УДК 681.3

АЛГОРИТМЫ ОПТИМИЗАЦИИ ФАСЕТЧАТЫХ МОДЕЛЕЙ И МЕТОДЫ ИХ СЕТЕВОЙ ПЕРЕДАЧИ

Р. В. Федотов[1]

В статье описываются некоторые методы геометрической оптимизации фасетчатых моделей. Рассматриваются способы с различной степенью быстродействия и качества получаемых результатов. Приводится описание способа передачи 3D моделей по сети с заданной точностью и с плавным уровнем детализации. Описывается структура прогрессивной сетки и приводится качественное сравнение рассмотренных алгоритмов.

Введение. Задачи геометрического моделирования сложных технических объектов требуют большой детализации проекта, что приводит к значительному росту объема обрабатываемых данных. Эта проблема в рамках одного рабочего места проектировщика решается экстенсивным наращиванием ресурсов. При работе проектных групп, состоящих из нескольких проектировщиков, работающих на различных компьютерах, прямое экстенсивное наращивание проектных ресурсов вызывает серьезные затруднения, связанные с сетевой организацией работы над проектом. Обрабатывая трехмерную модель, ПК производит считывание ее с сервера проекта в ОЗУ. Поскольку скорость доступа к данным по сети ограничена, то чем более детализирована модель, тем больше времени занимает эта операция. В машиностроительном проектировании компонуемые детали образуют иерархическую структуру в дереве сборки. Это приводит к тому, что при открытии на "чтение/запись" файла сборки на чтение открываются все файлы, размещаемые в вершинах дерева, что существенно повышает требования к пропускной способности сети, которая определяет время доступа к данным и соответственно "комфортность" работы проектировщика.

Важно заметить, что при открытии детали на "чтение/запись" ее файл может открываться в режиме "только для чтения". Сокращение трафика в этом случае связано с сокращением объема передаваемого файла. Для сетевой организации проекта необходима возможность загрузки модели с заданной точностью. Эти задачи решаются при помощи алгоритмов оптимизации 3D моделей.

Использование оптимизированных моделей связано с разработкой алгоритмов "упаковки" и "распаковки" моделей. Такие алгоритмы должны не только сохранять точность, но и обеспечивать минимальный объем передаваемых данных, сокращая тем самым сетевой трафик и время доступа к данным, что обеспечивает группе проектировщиков "комфортные" условия работы.

Целью данной статьи является описание некоторых алгоритмов геометрической оптимизации и анализ их сравнительной эффективности в сетевой среде.

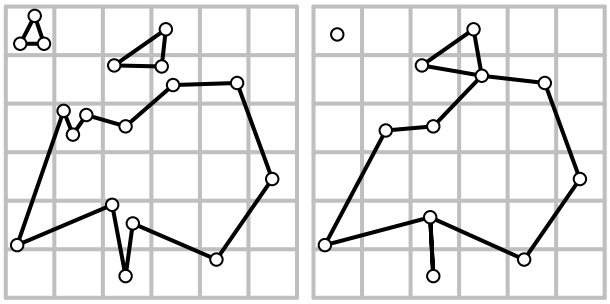
1. Алгоритмы геометрической оптимизации. Трехмерная модель в САПР представляется в виде конечного множества вершин V и множества элементов (ребер, треугольных граней), описывающих топологию объекта K и ссылающихся на множество V.

Задача геометрической оптимизации — уменьшение числа треугольных граней, описывающих модель, с сохранением (или с минимальным изменением) топологии. Это приводит к повышению быстродействия алгоритмов просчета изображения, упрощает манипуляции с моделью и сокращает используемую память и, как следствие, трафик при сетевом доступе.

На выходе алгоритма оптимизации имеем множества V_c и K_c. Множество V_c может быть подмножеством V или иметь с ним область пересечения в зависимости от того, перестраивает ли алгоритм модель или просто упрощает ее, удаляя вершины, менее всего влияющие на геометрию по заданным параметрам.

Наиболее быстрым алгоритмом геометрической оптимизации является алгоритм кластеризации вершин, предложенный в [5 – 7]. Метод состоит в объединении вершин, находящихся в одном кластере пространства, в одну. Кластеры представляют собой трехмерную сетку, т.е. координаты вершин фактически округляются с заданной точностью, а при совпадении нескольких вершин, после округления, они заменяются одной. Все ссылки множества K на коллапсируемые вершины заменяются ссылками на новую (рис. 1).

[1] Поволжская государственная академия, факультет экономики телекоммуникаций и информатики, 443010, г. Самара, ул. Льва Толстого, 23; e-mail: fedotov@hotbox.ru



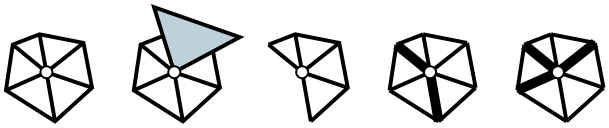
a *b*

Рис. 1. Стадии действия алгоритма кластеризации вершин

Большим преимуществом данного метода является простота и быстродействие, что дает возможность использовать его в режиме реального времени. Основным минусом является потеря первоначальной топологии объекта, что может привести к серьезным ошибкам, поэтому метод не может быть универсальным. Основной областью его применения являются задачи, связанные с визуализацией в режиме реального времени.

Более точно передает топологию модели алгоритм прореживания вершин, предложенный в [4]. Это многопроходный алгоритм, суть которого состоит в удалении на каждом проходе вершин, расположенных от усредненной плоскости соседних вершин на расстояние, которое меньше заданного.

Для сохранения топологии модели и минимизации повреждения формы при оптимизации все вершины модели делят на пять типов (рис. 2).



a *b* *c* *d* *e*

Рис. 2. Типы вершин при оптимизации методом прореживания вершин

a) Простая вершина окружена полным кругом граней и каждое ребро, содержащее эту вершину, граничит с двумя гранями.

b) Комплексная вершина окружена полным кругом граней (но имеется ребро, содержащее эту вершину, которое не граничит с двумя гранями) или не окружена полным кругом граней.

c) Пограничная вершина находится на границе объекта и окружена полукругом граней.

d) Если двугранный угол, содержащий простую вершину, велик или ребра, содержащие вершину, имеют разрыв дополнительных скалярных параметров (например, разрыв нормали), то такая вершина классифицируется как внутриреберная.

e) Если внутриреберная вершина содержит более двух ребер с разрывом или с большими двугранными углами, то вершина классифицируется как угловая.

При выполнении первого прохода алгоритма находятся вершины, подлежащие удалению в зависимости от величины расстояния d от вершины до усредненной плоскости, максимально близкой к соседним вершинам (рис. 3).

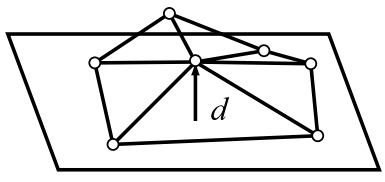


Рис. 3. Расстояние d от удаляемой вершины до средней плоскости

Для пограничных и внутриреберных вершин большую роль играют соседние пограничные или внутриреберные вершины, поэтому расстояние d измеряется до линии, соединяющей эти вершины (рис. 4).

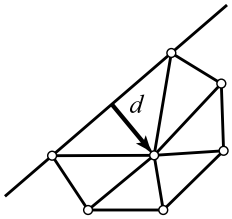


Рис. 4. Расстояние d для внутриреберных и граничных вершин

После удаления вершины образуется отверстие в трехмерном объекте, которое заполняется гранями. Процесс заполнения отверстия при удалении вершины называется триангуляцией [3, 6] и не является тривиальной задачей.

В [6] для триангуляции отверстия вводится рекурсивная функция, которая случайным образом делит (сечет) триангулируемое отверстие пополам. Если после такого сечения образуется группа из трех вершин, то строится грань, в противном случае продолжается процесс сечения (рис. 5).

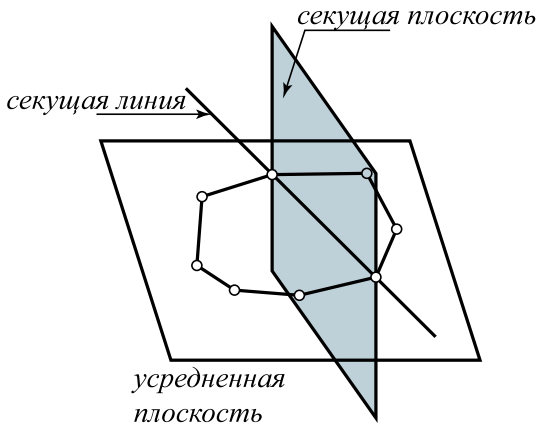


Рис. 5. Триангуляция отверстия рекурсивной функцией

При триангуляции ведется контроль на самопересечение образуемых граней. В случае обнаружения самопересечения процесс триангуляции повторяется с другим начальным рассечением. Задача триангуляции может и не иметь решения, поэтому количество повторений процедуры триангуляции ограничивается заранее. Если за заданное количество повторений решение не найдено, то вершина не удаляется.

После триангуляции на проходе n из множества V_{n-1} получают множество V_n, удаляя вершины. Из множества K_{n-1} удаляют все элементы, содержащие удаляемые вершины, и добавляют элементы, полученные в результате триангуляции, в результате чего получают множество K_n. Затем запускается следующий проход алгоритма. Алгоритм завершается, если для заданного расстояния d нет вершин, предназначенных для удаления.

Метод прореживания вершин — достаточно быстродействующий метод, но ресурсоемкий по памяти. Благодаря первоначальному разделению вершин на типы топология объекта сохраняется и не происходит сглаживания углов поверхности, что повышает универсальность метода и позволяет использовать его как для оптимизации гладких поверхностей, так и для моделей с резкими изломами, типичными для машиностроительного проектирования.

Описанные выше методы оптимизации сокращают множество V_0 так, что множество V_c оказывается его подмножеством. Такие методы обладают бóльшим быстродействием, поскольку независимо от критериев выбора удаляемых вершин выполняются только две операции: удаление вершины и триангуляция полученного отверстия. Алгоритмы, полностью перестраивающие V_0, более ресурсоемки, но дают более точные результаты.

Метод перестройки полигональной сетки предложен в [3] первоначально для обработки медицинских данных при оптимизации гладких поверхностей, не имеющих разрывов первой производной. Суть метода состоит в полной перестройке модели M_0 в модель M_c с меньшим количеством треугольных граней; для этого на поверхности модели M_0 произвольно размещают m_c вершин (где m_c — количество вершин оптимизированной модели, задаваемое пользователем). Затем на базе множества V_c и первоначальной геометрии строится множество K_c.

На первой стадии после случайного разбрасывания вершин выполняют процедуру “расслабления сетки”. Каждая вершина отталкивает соседние с силой, линейно убывающей с увеличением расстояния (линейный закон убывания используется для ускорения работы алгоритма), и смещается на суммарный отталкивающий вектор; при этом если вершина выталкивается за грань, то она переносится на соседнюю грань. Алгоритм “расслабления сетки” может выполнить несколько итераций для достижения оптимального результата.

Множество V_c будет более точно описывать первоначальную форму объекта, если при его построении распределять вершины не случайным образом, а в зависимости от кривизны поверхности: чем больше кривизна, тем больше вероятность появления вершины. Тогда более искривленные участки будут описываться большим количеством вершин.

Для нахождения кривизны поверхности в вершине в [3] предлагается следующий алгоритм: пусть P — заданная вершина, соединенная ребрами с вершинами $A_1, A_2, \ldots, A_n$, $\vec{N}$ — вектор нормали к поверхности в точке P, Θ_i — угол между вектором $\vec{N}$ и $\overrightarrow{P - A_i}$ (рис. 6). Тогда кривизна r поверхности находится следующим образом:

$$\Theta_i = \arccos(\vec{N}, \overrightarrow{P - A_i}), \quad r_i = \operatorname{tg}(\Theta_i)\frac{|P - A_i|}{2}, \quad r = \min(r_i).$$

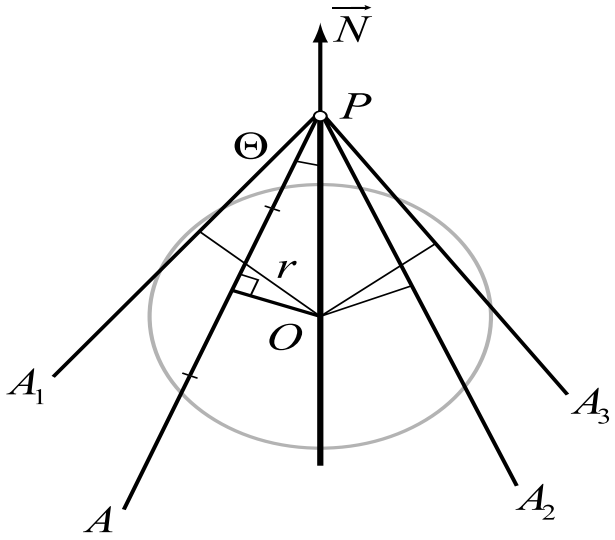


Рис. 6. Определение кривизны поверхности

После построения множества V_c необходимо построить множество K_c. Следует заметить, что алгоритм соединения гранями соседних вершин может давать серьезные ошибки (рис. 7). На рис. 7 a рассмо-

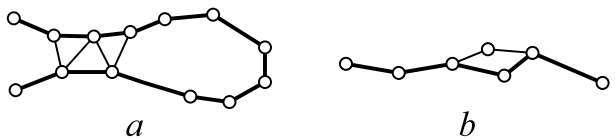


Рис. 7. Возможные ошибки при соединении соседних вершин

трен случай соединения гранями близлежащих вершин, не соединенных в первоначальной модели (тонкие линии). На рис. 7 b показан случай возникновения “пузыря” на первоначально непрерывной поверхности.

Для того чтобы избежать некорректно построенных граней, строится промежуточная модель M_{0C}, в которой $V_{0C} = V_0 + V_c$. Модель K_{0C} получают из K_0, убирая грань, если она содержит новые вершины, и дополняя новыми гранями. После этого последовательно удаляют старые вершины, триангулируя полученные отверстия. Метод триангуляции, использованный в данном алгоритме, подробно описан в [3]; перед удалением вершины все соседние точки проектируются на плоскость, касательную к поверхности в вершине. Если при таком построении проекций получена несамопересекающаяся замкнутая ломаная, то переходят к следующему шагу алгоритма, в противном случае используют другие плоскости проектирования. В [3] программная реализация алгоритма использует до 12 способов построения проекций. Если необходимая проекция не была получена, то текущая вершина не удаляется, т.к. ее невозможно удалить,

не повредив исходную топологию объекта. Если же проекция на плоскость получена, то триангуляция производится на плоскости, что гарантирует сохранение топологии объекта. После этого проверяют, не пересекаются ли полученные новые грани с другими гранями объекта. Эта проверка необходима, чтобы избежать соединения части поверхности с другой частью поверхности, находящейся перед или позади триангулируемого отверстия. На рис. 8 показан случай, когда удаление вершины R не возможно

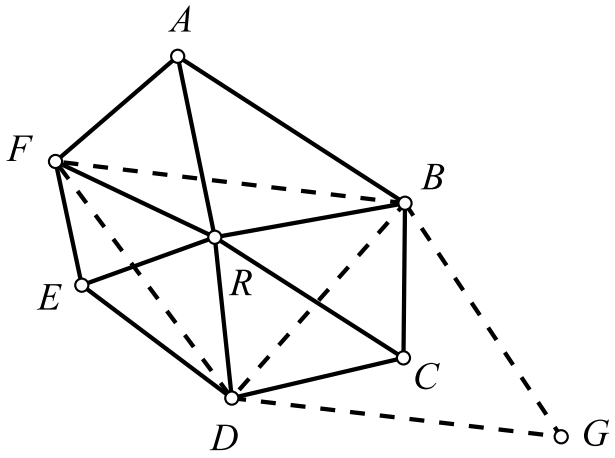


Рис. 8. Случай, когда триангуляция не возможна

из-за того, что после триангуляции отверстия $ABCDEF$ поверхность соприкоснется с гранью BFD, что нарушит топологию оптимизированной поверхности. В этом случае вершину также нельзя удалять из множества V_{0C}. Чтобы выполнить проверку на пересечение, необязательно проверять на пересечение все грани, полученные в результате триангуляции, со всеми гранями фасетчатой модели, достаточно проверить на пересечение только ближайшие грани.

Полученное таким образом множество K_c "чистят", триангулируя каждую вершину новой модели для получения лучшей сетчатой формы. Эта фаза подобна фазе удаления вершин за исключением того, что вершина не удаляется, а участвует в триангуляции, при которой соблюдаются все описанные выше правила. На рис. 9 иллюстрируется перестройка вершины Q.

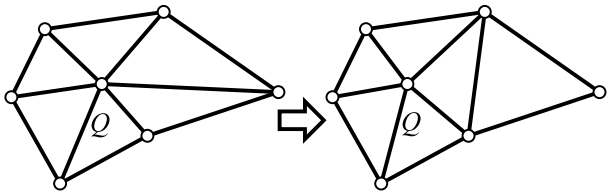


Рис. 9. Результат "чистки" вершин после триангуляции

На стадии удаления вершин и на стадии "чистки" краевые вершины обрабатываются особенным способом. При триангуляции проекция соседних вершин представляет собой незамкнутую ломаную, для триангуляции концы ломаной соединяют, так же поступают при "чистке" поверхности. Если обрабатываемая вершина является угловой, то ее не удаляют, а помещают в оптимизированную модель.

После проверки всех старых вершин на возможность удаления и чистки поверхности оптимизация может считаться завершенной.

Благодаря такому подходу к получению новых граней топология оптимизированной модели точно повторяет топологию первоначального объекта, но если в первоначальной поверхности были разрывы первой производной (острые углы и грани), то эти фрагменты при оптимизации станут "оплавленными". Из-за этого метод перестройки модели применим только для гладких поверхностей. В машиностроительном проектировании такой способ может использоваться при кодировании фасетчатых моделей из NURBS-объектов.

Рассматриваемый метод довольно сложен, не обладает быстродействием и достаточно требователен к ресурсам памяти. Метод неприменим для оптимизации в режиме реального времени.

В методах геометрической оптимизации одним из сложных критериев является критерий оценки соответствия оптимизированной модели первоначальному объекту. Фактически на основе подобного критерия можно создать многопроходный алгоритм: на каждом проходе проверяется соответствие перво-

начальному объекту; в случае допустимой точности осуществляется переход на следующий шаг, иначе повторяется текущий. Один из таких методов — метод минимизации энергетической функции [1]. Используемый в нем алгоритм оптимизации помимо первоначальной модели M_0 опирается на множество точек X, принадлежащих исходному идеальному объекту. Множество X может быть образовано из данных, полученных при сканировании реального трехмерного объекта, или же построено на основе точного описания объекта (твердотельного, параметризированного, NURBS).

Метод оптимизации фасетчатых сеток путем минимизации энергетической функции [1] разработан для оптимизации модели M_0 при наличии множества точек $X \in R^3$, принадлежащих исходному нефасетчатому объекту.

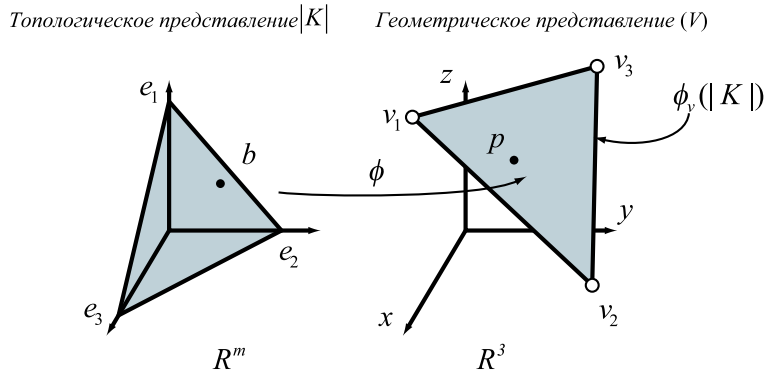


Рис. 10. Топологическое и геометрическое представления объекта

Фасетчатая модель может иметь геометрическое представление в трехмерном пространстве R^3, $v_i \in R^3$. Множество K, состоящее из подмножеств вида $\{1, \ldots, m\}$, может быть представлено как $|K|$ в топологическом пространстве R^m с базисными векторами $\{e_1, \ldots, e_m\}$. Функция перевода $\phi : R^m \to R^3$ считается заданной, причем i-й базисный вектор $e_i \in R^m$ переводится в вектор $v_i \in R^3$ (рис. 10).

Любая точка $p \in \phi_v(|K|)$ может быть представлена в топологической модели $|K|$. Векторы $b \in |K|$ и $p \in \phi_v(|K|)$ называют барицентрическими координатами вектора p. Барицентрические координаты вектора — комбинация стандартных базисных векторов $e_i \in R^m$, соответствующих вершинам грани [8]. Самое большое количество ненулевых барицентрических координат — три: это случай, когда одна точка принадлежит грани и две — ребру, а одна совпадает с вершиной модели.

Цель оптимизации — минимизация энергетической функции, зависящей от количества граней соответствия модели первоначальному объекту, содержащему множество X:

$$E(K, V) = E_{\text{dist}}(K, V) + E_{\text{rep}}(K) + E_{\text{spring}}(K, V).$$

Здесь первое слагаемое — сумма квадратов расстояния от точек $X = \{x_1, x_2, \ldots, x_n\}$ до сетки модели:

$$E_{\text{dist}}(K, V) = \sum_{i=1}^{n} d^2(x_i, \phi_v(|K|)).$$

Это слагаемое контролирует отклонение оптимизированной модели от первоначального объекта. Топологическая точность не оценивается. Топология модели M_0 будет сохраняться для всех итераций, поэтому в энергетической функции нет топологической составляющей.

E_{rep} — слагаемое, необходимое для уменьшения числа граней (собственно говоря, это и является задачей оптимизации):

$$E_{\text{rep}}(K) = c_{\text{rep}} m.$$

Коэффициент c_{rep} — параметр, вводимый пользователем алгоритма: чем больше c_{rep}, тем меньшее количество граней будет содержать модель. Заметим, что слагаемое E_{dist} увеличивает количество граней при наиболее точной подгонке к X. Коэффициент c_{rep} предназначен для минимизации E_{dist} за счет более оптимального расположения вершин, а не за счет их количества.

Для регуляризации поверхности вводится третье слагаемое:

$$E_{\text{spring}}(K, V) = \sum_{\{j,k\} \in K} k \, \|v_i - v_k\|^2.$$

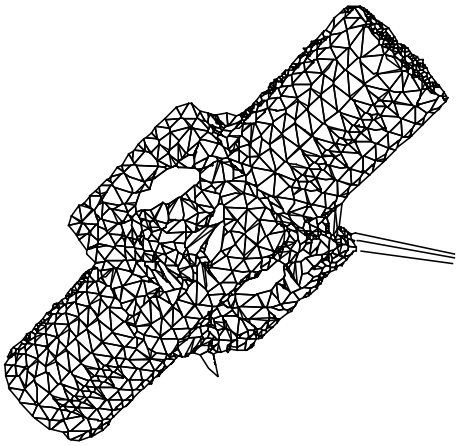


Рис. 11. Работа алгоритма без использования E_{spring}

Это слагаемое добавляет упругость ребрам с коэффициентом k. Слагаемое E_{spring} не сглаживает углы, а служит для устранения локальных скачков вершин (рис. 11).

Общий алгоритм оптимизации — это многостадийный алгоритм, который можно представить на псевдокоде следующим образом.

OptimizeMesh(K_0, V_0) {
  K:=K_0
  V:=OptimizeVertexPositions(K_0, V_0)
  - *Решение внешних минимизационных проблем*
  Repeat{
    (K', V'):=GenerateLegalMove(K, V)
    V'=OptimizeVertexPozition(K', V')
    if $E(K', V') < E(K, V)$ then
    $(K, V) := (K', V')$
    endif
  }until convergence
  return(K, V)
}
- *Решение внутренних оптимизационных*
-$E(K) := min_v E(K, V)$
- *для фиксированного множества* K
OptimizeVertexPozition(K, V) {
  Repeat {
    - *Вычисление барицентрических координат проецированием*
    B:=ProjectPoints(K, V)
    - *Минимизация* $E(K, V, B)$ *изменяя* V
    V:=ImproveVertexPositions(K, B)
  } until convergence
  return V
}
GenerateLegalMove(K, V) {
  выбрать легальное перемещение $K \Rightarrow K'$.
  Модифицировать V, для получения V' соответствующего K'
  Return (K', V')
}

Оптимизация модели при постоянном множестве K (Procedure Optimize–Vertex–Position) сводится к минимизации суммы $E_{\text{dist}}(K, V) + E_{\text{spring}}(K, V)$.

Для определения энергии дистанции $E_{\text{dist}}(K, V)$ необходимо вычислить все расстояния от точек x_i до $M = \phi_v(|K|)$. Чтобы вычислить каждое из этих расстояний, необходимо решить задачу минимизации

$$d^2(x_i, \phi_v(|K|)) = \min_{b_i \in |K|} \|x_i - \phi_v(b_i)\|^2 .$$

Задача минимизации энергетической функции $E(K, V)$ при фиксированном множестве K сводится к

минимизации новой функции

$$E(K,V,B)=\sum_{i=1}^{n}\|x_i-\phi_v(b_i)\|^2+E_{\text{spring}}(K,V)=\sum_{i=1}^{n}\|x_i-\phi_v(b_i)\|^2+\sum_{\{j,k\}\in K}k\,\|v_j-v_k\|^2,$$

где $V=\{v_1,\ldots,v_m\}$, $v_i\in R^3$ — вершины сетки и $B=\{b_1,\ldots,b_n\}$, $b_i\in|K|\subset R^m$.

Для решения этой задачи необходимо решить две подзадачи:

— при фиксированных вершинах M найти оптимальные барицентрические координаты векторов B на основе операции проектирования (procedure ProjectPoints);

— для найденных барицентрических координат вектора B найти оптимальные позиции вершин V (procedure ImproveVertexPosition).

Нахождение оптимальных барицентрических координат сводится к n задачам:

$$b_i=\arg\min_{\in|K|}\|x_i-\phi_v(b)\|.$$

Фактически для вычисления b_i находится проекция x_i на все грани M, а затем выбирается минимальное расстояние.

Минимизация $E(K,V,B)$ путем смены позиций вершин V сводится к минимизации $\left\|Av^1-d^1\right\|^2$ для каждой из трех координат (здесь v^1 — m-вектор, состоящий из первых координат точек v_i, и d^1 — $(n+e)$-вектор, в котором первые n элементов являются первыми координатами точек x_i, а последние e элементов нулевые, e — количество ребер в K, A — $(n+e)\times m$-матрица). Первые n строк — барицентрические координаты векторов b_i, оставшиеся e строк содержат по два ненулевых значения $\sqrt{k}$ и $-\sqrt{k}$ в столбцах, соответствующих концам ребер. Решение этой задачи описано в [9].

Для оптимизации $E(K)$ (procedure OptimizeMesh) изменяем множество K, совершая три элементарные операции над ребрами модели: стягивание ребра, разделение ребра и поворот ребра (рис. 12). Легальным перемещением называют перемещения, при которых не меняется топология K. Так же как и

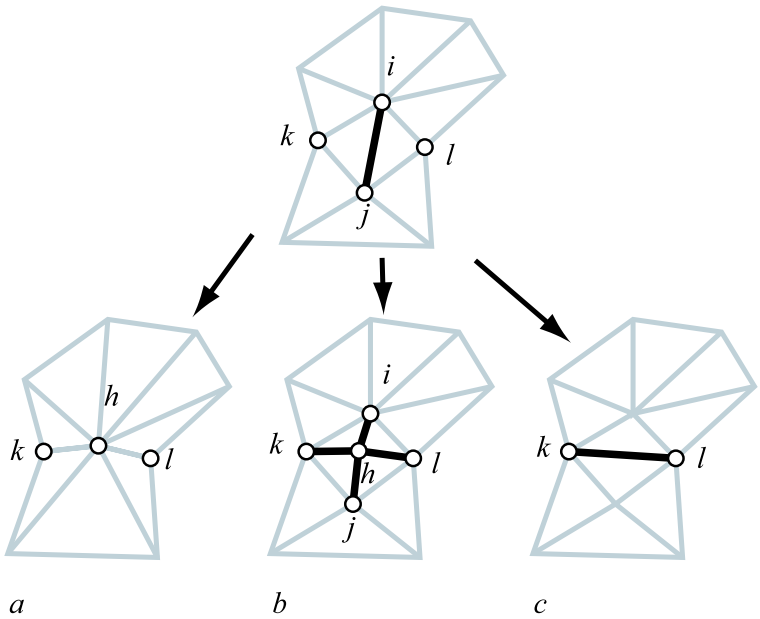


Рис. 12. Действия над ребрами модели

в других методах оптимизации, вводится понятие граничного ребра: если ребро $\{i,j\}$ граничит только с гранью $\{i,j,k\}\in K$, то такое ребро называется пограничным; вершина $\{j\}$ называется граничной, если она входит в граничное ребро $\{i,j\}$.

Изменение $K\Rightarrow K'$ путем стягивания ребра $\{i,j\}\in K$ является легальным действием, если выполняются следующие условия:

— если вершина $\{k\}$ — смежная с $\{i\}$ и $\{j\}$, то $\{i,j,k\}$ — грань K;

— если $\{i\}$ и $\{j\}$ — граничные вершины, то $\{i,j\}$ — граничное ребро;

— если K содержит более четырех вершин, когда $\{i\}$ и $\{j\}$ не граничные вершины, или если K содержит более трех вершин, когда $\{i\}$ и $\{j\}$ пограничные вершины.

Изменение $K \Rightarrow K'$ путем вращения ребра $\{i,j\} \in K$, $\{k,l\} \in K'$, является легальным действием, только если $\{k,l\} \notin K$. Наша цель — получить при помощи легальных действий такое K', чтобы $E(K') < E(K)$. Если это достигнуто случайными легальными действиями, то заменяем K на K' и повторяем итерацию.

Метод минимизации энергетической функции дает в результате корректную оптимизированную модель. При оптимизации учитывается первоначальная топология объекта и точная геометрия исходного "идеального" объекта. Данный алгоритм универсален и применим для оптимизации широкого спектра различных моделей. Основным минусом является его низкое быстродействие.

2. Оптимизация модели с учетом передачи по сети. В данном разделе рассматривается возможность решения задачи передачи оптимизированной модели по сети с заданной точностью с учетом пакетной передачи данных. Во время передачи данных получатель уже должен иметь грубодетализированную модель, детальность которой с каждым новым пакетом возрастает до заданной точности. Подобные алгоритмы реализованы в стандарте JPG для сжатия плоских изображений. Фактически речь идет об организации модели с переменным уровнем детализации (LOD). Способ организации модели с переменным LOD был описан в [2] и получил название прогрессивной сетки.

Выше был рассмотрен метод минимизации энергетической функции, в котором с ребрами производились три операции (стягивание ребра, деление ребра и поворот ребра). В прогрессивной сетке используется только операция стягивания ребра (рис. 13). На рис. 13 показано стягивание ребра $ecol\,\{v_s, v_t\}$ и

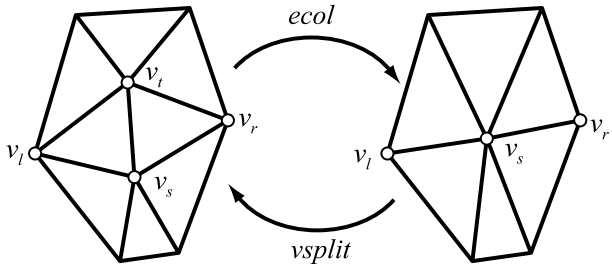


Рис. 13. Стягивание ребра и разбиение вершины

замена двух вершин v_s, v_t одной ν_s. Вершина ν_s и грани $\{v_s, v_t, v_l\}$ и $\{v_t, v_s, v_r\}$ убраны из модели. Таким образом, любую начальную модель $M_0 = M^n$ можно привести последовательным стягиванием ребер к модели M^0:

$$(M_0 = M^n) \xrightarrow{ecol_{n-1}} \cdots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0. \tag{1}$$

Мы получили последовательность моделей $M^i, j < n$ с убывающим LOD.

Пусть m_0 — количество вершин модели M^0 и пусть модель M^i задана множеством вершин $V^i = \{v_1, v_2, \ldots, v_{m_0+1}\}$. Тогда ребро $\{v_{s_i}, v_{m_0+i+1}\}$ стягивается процедурой $ecol_i$ (рис. 14 а). Так как после стягивания образуемая вершина может иметь различное положение, обозначим позицию v_j в M^i через ν^i_j. Процедура, обратная ecol, — это процедура разбиения вершины vsplit (рис. 13). Процедура разбиения вершины $vsplit\{s, l, r, t, A\}$ добавляет вершину v_t около v_s и две грани $\{v_s, v_t, v_l\}$ и $\{v_t, v_s, v_r\}$ (если ребро $\{v_s, v_t\}$ граничное, т.е. $v_r = 0$, то добавляется только одна грань); положение v_s также меняется. Другими словами, мы можем получить обратную последовательность:

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \cdots \xrightarrow{vsplit_{n-1}} (M_0 = M^n).$$

Каждое действие vsplit — это добавление данных к модели. Способ организации данных, состоящий из первоначальной модели M^0 и последовательности $\{vsplit_0, vsplit_1, \ldots, vsplit_{n-1}\}$, называется прогрессивной сеткой.

Для плавного изменения $M^i \to M^{i+1}$ при выполнении $vsplit_i(s_i, l_i, r_i, A_i) = (\nu^{i+1}_{s_i}, \nu^{i+1}_{m_0+i+1})$ мы создаем модель $M^G(\alpha)$ с переменным параметром $0 \leqslant \alpha \leqslant 1$, такую, что $M^G(0)$ имеет вид M^i, а $M^G(1)$ — вид M^{i+1}. Фактически $M^G(1) = M^{i+1}$ по определению:

$$M^G(\alpha) = (K^{i+1}, V^G(\alpha)).$$

В сущности выполняется линейная интерполяция двух вершин, соединенных ребром из $v_{s_i} \in M^i$, в вер-

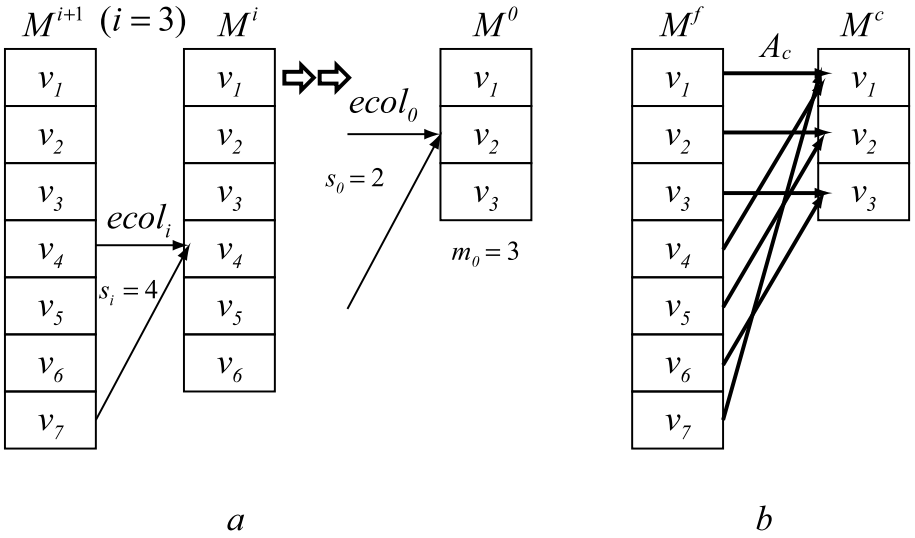


Рис. 14. Структура прогрессивной сетки

шины $v_{s_i}, v_{m_0+i+1} \in M^{i+1}$:

$$\nu_j^G(\alpha) = \begin{cases} (\alpha)\nu_j^{i+1} + (1-\alpha)\nu_{s_i}^i, & j \in \{s_i, m_0+i+1\}, \\ \nu_j^{i+1} = \nu_j^i, & j \notin \{s_i, m_0+i+1\}. \end{cases}$$

Используя этот метод в приложениях, можно перейти к $M^i \to M^{i+1}$ без всяких видимых швов.

Плавная интерполяция также возможна между двумя любыми сетками, входящими в прогрессивную сетку. Пусть M^f — более высоко детализированная модель, чем M^c, т.е. $0 \leqslant c < l \leqslant n$. Тогда M^f — это результат действия последовательности процедур $vsplit$ на M^c. Каждая вершина M^f сжимается в одну из вершин M^c. Картой таких перемещений служит A^c (рис. 14b), т.е. каждая вершина v_j в M^f соответствует $v_{A^c(j)}$ в M^c, где

$$A^c(j) = \begin{cases} j, & j \leqslant m_0 + c, \\ A^c(s_{j-m_0-1}), & j > m_0 + c. \end{cases}$$

Как и в предыдущем случае, определим $M^G(\alpha)$ следующим образом: $M^G(0)$ имеет вид M^c и $M^G(1) = M^f$, $M^G(\alpha) = (K^f, V^G(\alpha))$; тогда координаты вершин определяются из соотношения

$$\nu_j^G(\alpha) = (\alpha)\nu_j^f + (1-\alpha)\nu_{A^c(j)}^c.$$

Таким образом, возможно плавное изменение уровня детализации без швов и скачков. Для передачи по сети с плавным повышением уровня детализации сначала передается модель M^0, а затем передается последовательность пакетов $vsplit$. В результате модель плавно детализируется. Если за единицу времени приходит только один $vsplit$-пакет, то интерполяция выполняется для одной вершины; если же скорость позволяет передавать сразу несколько пакетов, то интерполяция выполняется по описанному выше алгоритму. Перед началом передачи файла пользователь может запросить ограниченное число $vsplit$-пакетов для получения модели с необходимой ему точностью. Преимущество способа состоит также в том, что детализацию модели всегда можно повысить, запросив недостающие $vsplit$-пакеты.

Предварительная оптимизация для подготовки прогрессивной сетки может проводиться любым алгоритмом оптимизации. Максимально быстрый результат дают алгоритмы, сокращающие число вершин. Многостадийность в данном случае работает для формирования пакетов $vsplit$. К таким методам относится метод кластеризации пространства и метод прореживания сетки [4–6]. Способ организации модели с переменным уровнем детализации, описанный в [3], основан не на последовательном приращении по вершине, а на разделении вершин на группы от самых низко детализированных до высоко детализированных. Проблемы топологической интерполяции решаются сложным способом, при котором каждый элемент низко детализированной модели M^0 содержит дерево элементов, перекрывающих его в более высоко детализированных стадиях. Плавность перехода обеспечивается построением промежуточных граней, не присутствующих ни в начальной, ни в конечной модели. Тем не менее, данные, полученные при оптимизации этим способом, можно использовать для формирования прогрессивной сетки.

Оптимизация, описанная в [2] для получения прогрессивной сетки, основана на способе [1] минимизации энергетической функции в следующей реализации: применяется алгоритм оптимизации скалярных параметров и при оптимизации множества K используются не три операции над ребрами (стягивание, разбиение, поворот), а только стягивание с тремя возможными положениями вершины после стягивания: $\nu s^i = (1-\alpha)\nu_s^{i+1} + (\alpha)\nu_t^{i+1}$ при $\alpha = \{0, 1/2, 1\}$.

3. Заключение. Задачи оптимизации моделей имеют множество способов корректного решения. Существуют методы для решения ее в системах реального времени, необходимые для интерактивной графики, а также алгоритмы, более точные для решения задач САПР и точного моделирования. Сравнительные данные приведены в таблице.

	Кластеризация вершин	Прореживание вершин	Перестройка модели	Минимизация энергетической функции
Быстродействие	Высокое	Высокое	Не высокое	Низкое
Топологическая точность	Топологическое несоответствие	Полное топологическое соответствие	Точное топологическое соответствие, оплавление углов	Самый корректный результат
Универсальность	Только для визуализации	Универсален	Только для гладких поверхностей	Высокая универсальность
Использование памяти	Не требователен	Большие затраты памяти	Большие затраты памяти	Большие затраты памяти
Дополнительные данные	Не использует	Не использует	Возможно использование данных о точной модели	Использует данные о точной модели.
Простота алгоритма	Простой	Многоитерационный алгоритм	Сложный комплексный алгоритм	Сложный алгоритм. Оптимизируется для прогрессивной сетки

Следует заметить, что на данном этапе разработки алгоритмов оптимизации развитие получили в основном универсальные алгоритмы, обрабатывающие любую абстрактную модель с любой топологией и первоначальной формой. Исключением может служить только метод перестройки модели, работающий для гладких поверхностей.

Актуальна разработка алгоритмов кодирования с учетом природы первоначальной "идеальной" модели. Возможно, совместное использование данных подходов приведет к лучшим результатам.

СПИСОК ЛИТЕРАТУРЫ

1. *Hoppe H., DeRose T., Duchamp T., McDonald J., and Stuetzle W.* Mesh optimization // SIGGRAPH 93. 1993. 19–26.
2. *Hoppe H.* Progressive meshes // SIGGRAPH 96. 1996. 99–108.
3. *Turk G.* Re-tiling polygonal surfaces // SIGGRAPH 92. 1992. 55–64.
4. *Shroeder W., Zarge J., and Lorensen W.* Decimation of triangle meshes // SIGGRAPH 92. 1992. 65–70.
5. *Rossignac J.* Geometric simplification and compression // GVU Center and College of Computing Georgia Institute of Technology. SIGGRAPH 97. 1997. 74–81.
6. *Rossignac J., Borrel P.* Multi-resolution 3D approximations for rendering complex scenes // Geometric Modeling in Computer Graphics, Springer Verlag. Eds. B. Falcidieno and T.L. Kunii. Genova, Italy. June 28–July 2, 1993. 455–465
7. *Low K-L., Tan T-S.* Model simplification using vertex-clustering (to appear).
8. *Warren J.* Barycentrie coordinates for convex polytopes // Departament of Computer Science, Rice University, 1996.
9. *Golub G. and Van Loan C.* Matrix Computations. Baltimore: John Hopkins University Press, 1989.

Поступила в редакцию
06.06.2003