгоритмы упрощения триангуляции.

Аннотация

Триангуляция широко используется для представления моделей реальных объектов в цифровой форме, и часто, чтобы получить желаемую модель, нам нужно построить триангуляцию по данным другого вида, например по воксельной модели. Существуют методы, которые позволяют это сделать, однако итоговая триангуляция не всегда имеет желаемое качество. Один из способов решить эту проблему — это алгоритмы упрощения триангуляции. Однако они имеют свои недостатки, в частности, в некоторых случаях топология модели может изменяться в процессе упрощения, что ведёт к отказу от упрощения тетраэдрической сети в некоторой локальной области. В этой статье мы рассмотрим наивный метод упрощения триангуляции посредством стягивания рёбер и его недостатки, а также предложим его модификацию, позволяющую стягивать любые рёбра, избегая нарушения топологии.

Abstract

Ya. S. Pepko, V. V. Borisenko, Topology-preserving triangulation simplification algorithm by edge contraction, Fundamentalnaya i prikladnaya matematika, vol. 24 (2023), no. 3, pp. 153—169.

Triangulation is widely used to represent models of real objects in digital form, and often, in order to get the desired model, we need to triangulate it from data of another kind, for example, from a voxel model. There are methods that allow one to do it, but the resulting triangulation does not always have the desired quality. One way to solve this problem is triangulation simplification algorithms. However, they have disadvantages; in particular, in some cases they can destroy the model topology during the simplification process, which leads to the rejection of the simplification of the tetrahedral mesh in some local area. In this paper, we will consider the naive method of triangulation simplification using edge contraction, its shortcomings, and propose its modification that allows us to contract any edges without topology violations.

Фундаментальная и прикладная математика, 2023, том 24, № 3, с. 153—169.

1. Введение

Данная работа берёт своё начало в задаче построения изоповерхности по дискретным данным, например по трёхмерной матрице плотности, полученной по набору томографических снимков. Одним из основных алгоритмов, позволяющих решить эту задачу, является метод марширующих тетраэдров [5]. Однако использование его в чистом виде обладает некоторыми существенными недостатками. А именно, итоговая триангуляция, как правило, является очень неравномерной. В ней размер треугольника варьируется от порядка грани тетраэдра до сколь угодно малого. При этом присутствуют узкие вытянутые треугольники со сколь угодно большим аспектом (рис. 1). Это не единичные случаи — такие «некачественные» треугольники составляют значительную часть от общего числа. При этом важно подчеркнуть, что возникают они не из-за особенностей триангулируемой поверхности, таких, например, как крутые изгибы, а просто потому, что сечение так попало на тетраэдрическую сеть. Наличие таких треугольников может быть плохо по разным причинам: например, объём модели в памяти и количество вычислений, требуемых для её обработки, неоправданно увеличиваются, когда можно было бы обойтись значительно меньшим числом треугольников; кроме того, это может понижать достоверность результатов работы алгоритмов, чувствительных к качеству триангуляции, таких как

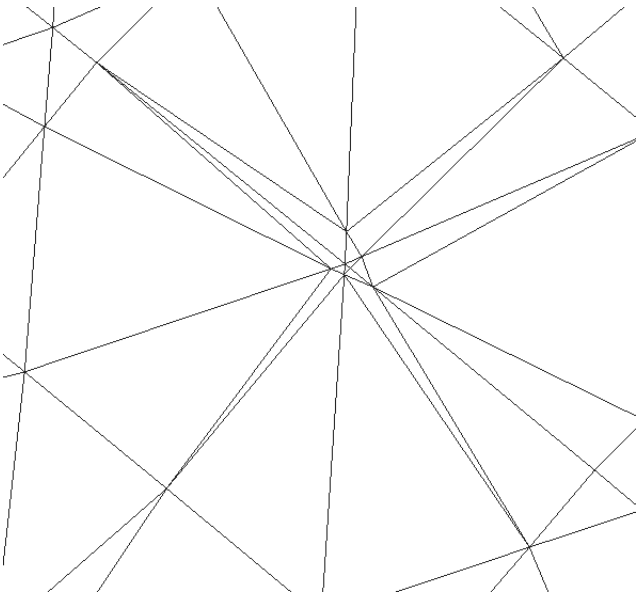

Рис. 1.

гидродинамические расчёты. Наконец, при импорте триангуляции, в которой присутствуют треугольники с очень короткими рёбрами, некоторые программы могут выполнять округления и склейку вершин, что во многих случаях приводит к нарушению топологии исходной модели. Существуют различные подходы к решению этой проблемы, например модификации самого алгоритма марширующих тетраэдров [15]. Но мы в этой работе рассмотрим подход, заключающийся в упрощении сетки уже после её построения; будут изучены проблемы, возникающие при этом, и предложены способы их решения.

2. Сопутствующие работы

Существует множество алгоритмов, направленных на упрощение сети. Задача этих алгоритмов заключается в том, чтобы по данной полигональной сети построить новую, с меньшим числом вершин, рёбер и граней, при этом достаточно близкую к изначальной. Глобально можно выделить два подхода: первый — это алгоритмы уточнения, когда мы начинаем с грубого приближения исходной сети и далее на каждом шаге добавляем в него новые элементы, получая более детальную сеть и приближаясь к исходной; второй — алгоритмы удаления, которые, напротив, начинают с исходной триангуляции и далее на каждом шагу удаляют какие-либо её части, получая упрощённую сеть. В [1, 7, 9, 14] можно найти подробные обзоры существующих алгоритмов. Далее мы будем говорить только о втором подходе. В процессе упрощения мы руководствуемся одним или несколькими критериями, указывающими нам на качество сети, такими критериями могут быть, например, ограничения на аспект входящих в сеть треугольников, размер рёбер или целых треугольников, степень вершин. Такие алгоритмы обычно носят итеративный характер. Одним из самых эффективных и широко распространённых алгоритмов является стягивание ребра в точку [8]. В этом алгоритме по какому-либо критерию выбирается ребро, подлежащее стягиванию, две инцидентные ему вершины заменяются на одну новую, а все рёбра, инцидентные им, становятся инцидентны новой вершине. Два инцидентных данному ребру треугольника при этом удаляются, или, иными словами, схлопываются в рёбра (рис. 2). Местоположение новой вершины можно выбирать по-разному, в простейшем случае это будет середина данного ребра. Но существует и более эффективные алгоритмы выбора новой вершины, например описанный в [6, 8].

Однако не во всех случаях ребро можно беспрепятственно стянуть. При стягивании ребра может произойти переворачивание одной из инцидентных его вершинам граней, в результате чего образуется складка [10]. Что ещё важнее, в некоторых случаях стягивание ребра может приводить к нарушению топологии исходной сети. П. Чарле и Ф. Ламур приводят критерий сохранения топологии при стягивании ребра в терминах теории графов [3], позже Т. Дей, Х. Эдельсбруннер и др. дают более общую топологическую формулировку [4]. С тех пор

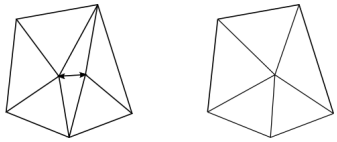

Рис. 2. Здесь и далее рассматриваемое ребро выделено стрелками

при упрощении триангуляции рёбра, стягивание которых может приводить к нарушению топологии, принято просто пропускать, как, например, это делается в [11, 13], по крайней мере нам не известно ни одной работы, где это было бы не так. Обычно таких рёбер очень мало и такой подход допустим, однако, как обсуждалось выше, в некоторых ситуациях требуется удалить их все до единого. Мы же далее приведём алгоритм, позволяющий стягивать такие рёбра без нарушения топологии.

3. Обозначения

Мы используем следующие обозначения:

$V(v)$ — множество вершин, смежных вершине v;
$V(e)$ — множество вершин, инцидентных ребру e;
$V(f)$ — множество вершин, инцидентных грани f;
$E(v)$ — множество рёбер, инцидентных вершине v;
$E(e)$ — множество рёбер, смежных ребру e;
$E(f)$ — множество рёбер, инцидентных грани f;
$F(v)$ — множество граней, инцидентных вершине v;
$F(e)$ — множество граней, инцидентных ребру e;
$F(f)$ — множество граней, смежных грани f;
$T(v)$ — множество тетраэдров, инцидентных вершине v;
$T(e)$ — множество тетраэдров, инцидентных ребру e;
$T(f)$ — множество тетраэдров, инцидентных грани f.

Для вершины, ребра или грани a и множества тетраэдров (в том числе, возможно, множества, состоящего из одного элемента) T будем писать $a \in T$, если найдётся $t \in T$, для которого a целиком лежит в $\bar{t}$.

4. Нарушение топологии

Как было сказано выше, существуют эффективные алгоритмы упрощения триангуляции, основанные на стягивании рёбер. Однако в некоторых ситуациях стягивание рёбер может приводить к нарушению топологии. Здесь нужно

условиться, что мы рассматриваем триангуляции только замкнутых двумерных многообразий, где каждое ребро имеет степень 2; под степенью подразумевается количество треугольных граней, инцидентных данному ребру. Под нарушением топологии мы будем понимать появление рёбер, степень которых отлична от 2. Обычно если стягивание ребра приводит к нарушению топологии, то его просто пропускают, и главная цель этой работы — понять, в каких случаях возникает нарушение топологии, и разработать модификацию алгоритма упрощения триангуляции, которая позволит стягивать такие рёбра, избегая этих нарушений. Приведём пример ситуации, в которой при стягивании ребра возникает подобное нарушение (рис. 3).

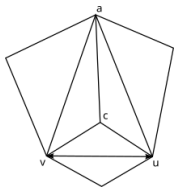

Рис. 3.

Здесь при стягивании выделенного ребра сливаются не только пара рёбер vc и cu, но и рёбра va и au, которые также смежны короткому ребру. Так как удаляются только треугольники, инцидентные выделенному ребру, то пара треугольников, инцидентных ребру va, и другая пара треугольников, инцидентных ребру au, становятся инцидентны одному новому ребру, полученному в результате слияния. В результате мы получаем ребро степени 4, что является нарушением топологии. Критерий, при котором триангуляция после стягивания ребра остаётся триангуляцией с той же топологией, известен [3, 4], однако мы приведём его в слегка изменённой формулировке и со своим доказательством.

Теорема 1. *Нарушение топологии при стягивании ребра vu возникает тогда и только тогда, когда*

$$|V(v) \cap V(u)| > 2.$$

Доказательство. Достаточность. Если $|V(v) \cap V(u)| > 2$, очевидно, что, как в примере выше, найдутся два сливающихся ребра, инцидентные которым грани не удаляются. После стягивания мы получим ребро степени 4.

Необходимость. Пусть после стягивания возникло ребро ab, степень которого отлична от 2. Заметим, что при стягивании мы не добавляем новых рёбер, поэтому данное ребро существовало и раньше. Его степень могла либо увеличиться, либо уменьшиться. Пусть она увеличилась, это значит, что добавилась как минимум одна инцидентная этому ребру грань g. Заметим также, что мы не создаём новых граней, поэтому добавленная грань существовала до стягивания. Это означает, что $a \cup b \not\subset V(g)$ до стягивания, но $a \cup b \subset V(g)$ после. При этом мы не меняем вершины, инцидентные граням. Тогда такое возможно

только в том случае, если, без ограничения общности, $a \in V(g)$ до стягивания, а $b = u$ слилось с $v \in V(g)$ в результате стягивания (рис. 4, 1). Таким образом, $a \in V(v) \cap V(u)$. Покажем, что $a \notin V\big(F(vu)\big)$. Пусть это не так (рис. 4, 2). Заметим, что существует только одна пара подходящих вершин a и v, т. е. только одно ребро av сливается с ab, значит, к ab можно присоединить максимум две новых грани. При этом один из треугольников, инцидентных vu, будет инцидентен и ab и av и будет удалён в процессе стягивания. Таким образом, одна грань, инцидентная ab, будет удалена и одна грань будет к ab добавлена, т. е. степень ab не увеличится.

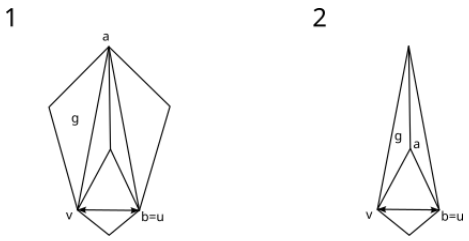

Рис. 4.

Теперь рассмотрим случай, когда степень ab уменьшается. Это может произойти, только если мы удаляем хотя бы одну из граней, инцидентных ab. При стягивании ребра мы удаляем только две грани, инцидентные этому ребру, т. е. одна из них должна быть инцидентна ab. В таком случае, как было рассмотрено выше, ребро ab сольётся с ребром av, что добавит ему одну грань, т. е. количество граней не изменится. Следовательно, случай уменьшения степени невозможен. □

Из рис. 3 ясно, что если в триангуляции существует вершина степени 3, то возможно нарушение топологии вследствие стягивания ребра. Можно задаться вопросом, верно ли обратное, т. е. верно ли, что если $|V(v) \cap V(u)| > 2$, то найдётся w, такая что $\deg w = 3$? Оказывается, что это не так, контрпример приведён на рис. 5, 8. Однако из этой посылки можно получить более общее утверждение о локальной структуре триангуляции, что и делает следующая теорема.

Наблюдение 1. Если между двумя смежными рёбрами нет других смежных им рёбер, то они инцидентны одному треугольнику.

Теорема 2. *Если существует vu, такое что $|V(v) \cap V(u)| > 2$, то найдутся vw, wu, такие что $|V(v) \cap V(w)| > 2$, $|V(w) \cap V(u)| > 2$.*

Доказательство. Доказательство будем проводить конструктивно, рассматривая интересующие нас подграфы триангуляции и то, как они могут быть достроены до полной триангуляции.

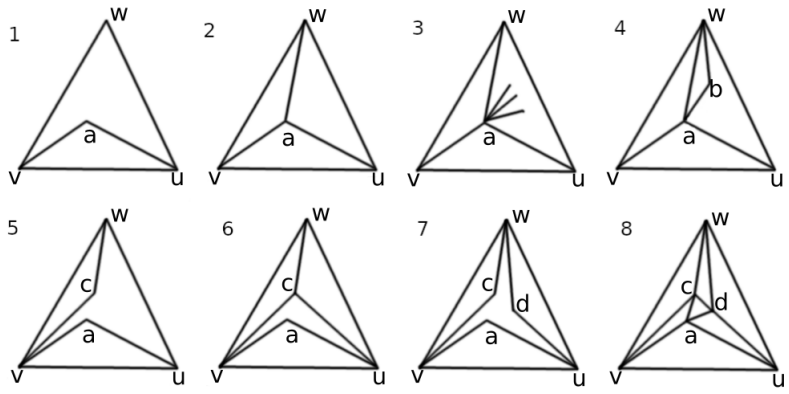

Рис. 5.

Если найдётся vu, такое что $|V(v) \cap V(u)| > 2$, то возникает ситуация, изображённая на рис. 5, 1. Здесь мы рассматриваем только внутренность интересующего нас треугольника (он не является треугольником триангуляции). Рассмотрим ребро va, оно может являться или не являться ближайшим к ребру vw. Пусть оно является ближайшим, тогда по наблюдению 1 существует треугольник vaw (рис. 5, 2). В данном подграфе $\deg a = 3$. Если $\deg a = 3$ и в исходном графе, то в этом случае теорема доказана, если нет, то из a выходит ещё как минимум одно ребро. Пусть, без ограничения общности, рёбра выходят как на рис. 5, 3. Возьмём из них ближайшее к aw, оно существует, так как исходная триангуляция конечна. По наблюдению 1 существует треугольник abw (рис. 5, 4). Заметим, что в треугольнике auw образовался исходный паттерн. Не будем рассматривать его отдельно, а рекурсивно перенесём на него рассуждения, проводимые в отношении треугольника vuw. В силу конечности исходной триангуляции рекурсия когда-нибудь остановится. Теперь рассмотрим случай, когда va не является ближайшим к vw ребром. Пусть таковым является vc, тогда по наблюдению 1 существует треугольник vcw (рис. 5, 5). Теперь wc может являться или не являться ближайшим к wu ребром. Если является, возникает вершина степени 3 (рис. 5, 6), в отношении которой можно снова провести приведённые выше рассуждения; если не является, то возникает ситуация, отражённая на рис. 5, 7, которая доказывает теорему. Скажем также, что такая ситуация реализуема, т. е. такой подграф может быть достроен до исходного графа, один из возможных вариантов показан на рис. 5, 8. □

Данная теорема не потребуется далее и была приведена лишь в качестве теоретического наблюдения. Теперь проверим, могут ли ситуации, приводящие к нарушению топологии, возникать при использовании метода марширующих тетраэдров. Следующая теорема даёт отрицательный ответ на этот вопрос.

Теорема 3. *При использовании метода марширующих тетраэдров для каждого* vu $|V(v) \cap V(u)| = 2$ *в полученной триангуляции.*

Доказательство. Любое ребро не выходит за пределы тетраэдра, которому принадлежит, т. е. двум тетраэдрам ребро может принадлежать, только если оно лежит на гранях, которыми они соприкасаются. Тогда для каждого vu если $|V(v) \cap V(u)| > 0$ и для каждого $w \in V(v) \cap V(u)$ справедливо $vw \in T(v)$, $uw \in T(u)$, то $w \in T(v) \cap T(u) = T(vu)$. Каждый тетраэдр, согласно методу марширующих тетраэдров, добавляет в триангуляцию либо один треугольник, рёбра которого лежат на гранях тетраэдра, либо четырёхугольник с рёбрами на гранях тетраэдра, разделённый на два треугольника диагональю, проходящей внутри тетраэдра. Если vu лежит на грани тетраэдра, то $|T(vu)| = 2$. Тогда для каждого тетраэдра $t \in T(vu)$ если t добавляет в триангуляцию треугольник, то в нём есть лишь одна вершина триангуляции, помимо v и u, и она входит в $V(v) \cap V(u)$; если же t даёт два треугольника, то, помимо v и u, в них будут ещё две вершины, но лишь одна из них принадлежит $V(v) \cap V(u)$. Если же vu лежит внутри тетраэдра, то $|T(vu)| = 1$ и этот тетраэдр даёт четырёхугольник, в котором vu является диагональю, оставшиеся же две вершины четырёхугольника принадлежат $V(v) \cap V(u)$. Таким образом, в любом случае мы получаем $|V(v) \cap V(u)| = 2$. □

Мы видим, что непосредственно после построения триангуляции нарушений топологии возникнуть не может. Однако алгоритм упрощения триангуляции итеративен, и после каждой итерации мы получаем новую сетку. Возможно ли возникновение ситуаций, приводящих к нарушению топологии, на какой-либо итерации алгоритма? Да, возможно. Продемонстрируем это на примере триангуляции, полученной методом Чаня—Пурисимы—Скалы [2, 12].

На рис. 6 изображена часть тетраэдризации, полученной методом Чаня—Пурисимы—Скалы, где четыре тетраэдра расположены вокруг общего вертикального ребра. Сечение дано в двух ракурсах, с видом под углом на первом рисунке и ровно сверху на втором. Сначала стягивается ребро vu, а потом ребро ua. После этих двух шагов алгоритма упрощения происходит нарушение топологии.

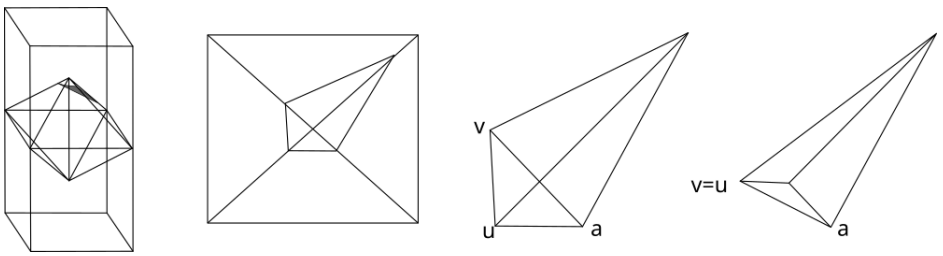

Рис. 6.

Таким образом, мы видим, что ситуации, приводящие к нарушению топологии, действительно могут возникать на практике. В следующем разделе мы предложим модификацию алгоритма упрощения, позволяющую сохранять топологию модели в любых случаях.

5. Алгоритм перестроения триангуляции без нарушений топологии

Прежде чем излагать полный алгоритм упрощения триангуляции, рассмотрим ту его часть, которая касается стягивания одного ребра. Итак, пусть нам дано ребро vu, которое подлежит стягиванию. Первым делом мы должны посчитать $|V(v) \cap V(u)|$. Если оно равно 2, то мы действуем согласно базовому алгоритму, описанному ранее. Если же оно больше 2, то возникают вложенные треугольники выделенного ребра (рис. 7), где треугольниками выделенного

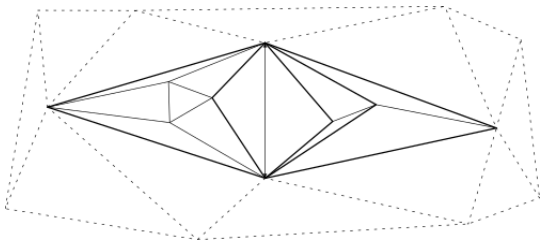

Рис. 7. Вложенные треугольники выделенного ребра отмечены жирным

ребра мы называем все треугольники, образованные тройками попарно смежных рёбер, включающими ребро vu, пусть они и не принадлежат триангуляции. Идея алгоритма заключается в том, что мы будем удалять из триангуляции не только инцидентные vu треугольники, но и все те, что лежат внутри наибольших треугольников выделенного ребра. Иными словами, с двух сторон от ребра vu находятся наборы вложенных треугольников выделенного ребра. С каждой стороны мы находим самый большой из них, назовём его внешним, и удаляем все треугольники триангуляции, лежащие внутри этих двух больших треугольников выделенного ребра.

Теперь зададимся вопросом, как найти внешние треугольники. Если бы мы находились на плоскости, то можно было бы посмотреть на какие-то геометрические характеристики, например, в таком случае сумма длин рёбер однозначно бы определяла, какой треугольник в какой вложен. Однако наша триангуляция находится в трёхмерном пространстве, а потому внутренний треугольник по геометрическим размерам может быть больше внешнего. Таким образом, геометрические характеристики не могут дать нам достоверного ответа на вопрос,

какой треугольник является внешним. Будет уместнее говорить не о размере, а о том, какой из треугольников выделенного ребра отделяет набор вложенных треугольников от остальной триангуляции. Однако на этот вопрос не всегда есть однозначный ответ. Так, например, на сфере, которой гомеоморфны многие встречающиеся на практике поверхности, все треугольники выделенного ребра, кроме двух ближайших к этому ребру, входящих в триангуляцию, будут в этом смысле топологически неразличимы. Это легко можно увидеть, если перетянуть по сфере внешний треугольник с одной стороны на другую, тогда он там станет внешним, а вся остальная триангуляция окажется между ним и предыдущим внешним с этой стороны треугольником, т. е. станет внутренностью в наших вложенных треугольниках выделенного ребра. Ближайшие же к выделенному ребру треугольники можно отличить, во-первых, по тому, что они входят в триангуляцию, во-вторых, вершины этих треугольников, не равные ни v, ни u, являются единственной парой вершин из $V(v) \cap V(u)$, такой что между ними можно построить путь без повторяющихся вершин, не проходящий через v и u и содержащий все вершины из $V(v) \cap V(u)$.

Итак, мы поняли, что в общем случае топологически нельзя выделить внешние треугольники, поэтому нам придётся прибегнуть к эвристическим методам. Примером наивной эвристики может служить евклидово расстояние от каждой вершины $V(v) \cap V(u)$ до двух вершин ближайших к vu треугольников (рис. 8), которые, как было сказано выше, можно установить. Эти два ближай-

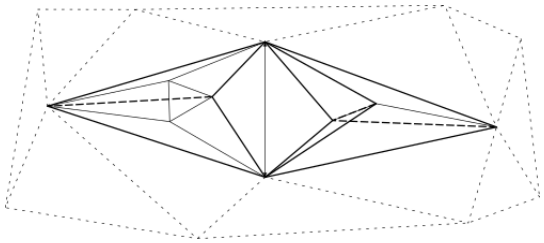

Рис. 8. Евклидово расстояние отмечено жирным пунктиром

ших треугольника будут определять две стороны от ребра vu, и до какого из них расстояние короче, к той стороне мы и будем относить конкретную вершину из $V(v) \cap V(u)$ и соответствующий ей треугольник. Далее, с каждой стороны тот треугольник, расстояние для которого было наибольшим, мы и будем считать внешним. Однако евклидово расстояние не гарантирует нам топологически корректный порядок вложенности треугольников. Более хорошей эвристикой, которая отвечает этому требованию, является длина кратчайшего пути, не проходящего через вершины v и u, по рёбрам (рис. 9). Ещё лучшей эвристикой является количество вершин, находящихся по каждую сторону от того или иного треугольника выделенного ребра (рис. 10), или, иными словами, в его внутренности и внешности. Тогда треугольник будет относиться к той стороне,

с которой их меньше, а внешним с каждой стороны будет треугольник, внутренность которого содержит наибольшее количество вершин. Такая эвристика также даёт корректный порядок вложенности, кроме того, она исключает, что большая часть триангуляции окажется во внутренности одного из внешних треугольников. Стоит также отметить, что, чтобы посчитать количество вершин, не нужно проходить по всей триангуляции, достаточно параллельно запустить два алгоритма роста области с начальными точками в вершинах ближайших к выделенному ребру треугольников. При этом область в процессе роста должна целиком заполнять внутренность треугольника выделенного ребра, прежде чем выходить за его пределы. Тогда, как только одна область включает какую-либо вершину из $V(v) \cap V(u)$, мы относим эту вершину к соответствующей стороне.

Теперь приведём алгоритм. Сейчас сведём его лишь к нахождению внутренности внешних треугольников, подлежащей удалению, и назовём $\mathsf{processEdge}(e)$. Остальную часть изложим дальше. Пусть для каждой вершины c и ребра e триангуляции известны $V(c)$, $F(c)$, $F(e)$. Дано ребро vu, подлежащее стягиванию.

1. Для вершин v и u берём $V(v)$, $V(u)$ и находим $V(v) \cap V(u)$.
2. **if** $|V(v) \cap V(u)| == 2$ **then**
   $\mathsf{removedTriangles.insert}\big(F(vu)\big)$
   завершаем алгоритм
   end if
3. Иначе пусть $F(vu) = \{t1, t2\}$, $V(t1) = \{v, u, a\}$, $V(t2) = \{v, u, b\}$.
4. Применяем алгоритм нахождения вершин внешних треугольников

$$(\mathsf{outerVertex1}, \mathsf{outerVertex2}) = \\ = \mathsf{findOuterVertices}(\mathsf{startVert1}, \mathsf{startVert2}, \mathsf{stopList}),$$

где $\mathsf{startVert1} = a$, $\mathsf{startVert2} = b$, $\mathsf{stopList} = \{v, u\}$:
$\mathsf{unconsidered1.insert}(\mathsf{startVertex1})$
$\mathsf{unconsidered2.insert}(\mathsf{startVertex2})$

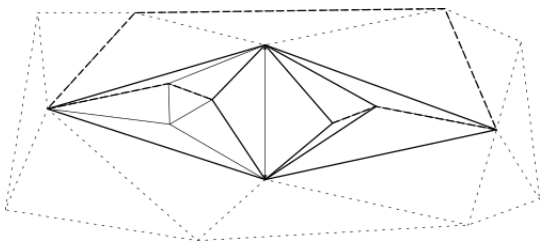

Рис. 9.

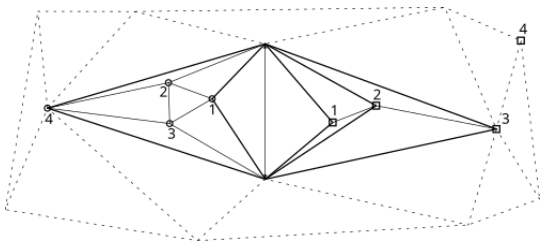

Рис. 10. Вершины по одну сторону отмечены окружностями, а по другую — квадратами. Обратите внимание, что слева после вершины 2 мы добавляем вершину 3, а не 4

```
outerVertex1 = startVertex1
outerVertex2 = startVertex2
loop
    if not unconsidered1.empty() then
        currentVertex = unconsidered1.first()
        if currentVertex ∈ V(v) ∩ V(u) then
            if unconsidered1.size() > 1 then
                currentVertex = unconsidered1.second()
            else
                outerVertex1 = currentVertex
            end if
        end if
        unconsidered1.erase(currentVertex)
        considered1.insert(currentVertex)
        for newVertex ∈ V(currentVertex) do
            if newVertex ∉ considered1 and newVertex ∉ unconsidered1 and
newVertex ∉ stopList then
                unconsidered1.insert(newVertex)
            end if
        end for
    end if
    Аналогичный алгоритм для startVertex2
    if V(v) ∩ V(u) ⊂ considered1 ∪ considered2 then
        выход из цикла
    end if
end loop
return (outerVertex1, outerVertex2)
```

5. Находим все вершины, лежащие внутри внешнего треугольника

$$\text{interVertices1} = \text{findInterVertices}(\text{start}, \text{stopList}),$$

где $\text{start} = a$, $\text{stopList} = \{v, u, \text{outerVertex1}\}$:

```
if start ∈ stopList then
    return ∅
end if
unconsidered.insert(start)
while not unconsidered.empty() do
    currentVertex = unconsidered.extract()
    for newVertex ∈ V(currentVertex) do
        if newVertex ∉ considered ∪ unconsidered ∪ stopList then
            unconsidered.insert(newVertex)
        end if
    end for
    considered.insert(currentVertex)
end while
return considered
```

Аналогично находим interVertices2.

6. Проверяем, не было ли стягивания ребра во внутренности внешних треугольников на одном из предыдущих этапов алгоритма. Если было, то прерываем алгоритм стягивания ребра vu.

```
if (interVertices1 ∪ interVertices2) ∩ replacedVertices ≠ ∅ then
    завершаем алгоритм, пропуская ребро vu
end if
```

7.
```
if outerVertex1 == a then
    removedTriangles.insert(vua)
else
    for c ∈ findInterVertices(a, {v, u, outerVertex1}) do
        removedVertices.insert(c)
        for t ∈ F(c) do
            removedTriangles.insert(t)
        end for
    end for
end if
```

Аналогично для outerVertex2.

Теперь изложим полный алгоритм упрощения триангуляции.

1. Найдём для каждой вершины v и ребра e триангуляции $V(v)$, $F(v)$, $F(e)$:

```
for t ∈ triangles do
    for v ∈ V(t) do
        F(v).insert(t)
```

```
            V(v).insert(V(t) \ v)        ▷ Здесь предполагается использование структуры с уникальными значениями для хранения V(v) либо реализация проверки уникальности вручную.
        end for
        for e ∈ E(t) do
            F(e).insert(t)
        end for
    end for
```

2. Проходим по всем рёбрам и находим среди них те, что достаточно коротки и не связаны с уже стянутыми рёбрами, обрабатываем их:

```
for t ∈ triangles do
    smallEdgeLen = ∞
    for e ∈ E(t) do
        if len(e) < ε and len(e) < smallEdgeLen and ∀v ∈ V(e)
v ∉ replacedVertices ∪ removedVertices ∪ frozenVertices then
            smallEdge = e
            smallEdgeLen = len(e)
        end if
    end for
    if smallEdgeLen == ∞ then
        continue
    end if
    processEdge(smallEdge)
    newVertex = newVertex(smallEdge)
    modifiedVertices.insert(newVertex)
    replacedVertices[V(smallEdge)] = modifiedVertices.size() − 1
                                     ▷ Вставляем не саму вершину, а её индекс в массиве modifiedVertices
    for v ∈ V(smallEdge) do
        frozenVertices.insert(V(v) \ V(smallEdge))
    end for
end for
```

3. Теперь нужно перестроить триангуляцию. Чтобы описать эту часть алгоритма, надо сказать, что вершины и треугольники триангуляции мы храним в массивах и доступ к ним получаем по их индексам. Здесь важно следить, каким образом мы проводим переиндексацию вершин.

```
newVertices.resize(modifiedVertices.size() +
+ (vertices.size() − replacedVertices.size()) − removedVertices.size())
newVertices[0..modifiedVertices.size()] = modifiedVertices
currentVertexIdx = modifiedVertices.size()
```

```
for v ∈ vertices do
    if v ∈ removedVertices then
        continue
    else if v ∈ replacedVertices then
        newVertexIdx[v.idx()] = replacedVertex[v.idx()]
    else
        newVertices[currentVertexIdx] = v
        newVertexIdx[v.idx()] = currentVertexIdx
        currentVertexIdx++
    end if
end for
newTriangles.resize(triangles.size() − removedTriangles.size())
currentTriangleIdx = 0
for t ∈ triangles do
    if t ∈ removedTriangles then
        continue
    else
        newTriangles[currentTriangleIdx] = newTriangle(newVertexIdx[V(t)])
        currentTriangleIdx++
    end if
end for
```

Отметим, что не везде, где мы оперируем такими объектами, как вершины, рёбра и треугольники, мы оперируем действительно ими, а не их индексами или иными ссылками. Работу с индексами мы отдельно подчеркнули лишь там, где это существенно в нашей реализации. Конкретная реализация, конечно, не так важна, однако мы решили привести свою для полноты изложения.

Куда более важно здесь объяснить, зачем нужно множество frozenVertices. Как было показано раньше, нарушения топологии могут возникать только тогда, когда $|V(v) \cap V(u)| > 2$, однако это верно лишь при удалении одного ребра. На практике же нам нужно удалять множество рёбер, и перестраивать сетку после каждого удаления было бы слишком затратно. В таком случае какое-либо ребро после стягивания может стать вершиной из $V(v) \cap V(u)$ для другого ребра, и эта вершина не будет учтена. Поэтому после стягивания ребра мы не трогаем рёбра, для которых оно может стать такой вершиной, вплоть до перестроения триангуляции.

6. Заключение

В заключение оценим сложность работы алгоритма и предоставим замеры времени и эффективности работы алгоритма. Сперва стоит сказать, что сложность алгоритма зависит от того, какие структуры мы используем для хранения

таких данных, как replacedVertices, frozenVertices и т. п. Чуть удобнее использовать структуры с поиском по ключу, такие, как красно-чёрные деревья. Заметим, что количество элементов k, которые в них будут храниться, является величиной порядка количества коротких рёбер и прямо пропорционально количеству вершин n триангуляции, но при этом значительно меньше его. Учитывая, что поиск по таким структурам осуществляется в среднем за $\mathcal{O}(\log n)$, а другие этапы алгоритма для каждого рассматриваемого ребра требуют $\mathcal{O}(1)$, наш алгоритм будет иметь асимптотическую сложность $\mathcal{O}(n \log n)$. Однако можно вместо этого использовать массивы, которые обладают константным временем доступа к элементу, хотя при этом нам придётся выделять больше памяти. Будем считать, что мы выбрали последний вариант и сложность нашего алгоритма $\mathcal{O}(n)$.

Теперь приведём сравнительные замеры времени работы нашего алгоритма и стандартного алгоритма, допускающего нарушение топологии, на моделях тора и двуполостного гиперболоида.

Таблица 1. Сравнение улучшенного и стандартного алгоритмов

Алг.	Модель	Исх. вершин	Исх. граней	Кон. вершин	Кон. граней	Время, с
Улучш.	Тор	16772	33544	4985	9970	0,22
Станд.	Тор	16772	33544	4990	9980	0,23
Улучш.	Гип.	56288	111140	18106	34776	0,80
Станд.	Гип.	56288	111140	18118	34800	0,82

Из табл. 1 видно, что наш алгоритм не уступает в эффективности стандартному. В заключение приведём изображение модели тора и части его триангуляции до и после упрощения (рис. 11).

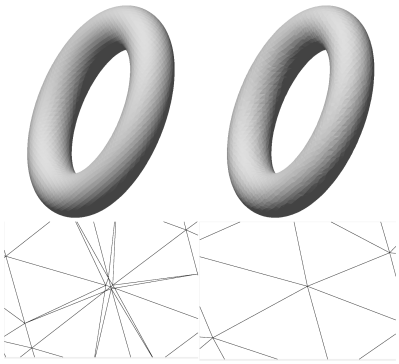

Рис. 11.

Литература

[1] Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B. Polygon Mesh Processing. — Peters, 2010. — P. 85—130.

[2] Chan S. L., Purisima E. O. A new tetrahedral tesselation scheme for isosurface generation // Computers Graphics. — 1998. — Vol. 22, no. 1. — P. 83—90.

[3] Ciarlet P., Lamour F. Does contraction preserve triangular meshes? // Numerical Algorithms. — 1996. — Vol. 13. — P. 201—223.

[4] Dey T. K., Edelsbrunner H., Guha S., Nekhayev D. V. Topology preserving edge contraction // Publ. Inst. Math., Nouv. Sér. — 1999. — Vol. 66 (80). — P. 23-45.

[5] Doi A., Koide A. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells // IEICE Trans. Inform. Systems. — 1991. — Vol. E74-D, no. 1. — P. 214—224.

[6] Garland M. Quadric-Based Polygonal Surface Simplification. — Carnegie Mellon Univ., 1998.

[7] Garland M. Multiresolution Modeling: Survey and Future Opportunities. — Eurographics, 1999.

[8] Garland M., Heckbert P. S. Surface simplification using quadric error metrics // SIGGRAPH'97: Proc. of the 24th Annual Conf. on Computer Graphics and Interactive Techniques. — 1997. — P. 209—216.

[9] Heckbert P. S., Garland M. Survey of Polygonal Surface Simplification Algorithms. — SIGGRAPH'97 Course Notes, No. 25. — New York: ACM Press, 1997.

[10] Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W. Mesh optimization // SIGGRAPH'93: Proc. of the 20th Annual Conf. on Computer Graphics and Interactive Techniques. — 1993. — P. 19—26.

[11] Munz T. Curvature-Adaptive Remeshing with Feature Preservation of Manifold Triangle Meshes with Boundary. — Bournemouth Univ., 2015.

[12] Skala V. Precision of iso-surface extraction from volume data and visualization // ALGORITHMY 2000: 15th Conf. on Scientific Computing. — 2000. — P. 368—378.

[13] Song Y., Fellegara R., Iuricich F., De Floriani L. Efficient topology-aware simplification of large triangulated terrains // SIGSPATIAL'21: Proc. of the 29th Int. Conf. on Advances in Geographic Information Systems. — 2021. — P. 576—587.

[14] Talton J. O. A Short Survey of Mesh Simplification Algorithms. — Univ. Illinois at Urbana-Champaign, 2004.

[15] Treece G. M., Prager R. W., Gee A. H. Regularised marching tetrahedra: improved iso-surface extraction // Computers Graphics. — 1999. — Vol. 23, no. 4. — P. 583—598.