DOI: https://doi.org/10.15688/mpcm.jvolsu.2025.2.2

УДК 514.76.3, 517.95
ББК 22.19, 22.151

Дата поступления статьи: 21.02.2025
Дата принятия статьи: 05.03.2025

ЦЕПНОЙ АЛГОРИТМ СЖАТИЯ 3D-МОДЕЛЕЙ

Владимир Александрович Клячин

Доктор физико-математических наук, заведующий кафедрой
компьютерных наук и экспериментальной математики,
Волгоградский государственный университет
klchnv@mail.ru, klyachin.va@volsu.ru
https://orcid.org/0000-0003-1922-7849
просп. Университетский, 100, 400062 г. Волгоград, Российская Федерация

Аннотация. В статье подробно излагается алгоритм сжатия информации о геометрическом строении трехмерных пространственных моделей и многомерных триангуляций, основанный на использовании смежности граней. Этот алгоритм преобразует набор граней 3D-модели в список цепочек (list of chains), последовательно расположенных в пространстве и смежных между собой. Сжатие информации происходит за счет отсутствия дублирования номеров вершин, образующих грани модели. Описанный в статье алгоритм состоит из трех основных частей. В первой части по множеству граней модели строится специальный граф граней – ребра графа соответствуют смежным граням. Используя алгоритм обхода вершин графа в глубину, этот граф разбивается на простые цепи. Вторая часть алгоритма преобразует каждую цепь графа в последовательность номеров вершин, участвующих в формировании граней этой цепочки. Третья часть алгоритма призвана выполнять обратное действие – переводить построенную последовательность номеров вершин обратно в наборы кортежей, состоящих из номеров вершин, соответствующих граням 3D-модели. Указанный алгоритм распространен и на случай пространственных триангуляций полигональных областей. Программная реализация алгоритма для частного случая 3D-моделей выполнена в виде встраиваемых модулей в программу Blender. Архивы модулей свободно доступны в репозитории автора статьи по адресу: https://github.com/KlyachinVA/LocFile.

Ключевые слова: триангуляция, граф модели, цепь граней, обход графа в глубину, смежность граней.

Введение

В настоящей статье предлагается алгоритм сжатия геометрических данных для представления 3D-моделей произвольной топологии. Надо сказать, что тема эта достаточно актуальна [9; 10] в связи с нарастающей сложностью 3D-моделей, и соответствующая задача имеет ряд готовых решений. Например, on-line сервис Free online app for Wavefront OBJ files compression [11] предлагает сжать модели и представить их в форматах OBJ, 3DS, STL в более компактном виде. Однако простые эксперименты, выполненные автором для сложных моделей неправильной формы, показывают, что при выполнении операции сжатия OBJ -> OBJ коэффициент сжатия практически равен 1. Отметим работу [7], в которой предлагается алгоритм сжатия, основанный на идее использования в моделях повторяющихся примитивов, а также – присутствия на модели плоских элементов. Следует также упомянуть работу [8], в которой для сжатия в основу положен алгоритм квадратичной ошибки (Quadric Error Metrics Algorithm). Интересен также обзор [12], в котором вкратце описан алгоритм, похожий на представленный в нашей статье.

Идея алгоритма, представленного в настоящей работе, и краткое его описание были даны также в работах [1; 3; 4]. В работе [4] рассмотрен произвольный случай триангуляций полигональных областей многомерного пространства, тогда как в работе [1] представлен алгоритм исключительно для двумерных полигональных поверхностей, однако не указаны оценки степени сжатия моделей. Более того, в этой работе рассматриваются два вида последовательности граней: веерные и полосные. В нашей статье описан алгоритм, не использующий структуру последовательности граней, и, в отличие от указанных выше работ, описан алгоритм разбиения 3D-модели на полосы – в нашей терминологии цепочки граней. Программная реализация описанного в настоящей статье алгоритма может быть полезна для представления геометрических объектов в сжатой форме в различных задачах, таких как, например, в задаче пространственной реконструкции объектов по их изображениям [2] или в задаче расчета формы равновесного состояния гиперупругого тела [6].

1. Представление 3D-моделей

Геометрическая структура 3D-моделей задается двумя последовательностями (списками, массивами) векторов. Первый список задает массив вершин модели

$$P = \{p_i = (x_i, y_i, z_i), i = 1, ..., N\}.$$

Второй список F составлен как список описания граней 3D-модели. Мы предполагаем, что грани являются треугольниками. Пусть M – число треугольных граней модели и F_j это j-я ее грань, $j = 1, ..., M$. Тогда

$$F_j = (n_1, n_2, n_3)$$

означает, что F_j – это пространственный треугольник с вершинами в точках $p_{n_1}, p_{n_2}, p_{n_3}$. Такое представление геометрической структуры 3D-моделей используется в формате OBJ. Описание элементов массива P в OBJ-файлах задается отдельными строками, начинающимися с символов «v» (от слова vertex – вершина), а описание элементов массива F – строками, начинающимися с символов «f» (от слова face – грань). Заметим,

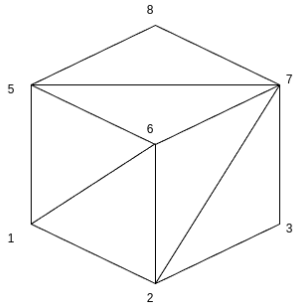

Рис. 1. Модель куба

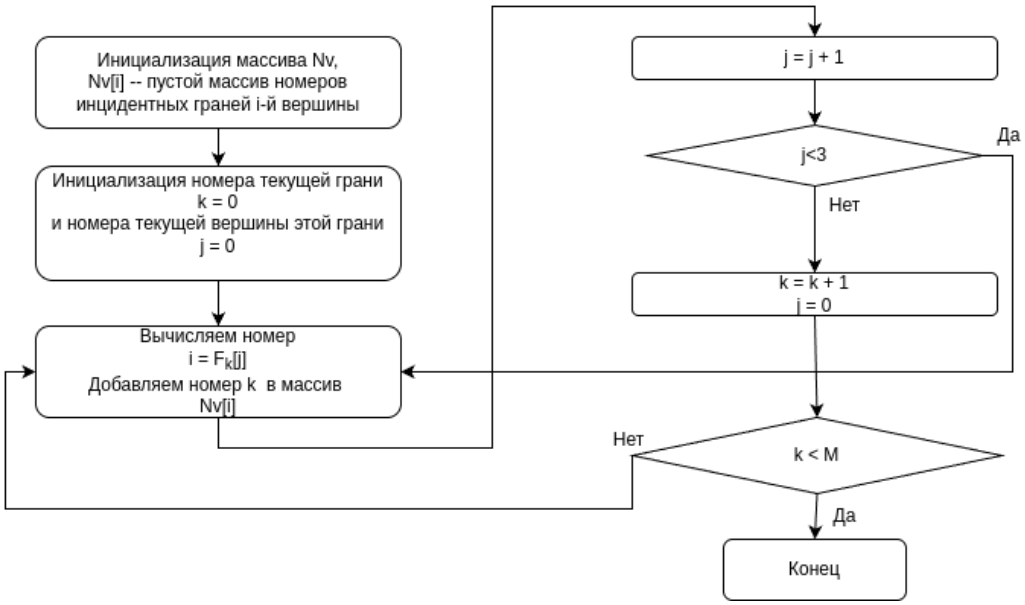

Рис. 2. Блок-схема алгоритма построения списков инцидентных граней

что для описания модели с M числом граней нам потребуется ровно $3M$ целочисленных номера.

Пример. Для задания куба $[0,1]^3 \subset \mathbb{R}^3$ (см. рис. 1) используем два массива

$$P = \{p_i = (x_i, y_i, z_i), x_i, y_i, z_i \in \{0,1\}, i = 1,..,8\},$$

$$F = \{(1,2,3),(2,3,4),(1,2,6),(1,6,5),(2,3,7),(2,7,6),$$
$$(3,4,8),(3,8,7),(4,1,5),(4,5,8),(5,6,7),(5,7,8)\}.$$

Во многих задачах по обработке 3D-моделей используется построение для каждой вершины списка ее инцидентных граней. На основе вышеописанного представления геометрии моделей несложно получить алгоритм такого построения. Он представлен на диаграмме рисунка 2.

2. Алгоритм сжатия

Как и многие алгоритмы сжатия, в представленном алгоритме используется избыточность информации. Заметим, что порядок перечисления граней в массиве F не важен. Но если мы будем использовать связность граней, то вместо 6 номеров $\{(n_1, n_2, n_3),$

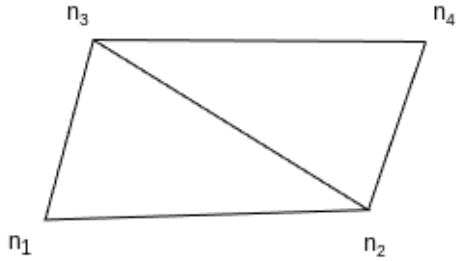

Рис. 3. Иллюстрация к алгоритму сжатия

$(n_2, n_3, n_4)\}$ для двух соседних (смежных) граней нам понадобится всего 4 $\{n_1, n_2, n_3, n_4\}$ (рис. 3).

Если мы свяжем по такому принципу в последовательную цепочку m граней, то, как не сложно вычислить, вместо $3 \cdot m$ номеров граней мы можем задействовать только $3 + m - 1 = m + 2$ номера. Предположим, что мы сможем все грани разместить в L цепочек и j-я цепочка будет содержать m_j граней. Тогда для описания каждой цепочки граней мы задействуем $m_j + 2$ целочисленных номера. Суммируя по всем цепочкам, получим

$$\sum_{j=1}^{L} m_j + 2 = M + 2L.$$

Таким образом, для описания граней мы задействуем $M + 2L$ номера вместо $3M$. Заметим, что при указанном способе записи граней получить доступ к отдельной грани можно только последовательно в каждой цепочке. Поэтому среднее время доступа к отдельной грани в j-й цепочке будет равно $c \cdot m_j$. Поскольку цепочки образованы также с помощью контейнера с последовательным доступом, то доступ к началу цепочки можно получить за среднее время $c \cdot L$. Если принять, что средняя длина цепочки равна M/L, получим, что полное среднее время доступа к отдельной грани равно

$$T(L) = c \cdot L + c\frac{M}{L}.$$

Минимальное значение этой функции равно

$$T_{min} = T(\sqrt{M}) = 2c\sqrt{M}.$$

Таким образом, оптимальное разбиение 3D-модели на цепочки мы получим при количестве цепочек $\sqrt{M}$ по $\sqrt{M}$ граней в каждой цепочке. При этом количество номеров для задания граней будет равно

$$M + 2L = M + 2\sqrt{M} = M\left(1 + \frac{2}{\sqrt{M}}\right).$$

Поэтому асимптотически, при $M \to +\infty$, предлагаемый способ записи граней модели будет в три раза экономней традиционного.

Замечание 1. Следует отметить, что при загрузке модели в какой-либо 3D-редактор, например Blender, соответствующая программа использует свое собственное ее представление. Загрузка модели в память подразумевает последовательный доступ к граням

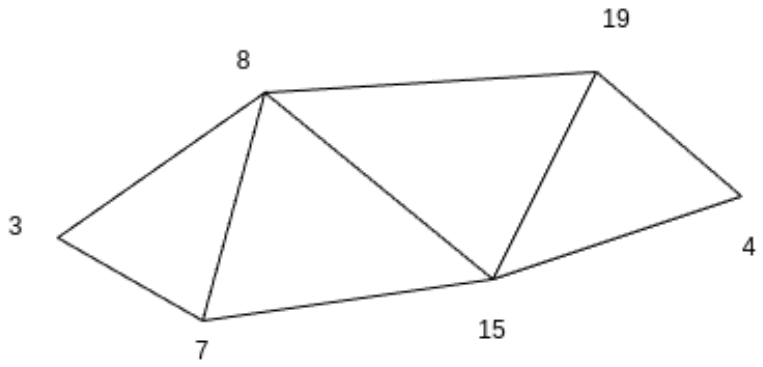

Рис. 4. Полоса граней

модели, независимо от способа их хранения – различается только порядок, в котором грани будут загружены. С этой точки зрения, задача уменьшения объема дискового пространства, который занимает файл с 3D-моделью, является более актуальной, чем задача ускорения доступа к граням модели.

Приступим к более детальному описанию способа записи цепочек. Заметим, что в работе [1] требуется согласованность ориентаций каждой грани с соседними. Это необходимо для того, чтобы правильно выделять тройку номеров очередной грани при просмотре цепочки номеров. При этом выделяют последовательности двух типов: последовательность типа «веер» и последовательность типа «полоса». При последовательности типа «веер» номера из последовательности номеров вершин чередуются в определенном порядке. Например, если последовательность граней такая, как на рисунке 4, то соответствующая последовательность номеров $3, 7, 8, 15, 19, 4$ легко преобразуется в последовательность троек номеров, описывающих грани: $\{(3, 7, 8), (7, 8, 15), (8, 15, 19), (15, 19, 4)\}$.

Если последовательность устроена как веер (см. рис. 5), то также имеется определенная закономерность получения троек, определяющих соответствующие грани. Именно, из последовательности номеров $7, 3, 8, 15, 19, 4$ находим последовательно тройки: $\{(7, 3, 8), (7, 8, 15), (7, 15, 19), (7, 19, 4)\}$. В соответствии с этими примерами, алгоритм из работы [1] выявляет последовательности этих двух типов и записывает их указанным выше способом.

Мы предлагаем алгоритм, в котором не разделяем эти два случая и обрабатываем последовательности граней независимо от их типа. Для иллюстрации идеи рассмотрим пример последовательности граней на рисунке 6. По последовательности номеров вершин $7, 3, 8, 15, 11, 19, 4, 5, 14, 1$ сложно собрать тройки, определяющие соответствующие грани. Воспользуемся таким правилом. Первую тройку запишем таким образом, чтобы первый номер соответствовал вершине, противоположной общему ребру следующей грани. В нашем случае это тройка $(7, 3, 8)$. Затем будем поступать так. Если следующее общее ребро лежит напротив вершины с номером 3, то в последовательность добавляем номер следующей вершины – номер 15. Поступая таким образом, получим последовательность $7, 3, 8, 15, 11, 19, 4$. Это дает способ построения троек номеров вершин граней, как и для последовательностей типа полосы: $\{(7, 3, 8), (3, 8, 15), (8, 15, 11), (15, 11, 19), (11, 19, 4)\}$. А вот при добавлении вершины с номером 5 заметим, что, действуя по такой же схеме, мы получим тройку $(19, 4, 5)$, которая не определяет никакой грани модели. Вместо этой тройки нужно использовать тройку $(11, 4, 5)$. Другими словами, в этой ситуации мы тройку собираем из нового номера из последовательности, последнего

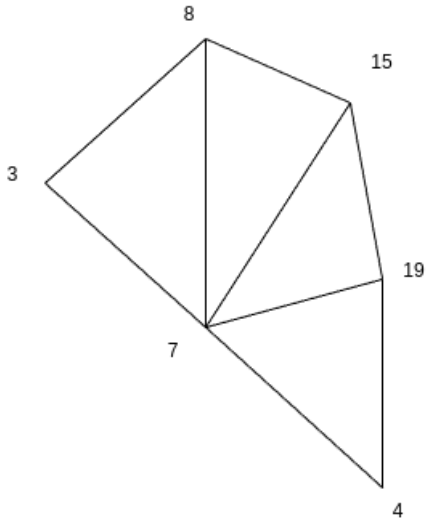

Рис. 5. Веер граней

номера последовательности и недостающего номера из предыдущей тройки. Чтобы последовательность управляла таким алгоритмом, мы эту ситуацию в последовательности будем отмечать отрицательными значениями – количество номеров от этого не изменится. Таким образом, мы получим два алгоритма – один строит последовательность номеров по связной цепочке граней, а другой, наоборот, – строит цепочку граней по последовательности.

Опишем алгоритм преобразования цепочки граней в последовательность номеров вершин в общих обозначениях.

- **Инициализация.**

  Извлекаем две тройки номеров вершин первой и второй грани цепочки. Для тройки первой грани находим номер вершины n_0, которая не является инцидентной второй грани. Помещаем первую тройку в формируемую последовательность номеров в порядке – сначала n_0, а потом остальные две.

- **Общий шаг цикла.**

  Пусть $n_0, n_1, ..., n_k$ – текущая сформированная последовательность номеров вершин цепочки граней, $T = (m_1, m_2, m_3)$ – тройка, соответствующая предыдущему шагу алгоритма, а $T' = (k_1, k_2, k_3)$ – вновь рассматриваемая тройка следующей грани. Предположим, что

  $$k_3 = T' \setminus (T' \cap T)$$

  есть номер вновь добавляемой вершины. Если равны множества

  $$\{n_k, n_{k-1}, k_3\} = \{k_1, k_2, k_3\},$$

  то в последовательность добавляем k_3. Если же равны такие множества

  $$\{n_k, n_{k-2}, k_3\} = \{k_1, k_2, k_3\},$$

  то в последовательность добавляем $-k_3$.

Теперь рассмотрим алгоритм, который последовательность номеров будет преобразовывать в последовательность троек смежных граней.

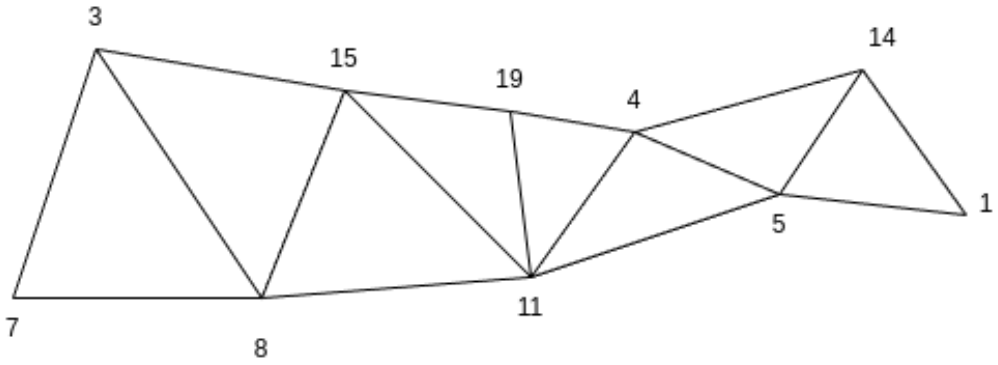

Рис. 6. Иллюстрация к алгоритму записи цепочек граней

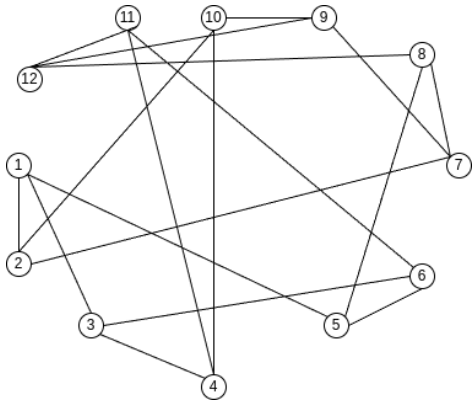

Рис. 7. Пример графа граней модели куба

- **Инициализация.**

  Создается пустой список и в него добавляется кортеж (n_0, n_1, n_2) из трех первых элементов последовательности.

- **Общий шаг цикла.**

  Пусть на предыдущем шаге сконструирован кортеж (m_1, m_2, m_3). Извлекаем следующий элемент последовательности n_k. Если $n_k > 0$, то в список добавляется кортеж (m_2, m_3, n_k). Если $n_k < 0$, то в список добавляется кортеж $(m_1, m_3, -n_k)$.

3. Алгоритм конструирования цепочек на основе графа граней

Для построения списка цепочек граней нам понадобится предварительно сконструировать вспомогательный граф граней.

Пусть пара (P, F) задает геометрическую структуру 3D-модели S с N вершинами и M треугольными гранями. Построим *граф* $G(S)$ *граней* модели S, полагая в качестве множества V его вершин множество F граней модели S. Две грани $f, g \in F$ образуют ребро $e = (f, g) \in E$ графа $G(S)$, если эти грани смежны в модели S. На рисунке 7 изображен пример такого графа для модели куба с рисунка 1.

Для получения списка цепочек граней необходимо представить граф $G(S)$ в виде набора простых цепей. Чтобы это сделать, воспользуемся алгоритмом обхода вершин графа в глубину. Предварительно для каждой вершины v графа вычислим список $N[v]$ номеров вершин, смежных с вершиной v.

- **Инициализация.**

  Создаем пустой список C цепей, создаем пустой список W пройденных вершин графа, создаем пустой список U номеров вершин для текущей цепи и создаем пустой стек Q. Выбираем первую вершину v графа и помещаем ее в стек.

- **Общий шаг цикла.**

  1) Если стек Q пуст – завершить цикл.
  2) Извлекаем из вершины стека номер очередной вершины $v = Q.pop()$.
  3) Проверяем, если $v \in W$, то переходим к следующей итерации цикла.
  4) Проверяем, если список U не пуст и для вершины u, последней добавленной в U, выполнено
  $$v \notin N[u],$$
  добавляем цепочку U в список C, создаем новую пустую цепочку U и вносим в нее вершину v.
  5) Если же $v \in N[u]$, то просто добавляем v в текущую цепочку U.
  6) Если оказалось, что цепочка U пустая, то добавляем в нее вершину v.
  7) Добавляем вершину в список пройденных вершин W.
  8) Перебираем все вершины $u \in N[v]$, и если $u \notin W$, добавляем ее в стек $Q.push(u)$.

- **После завершения цикла.**

  Завершив цикл, добавим последнюю цепочку U в список C.

4. Многомерный случай

В работах [7; 9; 10; 12] рассматривался случай двумерных граней 3D-моделей. Однако задача сжатия триангуляций может быть полезной и в пространствах большей размерности. Например, для хранения результатов расчетов равновесных форм деформированных гиперупругих тел в нелинейной теории упругости [4–6]. Здесь мы приведем описание алгоритма сжатия произвольных триангуляций в пространстве $\mathbb{R}^n$. При этом мы ограничиваемся случаем, когда объединение всех симплексов триангуляции образует связное множество – многогранник.

Пусть $P = \{p_i,\ i = 1, ..., N\}$ – вершины триангуляции T в пространстве $\mathbb{R}^n, n \geq 1$. Симплексы такой триангуляции задаются списком кортежей вида

$$(m_0^j, m_1^j, ..., m_n^j)\ j = 1, ..., M$$

номеров вершин симплексов. Граф граней $G = G(T)$ триангуляции T определяется также, как и выше. В качестве вершин используются грани триангуляции, а каждое ребро соответствует двум смежным граням. Алгоритм построения этого графа для многомерной триангуляции не сильно отличается от такового для 3D-моделей, и поэтому мы его не будем приводить. Алгоритм построения цепочек граней сводится к решению задачи на абстрактном графе и вообще не зависит от происхождения графа. Уточнения требуют только два алгоритма: алгоритм преобразования цепочки грани в последовательность номеров и обратный алгоритм преобразования последовательности номеров вершин в цепочку граней.

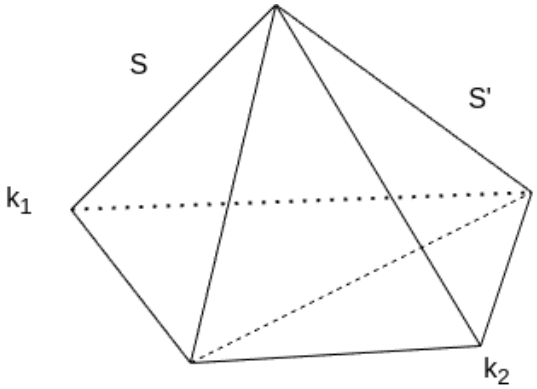

Рис. 8. К общему шагу алгоритма

Ниже представлен алгоритм записи цепочки симплексов в последовательность номеров.

- **Инициализация.**

  Через S будем обозначать текущий вспомогательный симплекс, а через S' – очередной обрабатываемый симплекс из цепочки (рис. 8). Инициализируем S первым, а S' вторым симплексом из цепочки. В последовательность номеров H запишем значения из S.

- **Общий шаг цикла.** Вычисляем номера

$$k_1 = S \setminus S', \quad k_2 = S' \setminus S.$$

  Поскольку симплексы S и S' всегда смежны (так как они последовательно расположены в одной цепочке), то номера k_1, k_2 однозначно определены. Пусть k обозначает индекс номера k_1 в S.

  Заменим в наборе номеров симплекса S номер k_1 на k_2 и добавим значение $k \cdot N + k_2$ в последовательность H. После этого извлекаем из цепочки очередной симплекс S'.

Замечание 2. В описанном выше алгоритме для указания индекса $k \in \{0, ..., n\}$ вершины симплекса, не принадлежащего общей грани смежных симплексов, используется смещение номера вершины k_2 на величину $k \cdot N$. Вместо этого можно было бы записывать в последовательность H само число k. Это выгоднее с точки зрения экономии дискового пространства, если результат записывается в текстовый формат. Если же предполагается последовательности номеров записывать в бинарном виде в файл, то экономнее будет описанный в алгоритме прием. В обратном алгоритме преобразования мы покажем, как это смещение используется для восстановления цепочки симплексов.

- **Инициализация.**

  Из последовательности H извлекаем первые $n+1$ номеров. Они образуют вспомогательный симплекс S. Создаем пустую цепочку симплексов C и добавляем в нее симплекс S.

- **Общий шаг алгоритма.**

Извлекаем из последовательности очередной номер l и вычисляем индекс, полагая

$$k = \left[\frac{l}{N}\right].$$

Далее, используя этот индекс, вычисляем номер вершины по формуле

$$k_1 = l - k \cdot N.$$

Формируем новый симплекс S', заменяя в S по индексу k номер на k_1. Добавляем S' в C и делаем S' текущим $S = S'$.

5. Программная реализация

Описанные выше алгоритмы в части 3D-моделей реализованы в виде встраиваемых модулей в пакет Blender на языке программирования Python. Эти модули в виде zip-архивов можно свободно скачать из репозитория автора статьи по адресу: https://github.com/KlyachinVA/LocFile. Чтобы воспользоваться модулями, нужно их установить для пакета Blender следующим образом. Необходимо в программе Blender выбрать пункт меню *Edit -> Preferences* и далее выбрать вкладку *Add-ons*. На появившейся панели найти инструмент *Install from Disk* и, не распаковывая zip-архивы скаченных модулей, установить их. После этого в меню *File -> Export* появится пункт *Export to loc file*. Чтобы сохранить модель в формат LOC (List Of Chains), необходимо, чтобы модель была связна, а ее грани были треугольными. Последнее легко добиться с помощью модификатора *Triangulation*. Второй модуль loc_file_import.zip реализует загрузку модели в формате LOC в окно 3D-вида программы Blender. Модуль доступен через меню *File -> Import -> Loc Data Import*.

СПИСОК ЛИТЕРАТУРЫ

1. Капустина, С. В. Конвертация 3D-модели методом оптимизирующего сжатия / С. В. Капустина, М. С. Клюев. — Материалы конференции MIT-2011. — URL: https://conf.nsc.ru/files/conferences/MIT-2011/fulltext/50144/56662/kapustina.pdf

2. Клячин, В. А. Алгоритм реконструкции трехмерных объектов по двум изображениям / В. А. Клячин, А. Ю. Кузьменко // Математическая физика и компьютерное моделирование. — 2024. — Т. 27, № 2. — С. 61–70. — DOI: https://doi.org/10.15688/mpcm.jvolsu.2024.2.5

3. Клячин, В. А. Метод триангуляции для приближенного решения вариационных задач нелинейной теории упругости / В. А. Клячин, В. В. Кузьмин, Е. В. Хижнякова // Известия Иркутского государственного университета. Серия: Математика. — 2023. — № 45. — С. 54–72.

4. Клячин, В. А. Метод цепей для организации хранения многомерных триангуляций / В. А. Клячин, В. В. Попов // Вестник Волгоградского государственного университета. Серия 1, Математика. Физика. — 2013. — Т. 19, № 2. — С. 71–79.

5. Клячин, В. А. Оценки кусочно-линейной аппроксимации производных функций классов Соболева / В. А. Клячин // Известия Иркутского государственного университета. Серия: Математика. — 2024. — № 49. — С. 78–89.

6. Кузьмин, В. В. Расчет 3D-формы гиперупругого тела для моделей нелинейной теории упругости методом Ньютона / В. В. Кузьмин // Математическая физика и компьютерное моделирование. — 2024. — Т. 27, № 2. — С. 80–91. — DOI: https://doi.org/10.15688/mpcm.jvolsu.2024.2.7

7. Федотов, Р. В. Алгоритмы оптимизации фасетчатых моделей и методы их сетевой передачи / Р. В. Федотов // Вычислительные методы и программирование. — 2003. — Т. 4, № 2. — С. 47–57.

8. A 3D-Model Compression Method for Large Scenes / Zh. Ying, W. Lingling, D. Lieyun, Z. Cheng // 35-th International Symposium on Automation and Robotics in Construction (ISARC 2018). — 2018. — P. 986–993.

9. Bin, Sh. J. An Efficient Edge-Based Compression Algorithm for 3D Models with Holes and Handles / Sh. J. Bin, H. Y. Wen, S. Shawn // Journal of Information Science and Engineering. — 2006. — Vol. 22, № 2. — P. 401–423.

10. De Floriani, L. Compressing Triangulated Irregular Networks / L. De Floriani, P. Magillo, E. Puppo // Geoinformatica. — 2000. — Vol. 1, № 4. — P. 67–88.

11. Free Online App for Wavefront OBJ Files Compression. — URL: https://products.aspose.app/3d/compression/obj

12. Jingliang, P. Technologies for 3D Mesh Compression: A Survey / P. Jingliang, K. Chang-Su, C.-C. Jay Kuo // J. Vis. Commun. Image R. — 2005. — № 16. — P. 688–733.

REFERENCES

1. Kapustina S.V., Klyuev M.S. Konvertatsiya 3D-modeli metodom optimiziruyushchego szhatiya [Converting a 3D-Model Using the Optimizing Compression Method]. *Materialy konferentsii MIT-2011* [Proceedings MIT-2011] URL: https://conf.nsc.ru/files/conferences/MIT-2011/fulltext/50144/56662/kapustina.pdf

2. Klyachin V.A., Kuzmenko A.Yu. Algoritm rekonstruktsii trekhmernykh obyektov po dvum izobrazheniyam [Algorithm for Reconstruction of Three-Dimensional Objects From Two Images]. *Matematicheskaya fizika i kompyuternoe modelirovanie* [Mathematical Physics and Computer Simulation], 2024, vol. 27, no. 2, pp. 61-70. DOI: https://doi.org/10.15688/mpcm.jvolsu.2024.2.5

3. Klyachin V.A., Kuzmin V.V., Khizhnyakova E.V. Metod triangulyatsii dlya priblizhennogo resheniya variatsionnykh zadach nelineynoy teorii uprugosti [Triangulation Method for Approximate Solving of Variational Problems in Nonlinear Elasticity]. *Izvestiya Irkutskogo gosudarstvennogo universiteta. Seriya: Matematika* [Bulletin of Irkutsk State University. Series Mathematics], 2023, no. 45, pp. 54-72.

4. Klyachin V.A., Popov V.V. Metod tsepey dlya organizatsii khraneniya mnogomernykh triangulyatsiy [Chain Method for Organizing Storage of Multidimensional Triangulations]. *Vestnik Volgogradskogo gosudarstvennogo universiteta. Seriya 1: Matematika. Fizika* [Mathematical Physics and Computer Simulation], 2013, vol. 19, no. 2, pp. 71-79.

5. Klyachin V.A. Otsenki kusochno-lineynoy approksimatsii proizvodnykh funktsiy klassov Soboleva [Estimates for Piecewise Linear Approximation of Derivative Functions of Sobolev Classes]. *Izvestiya Irkutskogo gosudarstvennogo universiteta. Seriya: Matematika* [Bulletin of Irkutsk State University. Series Mathematics], 2024, no. 49, pp. 78-89.

6. Kuzmin V.V. Raschet 3D-formy giperuprugogo tela dlya modeley nelineynoy teorii uprugosti metodom Nyutona [Calculation of 3D Shape of Hyperelastic Body for Models of Nonlinear Elasticity Theory by Newton's Method]. *Matematicheskaya fizika i kompyuternoe modelirovanie* [Mathematical Physics and Computer Simulation], 2024, vol. 27, no. 2, pp. 80-91. DOI: https://doi.org/10.15688/mpcm.jvolsu.2024.2.7

7. Fedotov R.V. Algoritmy optimizatsii fasetchatykh modeley i metody ikh setevoy peredachi [Algorithms for Optimizing Faceted Models and Methods for Their Network Transmission]. *Vychislitelnye metody i programmirovanie* [Computational methods and programming], 2003, vol. 4, no. 2, pp. 47-57.

8. Ying Zh., Lingling W., Lieyun D., Cheng Z. A 3D-Model Compression Method for Large Scenes. *35-th International Symposium on Automation and Robotics in Construction (ISARC 2018)*, 2018, pp. 986-993.

9. Bin Sh.J., Wen H.Y., Shawn S. An Efficient Edge-Based Compression Algorithm for 3D Models with Holes and Handles. *Journal of Information Science and Engineering*, 2006, vol. 22, no. 2, pp. 401-423.

10. De Floriani L., Magillo P., Puppo E. Compressing Triangulated Irregular Networks. *Geoinformatica*, 2000, vol. 1, no. 4, pp. 67-88.

11. *Free Online App for Wavefront OBJ Files Compression*. URL: https://products.aspose.app/3d/compression/obj

12. Jingliang P., Chang-Su K., Jay Kuo C.-C. Technologies for 3D Mesh Compression: A Survey. *J. Vis. Commun. Image R.*, 2005, no. 16, pp. 688-733.

CHAIN ALGORITHM FOR COMPRESSING 3D-MODELS

Vladimir A. Klyachin

Doctor of Sciences (Physics and Mathematics),
Head of the Department of Computer Sciences and Experimental Mathematics,
Volgograd State University
klchnv@mail.ru, klyachin.va@volsu.ru
https://orcid.org/0000-0003-1922-7849
Prosp. Universitetsky, 100, 400062 Volgograd, Russian Federation

Abstract. The article describes in detail the algorithm for compressing information about the geometric structure of three-dimensional spatial models and multidimensional triangulations based on the use of adjacency of faces. This algorithm transforms a set of 3D-model faces into a list of chains sequentially located in space and adjacent to each other. Information compression occurs due to the absence of duplication of the numbers of vertices that form the model faces. The algorithm described in the article consists of three main parts. In the first part, a special graph of faces is constructed based on the set of model faces, the edges of the graph correspond to adjacent faces. Using the algorithm of traversal of graph vertices in depth, this graph is divided into simple chains. The second part of the algorithm transforms each chain of the graph into a sequence of numbers of vertices participating in the formation of the faces of this chain. The third part of the algorithm is designed to perform the reverse action – to translate the constructed sequence of vertex numbers back into sets of tuples consisting of the numbers of vertices that form the faces of the 3D-model. This algorithm is also extended to the case of spatial triangulations of polygonal areas. The software implementation of the algorithm for the special case of 3D-models is made in the form of built-in modules in the Blender program. The archives of the modules are freely available in the repository of the author of the article at https://github.com/KlyachinVA/LocFile.

Key words: triangulation, model graph, face chain, depth-first graph traversal, face adjacency.